



CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

PEDRO COLEN CARDOSO

Orientador: Prof. Flávio Coutinho
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

BELO HORIZONTE
MAIO DE 2015

PEDRO COLEN CARDOSO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

Trabalho de Conclusão de Curso apresentado ao Curso
de Engenharia da Computação do Centro Federal de
Educação Tecnológica de Minas Gerais.

Orientador: Flávio Coutinho
Centro Federal de Educação Tecnológica
de Minas Gerais – CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
MAIO DE 2015

PEDRO COLEN CARDOSO

OTIMIZANDO DESEMPENHO DE FRONT-END EM WEBSITES PARA HTTP2

Trabalho de Conclusão de Curso apresentado ao Curso
de Engenharia da Computação do Centro Federal de
Educação Tecnológica de Minas Gerais.

Trabalho aprovado. Belo Horizonte, 24 de novembro de 2014

Flávio Coutinho
Orientador

Co-Orientador

Professor
Convidado 1

Professor
Convidado 2

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
MAIO DE 2015

Espaço reservado para dedicatória. Inserir
seu texto aqui...

Agradecimentos

Inserir seu texto aqui... (esta página é opcional)

"You can't connect the dots looking forward you can only connect them looking backwards. So you have to trust that the dots will somehow connect in your future. You have to trust in something: your gut, destiny, life, karma, whatever. Because believing that the dots will connect down the road will give you the confidence to follow your heart, even when it leads you off the well worn path."(Steve Jobs)

Resumo

Abstract

Lista de Figuras

Figura 1 – Média de Bytes por Página por Tipo de Conteúdo em 2011	1
Figura 2 – Média de Bytes por Página por Tipo de Conteúdo em 2015	2
Figura 3 – Visão Geral do Protocolo HTTP	6
Figura 4 – HTTP com (a) múltiplas conexões e requisições sequenciais. (b) Uma conexão persistente e requisições sequenciais. (c) Uma conexão persistente e requisições em pipeline	11
Figura 5 – Multiplexação de <i>streams</i> no HTTP/2 (a) duas <i>streams</i> separadas (b) <i>streams</i> multiplexadas	15

Lista de Tabelas

Tabela 1 – Impacto do desempenho de <i>website</i> na receita.	3
--	---

Lista de Quadros

Quadro 1 – Etiquetas para cabeçalhos HTTP.	7
Quadro 2 – Métodos HTTP.	8
Quadro 3 – Códigos de estado HTTP.	8
Quadro 4 – Mudanças introduzidas no HTTP/1.1	12
Quadro 5 – Mudanças introduzidas no HTTP/2	16

Lista de Algoritmos

Lista de Abreviaturas e Siglas

AJAX	Asynchronous JavaScript and XML
ASCII	American Standard Code for Information Interchange
CDN	Content Delivery Network
CSS	Cascade Style Sheet
DNS	Domain Name System
IETF	Internet Engineering Task Force
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPbis	Hypertext Transfer Protocol bis
HTTPS	Hypertext Transfer Protocol Secure
HTTP-WG	Hypertext Transfer Protocol Working Group
KB	Kilobytes
MIME	Multi-Purpose Internet Mail Extensions
ms	Milissegundo
RFC	Request for Comments
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Sumário

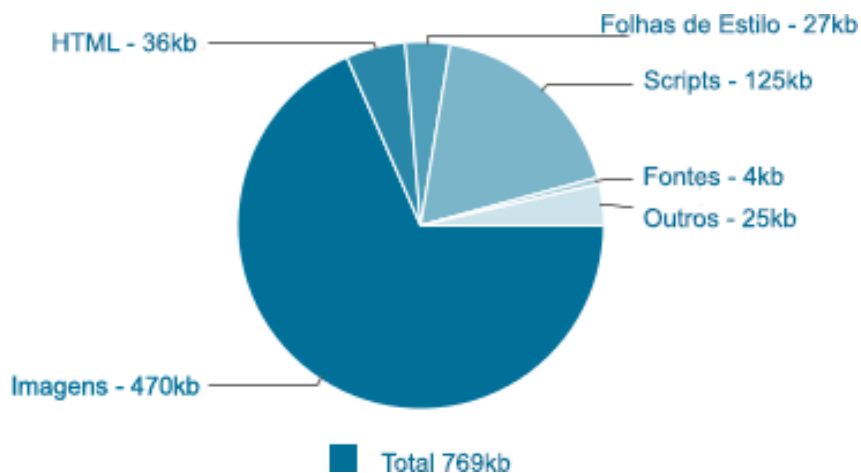
1 – Introdução	1
1.1 Motivação	3
1.2 Objetivos	3
2 – Fundamentação Teórica	5
2.1 O Protocolo HTTP	5
2.2 História	5
2.3 Visão Geral	6
2.4 HTTP/1.0 VS HTTP/1.1	9
2.5 HTTP/1.1 VS HTTP/2	13
3 – Trabalhos Relacionados	17
3.1 <i>High Performance Web Sites</i>	17
3.1.1 Regra 1: Faça menos requisições HTTP	18
3.1.2 Regra 2: Use Redes de Entrega de Conteúdo (CDN)	18
3.1.3 Regra 3: Adicione cabeçalhos de expiração	18
3.1.4 Regra 4: Utilize <i>gzip</i> no componentes	18
3.1.5 Regra 5: Coloque folhas de estilo no topo da página	18
3.1.6 Regra 6: Coloque <i>scripts</i> no fim da página	18
3.1.7 Regra 7: Evite expressões CSS	18
3.1.8 Regra 8: Faça arquivo <i>JavaScripts</i> e folhas de estilo externos	18
3.1.9 Regra 9: Reduza o número de pesquisas de DNS	18
3.1.10 Regra 10: Mimifique arquivo <i>JavaScript</i>	18
3.1.11 Regra 11: Evite redirecionamentos	18
3.1.12 Regra 12: Remova <i>scripts</i> duplicados	18
3.1.13 Regra 13: Configure <i>ETags</i>	18
3.1.14 Regra 14: Habilite <i>cache</i> para <i>AJAX</i>	18
4 – Metodologia	19
4.1 Delineamento da pesquisa	19
4.2 Coleta de dados	19
5 – Análise de Resultados	20
5.1 Situação atual	20
5.2 Análise dos dados coletados	20
6 – Conclusão	21

Referências	22
 Apêndices	 23
APÊNDICE A–Nome do Apêndice	24
APÊNDICE B–Nome do Apêndice	25
 Anexos	 26
ANEXO A–Nome do Anexo	27
ANEXO B–Nome do Anexo	28

1 Introdução

Desde que a *World Wide Web* foi proposta pelo cientista e pesquisador britânico Tim Bernes-Lee em 1989 ([CONNOLLY, 2000](#)), as páginas *web* vêm mudando de maneira acelerada. Nos últimos 4 anos, o tamanho médio de uma página *web* passou de 769kB para 2061kB, um expressivo aumento de 168%. Esse fato pode ser percebido observando os gráficos nas [Figura 1](#) e [Figura 2](#), gerados com a ajuda do [website](#) HTTP Archive¹. O primeiro foi gerado com dados de 15 de Abril de 2011 e o segundo com dados de 15 de Abril de 2015 e os dois mostram a média de *bytes* por página por tipo de conteúdo nas páginas *web*. Mas a informação mais relevante está localizado na parte debaixo do gráfico e mostra o tamanho médio de uma página *web* nas respectivas dadas.

Figura 1 – Média de Bytes por Página por Tipo de Conteúdo em 2011

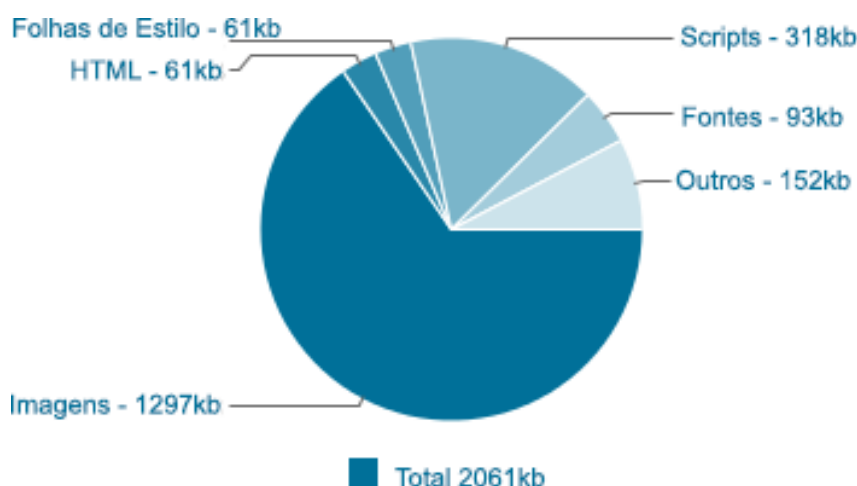


Fonte: Adaptado de [Archive \(2015a\)](#)

Apesar dessa grande mudança no tamanho das páginas (e consequentemente dos *websites*), a maneira como *websites* são entregues dos servidores para os clientes não sofreu nenhuma alteração desde 1999, ano de lançamento da RFC 2616 que especificou o HTTP/1.1 ([GROUP, 1999](#)). Como explicado por ([TANENBAUM, 2011](#)), o HTTP é um protocolo simples na camada de aplicação de requisições e respostas que é executado em cima da camada de transporte do protocolo TCP. O HTTP ficou famoso por ser fácil de entender e implementar e ao mesmo tempo cumprir sua função de transferência de recursos em rede com um bom desempenho. Contudo, o aumento no tamanho dos *websites* começou a fazer com que o tempo de resposta das páginas *web* ficasse muito grande e, como mudanças no HTTP seriam muito difíceis, pois teriam de envolver esforços de muitas partes interessadas na *World Wide Web* (como fabricantes de navegadores e

¹ <http://httparchive.org/>

Figura 2 – Média de Bytes por Página por Tipo de Conteúdo em 2015



Fonte: Adaptado de [Archive \(2015b\)](#)

mantenedores de servidores), os desenvolvedores passaram a ter de criar outras formas de resolver esse problema.

Técnicas de otimização de desempenho passaram a ser estudadas e implementadas por muitas empresas que queriam ter seus *websites* entregues mais rapidamente a seus clientes. Por muitos anos, a grande maioria dessas empresas focou seus esforços em otimizações para o *back-end*, principalmente para os seus servidores, o que hoje em dia pode ser considerado um erro. Como explicado por [Souders \(2007, p. 5\)](#) no que ele chamou de "Regra de Ouro do Desempenho": "Apenas 10-20% do tempo de resposta do usuário final são gastos baixando o documento HTML. Os outros 80-90% são gastos baixando todos os componentes da página."

Dessa forma ficou claro que técnicas de otimização de desempenho para o *front-end* dos *websites* deveriam se tornar prioridade quando procura-se melhorar o tempo de resposta para o usuário final. Para entender o quão importante esse tempo de resposta se tornou para empresas que baseiam seus negócios em vendas de serviços ou produtos na Internet, basta observar os dados da tabela [Tabela 1](#) expostos por Steve Souders na conferência Google I/O de 2009:

Tabela 1 – Impacto do desempenho de *website* na receita.

Empresa	Piora no tempo de resposta	Consequencia
Google Inc.	+500ms	-20% de tráfego
Yahoo Inc.	+400ms	-5% à -9% de tráfego
Amazon.com Inc.	a cada +100ms	-1% de vendas

Fonte: ([SOUDERS, 2009b](#))

Steve Souders tornou-se um grande evangelizador da área de otimização de

desempenho de *front-end* de *websites*. Em seus livros, *High Performance Websites* (SOU-[DERS, 2007](#)) e *Even Faster Websites* (SOU-[DERS, 2009a](#)) ele ensina técnicas de como tornar *websites* mais rápidos, focando nos componentes das páginas. E em 2012, ele lançou seu terceiro livro, *Web Performance Daybook* (SOU-[DERS, 2012](#)), como um guia para desenvolvedores que trabalham com otimização de desempenho de *websites*.

1.1 Motivação

Após mais de 15 anos sem mudanças, o protocolo HTTP (finalmente) receberá uma atualização. A nova versão do protocolo, chamada de HTTP2, teve sua especificação aprovada no dia 11 de Fevereiro de 2015, ([GROUP, 2015b](#)), e deverá começar a ser implantada, a partir de 2016. Muitas mudanças foram feitas com o objetivo de melhorar o desempenho e a segurança da *web*. Além disso, o HTTP2 foi desenvolvido para ser compatível com suas versões anteriores, não sendo necessárias mudanças em servidores e aplicações antigos para funcionar baseados no novo protocolo.

Com as novas funcionalidades do HTTP2 a caminho algumas coisas devem mudar na área de otimização de desempenho de *websites*. Como pode ser percebido em ([STENBERG, 2014](#)), o HTTP2 foi desenvolvido para melhorar o desempenho de todos os *websites* e aplicações *web*, e não apenas dos poucos que podem aplicar técnicas de otimização. Isso torna difícil uma previsão o resultado da aplicação de técnicas desenvolvidas para os protocolos HTTP/1.0 e HTTP/1.1. Acredita-se que algumas das técnicas antigas podem não apenas não melhorar o desempenho dos *websites* como podem acabar piorando o tempo de resposta para o usuário final.

No decorrer dos próximos anos o HTTP2 deve seguir o mesmo caminho do HTTP/1.1 e se tornar o protocolo mais utilizado da *web*. Apesar de todo o esforço do HTTPbis (grupo responsável por desenvolver a especificação do HTTP2) em desenvolver um protocolo que garanta o melhor desempenho de *websites* e aplicações, sempre é possível ser mais rápido se as medidas certas forem tomadas.

1.2 Objetivos

Este trabalho tem como objetivo analisar o comportamento de técnicas de otimização de desempenho de *websites* desenvolvidas para os protocolos HTTP/1.0 e HTTP/1.1 quando aplicadas a *websites* usando o protocolo HTTP2 e, se necessário, propor técnicas específicas para o novo protocolo.

Para realização do objetivo principal, os seguintes objetivos específicos foram determinados:

1. Fazer uma análise comparativa das versões do protocolo HTTP
2. Avaliar os ganhos de desempenho das técnicas propostas por Steve Souders ao aplicá-las ao HTTP2
3. Se necessário, propor novas técnicas de otimização de desempenho de *websites* específicas para o HTTP2

2 Fundamentação Teórica

Segue neste capítulo os principais conceitos necessários para a realização deste projeto. Através destes é possível entender as técnicas de otimização de desempenho usadas e as justificativas para a escolhas de tais técnicas.

2.1 O Protocolo HTTP



2.2 História

Em 1989 Tim Berners-Lee propôs uma rede de computadores global para ajudar a comunidade científica a compartilhar o conhecimento gerado em diferentes partes do mundo, acelerando assim o desenvolvimento tecnológico. Ao final de 1990 o cientista do CERN já tinha desenvolvido em um computador de seu laboratório o primeiro navegador *web* (chamado de WorldWideWeb), uma linguagem de marcação de hipertextos (o HTML) e um protocolo para a transferência de hipertextos (o HTTP).



O HTTP recebeu sua primeira documentação oficial em 1991, quando passou a ser chamado de HTTP/0.9. Como esclarece (GRIGORIK, 2013), todas as versões anteriores passaram a ser consideradas iterações com o objetivo de se chegar a versão HTTP/0.9, então todas elas acabaram recebendo o rótulo de HTTP/0.9.



Essa primeira versão do protocolo era extremamente simples. A ideia é explicada por (GRIGORIK, 2013) da seguinte maneira:

- Uma conexão TCP é aberta
- A requisição do cliente é uma cadeia simples de caracteres ASCII
- A resposta do servidor é uma torrente de caracteres ASCII
- A resposta do servidor é um HTML
- A conexão é fechada após a transferência do documento



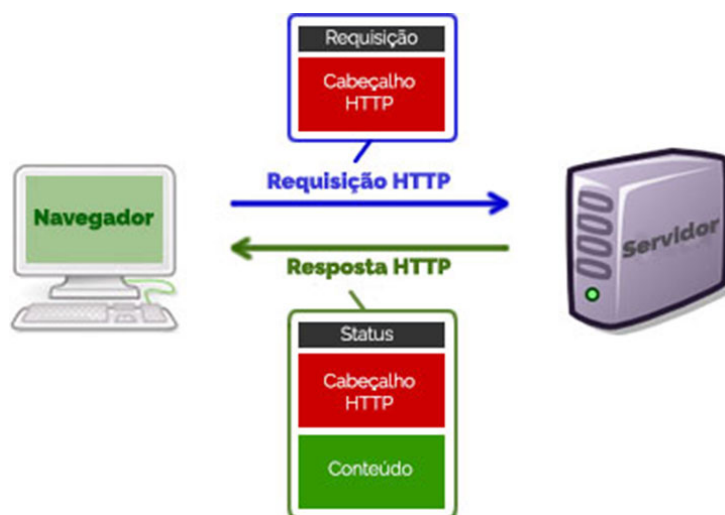
Com o passar dos anos a *World Wide Web* de Tim Berners-Lee cresceu rapidamente e isso fez com que o uso do HTTP aumentasse muito em pouco tempo. Viu-se a necessidade de um protocolo mais robusto e estruturado, mas que mantivesse a simplicidade do HTTP/0.9, pois esse era o segredo por trás de seu sucesso. Então o IETF passou a coordenar a criação de especificações para o HTTP e criou o HTTP-WG que tinha a função definir as especificações das versões seguintes do protocolo. Então,

em 1996 foi definido o HTTP/1.0 (GROUP, 1996), em 1999 o HTTP/1.1 (GROUP, 1999) e em 2015 foi aprovada a especificação para o HTTP2 (GROUP, 2015b).

2.3 Visão Geral

Como dito por (TANENBAUM, 2011), o HTTP é um simples protocolo de requisições e respostas que normalmente funciona em cima do protocolo TCP. As requisições e respostas são compostas por um cabeçalho e um conteúdo, e são enviadas do cliente para o servidor. O HTTP é um protocolo independente de estado, ou seja, cada tupla requisição-resposta pode ser tratada de maneira independente, sem que as anteriores e futuras interfiram nela. O modelo, ilustrado na Figura 3, é bem simples e direto, fácil de ser replicado.

Figura 3 – Visão Geral do Protocolo HTTP



Fonte: Adaptado de Saenz (2014)

O HTTP foi desenvolvido para funcionar na camada de Aplicação do protocolo TCP, conectando as ações do usuário à camada de Apresentação. Contudo, de acordo com (TANENBAUM, 2011), ele se transformou em um protocolo da camada de Transporte, criando uma maneira de processos se comunicarem através de diferentes redes. Hoje em dia não são apenas os navegadores *web* que utilizam o protocolo HTTP para se comunicar com servidores, tocadores de mídias, anti-vírus, programas de fotos, dentre outros tipos de aplicações utilizam o HTTP para recolher informações de servidores de maneira simples e rápida.



Os cabeçalhos HTTP definem características desejadas ou esperadas pelas aplicações e servidores, como tipo de codificação de caracteres ou tipo de compressão dos dados. Existem várias etiquetas padrões que podem ser utilizadas nos cabeçalhos e os desenvolvedores podem ainda criar etiquetas próprias para serem utilizadas dentro

das aplicações - por definição, caso um cliente ou servidor receba uma etiqueta que não reconhece ele simplesmente a ignora. No [Quadro 1](#) são apresentadas algumas das etiquetas mais utilizadas, mas existem muitas outras que não foram citadas e que podem variar com a versão do protocolo.

Quadro 1 – Etiquetas para cabeçalhos HTTP.

Etiqueta	Tipo	Conteúdo
<i>Accept</i>	Requisição	Tipo de páginas que o cliente suporta
<i>Accept-Encoding</i>	Requisição	Tipo de codificação que o cliente suporta
<i>If-Modified-Since</i>	Requisição	Data e hora para checar atualidade do conteúdo
<i>Authorization</i>	Requisição	Uma lista de credenciais do cliente
<i>Cookie</i>	Requisição	Cookie definido previamente enviado para o servidor
<i>Content-Encoding</i>	Resposta	Como o conteúdo foi codificado (ex. gzip)
<i>Content-Length</i>	Resposta	Tamanho da página em <i>bytes</i>
<i>Content-Type</i>	Resposta	Tipo de <i>MIME</i> da página
<i>Last-Modified</i>	Resposta	Data e hora que a página foi modificada pela última vez
<i>Expires</i>	Resposta	Data e hora quando a página deixa de ser válida
<i>Cache-Control</i>	Both	Diretiva de como tratar <i>cache</i>
<i>ETag</i>	Both	Etiqueta para o conteúdo da página
<i>Upgrade</i>	Both	O protocolo para o qual o cliente deseja alterar

Fonte: Adaptado de [Tanenbaum \(2011\)](#)

O conteúdo de uma resposta HTTP pode assumir diferentes formatos (HTML, CSS, JavaScript, [etc](#)), a definição desse formato é feita com a etiqueta *Content-Type* enviada no cabeçalho de resposta. O conteúdo é a maior parte de uma resposta HTTP e os desenvolvedores devem se esforçar para reduzi-lo ao máximo, garantindo que a comunicação de dados seja rápida e eficiente.

[Por rodar](#) em cima do protocolo TCP, o HTTP precisa que uma conexão TCP seja aberta para poder realizar a troca de dados entre o cliente e o servidor. Como essa conexão é gerenciada [vai depender](#) da versão do protocolo. Após a abertura da conexão, a requisição pode ser enviada. Na primeira linha da requisição são definidas a versão do protocolo e a operação que será realizada. Apesar de ter sido criado apenas para recuperar páginas *web* de um servidor, o HTTP foi intencionalmente desenvolvido de forma genérica, possibilitando a extensibilidade do seu uso. Sendo assim, o protocolo suporta diferentes operações, chamadas de métodos, além da tradicional requisição de páginas *web*. A lista completa de métodos com suas descrições pode ser vista no [Quadro 2](#), [vale](#) ressaltar que esses métodos são *case sensitive*, ou seja, o método *get*, por exemplo, não existe.

Dos citados no [Quadro 2](#), ~~os métodos~~ GET e POST são os mais utilizados pelos navegadores *web* e serão os mais utilizados neste trabalho. O método GET é utilizado para recuperar informações do servidor e o método POST para enviar informações para

Quadro 2 – Métodos HTTP.

Método	Descrição
GET	Ler página <i>web</i>
HEAD	Ler cabeçalho de página <i>web</i>
POST	Anexar à página <i>web</i>
PUT	Armazenar página <i>web</i>
DELETE	Remover página <i>web</i>
TRACE	Imprimir requisição de entrada
CONNECT	Conectar através de um <i>proxy</i>
OPTIONS	Listar opções para uma página <i>web</i>

Fonte: Adaptado de Tanenbaum (2011)

o servidor.

Sempre que uma requisição é enviada **ela recebe uma resposta**, mesmo que a requisição falhe. Na primeira linha do cabeçalho de resposta se encontra o código do estado da resposta em formato numérico de três dígitos. O primeiro dígito deste código define a qual sub-grupo ele pertence. O Quadro 3 mostra os sub-grupos existentes e o significado de cada um deles. Cada um destes sub-grupos possui vários códigos com diferentes significados e com a evolução do protocolo mais códigos foram sendo inseridos para lidar com necessidades específicas.

Quadro 3 – Códigos de estado HTTP.

Código	Significado	Exemplo
1xx	Informação	100 = servidor concorda em lidar com requisição do cliente
2xx	Sucesso	200 = sucesso na requisição; 204 = nenhum conteúdo presente
3xx	Redirecionamento	301 = página foi movida; 304 = <i>cache</i> ainda é válida
4xx	Erro do cliente	403 = página proibida; 404 = página não encontrada
5xx	Erro do servidor	500 = erro interno de servidor; 503 = tente novamente mais tarde

Fonte: Adaptado de Tanenbaum (2011)

A *cache* é uma característica importante do HTTP. O protocolo foi construído com suporte integrado para lidar com este requisito de desempenho. Os clientes e os servidores conseguem gerenciar *caches* com a ajuda dos cabeçalhos de requisição e resposta, mas o tamanho da *cache* é definido pelo navegador. O problema destas memórias locais é saber o momento de utilizar os dados armazenados nelas ou de pedir novos dados ao servidor. **Para isso existem variadas técnicas, como as propostas por (SOUDERS, 2007), que devem ser analisadas e utilizadas de acordo com a situação.**

As versões do protocolo HTTP foram aprimorando falhas identificadas quando ele passou a ser utilizado em larga escala. Mas apesar das mudanças a essência do protocolo continua a mesma: um simples protocolo de requisição e resposta. Nas próximas seções serão apresentadas comparações entre as versões do HTTP, com ênfase nas mudanças que ajudaram a melhorar a performance dos *websites* e aplicações *web*.

2.4 HTTP/1.0 VS HTTP/1.1

Com a popularização da Internet na década de 1990 o uso do HTTP estava crescendo rapidamente. Com isso o IETF teve que se apressar para criar um documento de consulta para desenvolvedores que queriam utilizar o protocolo em suas aplicações. Sendo assim, apesar de todo o debate por trás da (GROUP, 1996), aprovada em 1996, o documento apenas explicou os usos comuns do HTTP/1.0, mas não chegou a definir padrões de como utilizá-lo, como explica (KRISHNAMURTHY JEFFREY C. MOGUL, 1999). Por isso logo após a sua aprovação o HTTP-WG já começou a trabalhar na (GROUP, 1999), para poder corrigir os erros existentes no HTTP/1.0 com a criação de uma nova versão, o HTTP/1.1.

Várias funcionalidades importantes foram adicionadas no HTTP/1.1, e existiu uma preocupação muito grande com a compatibilidade entre as versões do protocolo, afinal de contas o HTTP/1.0 já era amplamente utilizado e não podia se esperar que todos os *websites* e aplicações se adaptassem de uma hora para outra. Esse fato também levou o HTTP-WG a criar um protocolo que suportasse mudanças futuras (lembrando que no HTTP todas as etiquetas que um cliente ou servidor não reconhecem são simplesmente ignoradas). Pensando na extensibilidade do protocolo as primeiras mudanças no protocolo HTTP/1.1 que valem ser citadas foram a criação de duas novas etiquetas para cabeçalhos, *Upgrade* - uma maneira do cliente informar qual versão do protocolo ele suporta - e *Via* - que define uma lista dos protocolos suportados pelos clientes ao longo do caminho de uma transmissão.

Como dito anteriormente o protocolo HTTP foi construído com suporte integrado para *cache*. Mas o mecanismo de *cache* do HTTP/1.0 era muito simples e não permitia que o cliente ou o servidor definisse instruções diretas de como a memória deveria ser utilizada. O HTTP/1.1 tentou corrigir esse problema com a criação de novas etiquetas para cabeçalhos. A primeira delas é a *ETag*, que define uma cadeia de caracteres única para um arquivo. Além do próprio conteúdo, essa cadeia utiliza a data e a hora da última modificação no arquivo, logo pode ser utilizada para verificar se dois arquivos são idênticos. O HTTP/1.1 também definiu novas etiquetas condicionais para complementar a já existente *If-Modified-Since*. As etiquetas *If-None-Match*, *If-Match* e *If-Unmodified-Since*, passaram a poder ser usadas para verificar se arquivos em *cache*

estão atualizados ou não. Uma das mudanças mais significativas no mecanismo de *cache*, foi a etiqueta *Cache-Control* que permite definir novas diretrizes para o uso da *cache*, como tempo de expiração relativos e arquivos que não devem ser armazenados.

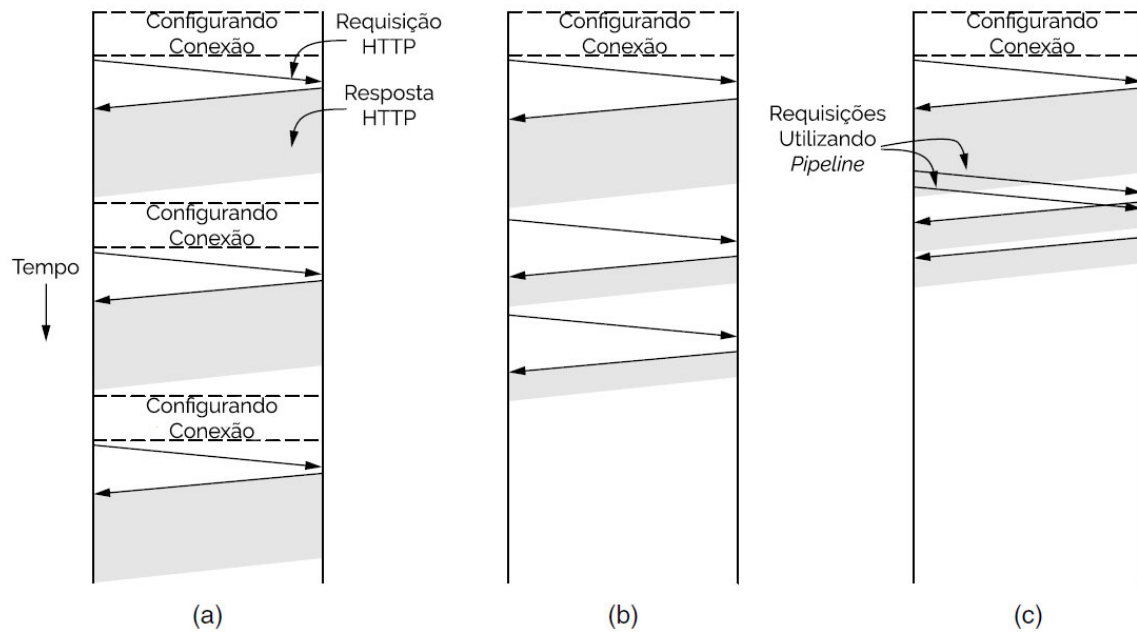
O HTTP/1.0 tinha muitos problemas em gerenciar a largura de banda, por exemplo não era possível enviar partes de arquivos, logo, mesmo se o cliente não precisasse de um arquivo inteiro, ele teria de recebê-lo. No HTTP/1.1 foram então criadas a etiqueta *Range*, o tipo *MIME multipart/byteranges* e o tipo de compressão *Chunked* para que cliente e servidores pudessem trocar mensagens com partes de arquivos. Para complementar esse novo mecanismo foi incluído um novo código de resposta, 100, que informava um cliente que o corpo de sua requisição deve ser enviado. Além de poder enviar partes de arquivos o HTTP/1.1 se preocupou em garantir a compressão dos dados durante todo o caminho da transmissão, então foi incluída a etiqueta *Transfer-Encoding*, que complementa a *Content-Encoding* indicando qual codificação foi utilizada na transmissão ponto a ponto.

O modelo original do protocolo HTTP utilizava uma conexão TCP para cada transmissão. Esse processo era extremamente danoso para o desempenho de *websites* e aplicações, pois se gastava muito tempo na criação e configuração de novas conexões e nos momentos iniciais da conexão (quando, por definição, ela é mais lenta). Para corrigir esse problema o HTTP/1.1 define conexões persistentes como seu padrão. Conexões persistentes permitem que clientes e servidores assumam que uma conexão TCP continuará aberta após a transmissão de dados, e que esta poderá ser utilizada para uma nova transmissão. Além disso foi definido que o HTTP/1.1 utilizaria *pipeline*, isto quer dizer que clientes não precisam aguardar a resposta de uma requisição para mandarem uma nova requisição, como era o padrão do HTTP/1.0. Os ganhos com essas novas técnicas podem ser notados na Figura 4.

Uma funcionalidade desejada no HTTP/1.1 era a de poder fazer requisições para servidores outros além da página principal sendo acessada. Isso possibilita que desenvolvedores hospedem arquivos CSS e JavaScript em um servidor e imagens em outro por exemplo. Isso se tornou possível com a adição da etiqueta *Host*, onde o cliente pode definir que é o caminho do servidor que será utilizado na requisição. Caso a etiqueta *Host* não esteja definida no cabeçalho é assumido que o caminho do servidor é o caminho da página principal.

Como conclui (KRISHNAMURTHY JEFFREY C. MOGUL, 1999): os protocolos HTTP/1.0 e HTTP/1.1 diferem em diversas maneiras. Enquanto muitas dessas mudanças têm o objetivo de melhorar o HTTP, a descrição do protocolo mais do que triplicou de tamanho, e muitas dessas funcionalidades foram introduzidas sem testes em ambientes reais. Esse aumento de complexidade causou muito trabalho para os desenvolvedores de clientes e servidores *web*.

Figura 4 – HTTP com (a) múltiplas conexões e requisições sequenciais. (b) Uma conexão persistente e requisições sequenciais. (c) Uma conexão persistente e requisições em pipeline




Fonte: Adaptado de [Tanenbaum \(2011\)](#)

No [Quadro 4](#) encontra-se um resumo das mudanças introduzidas no protocolo HTTP/1.1. ~~Observa-se que nem todas as mudanças mostradas no Quadro 4 foram detalhadas anteriormente, mas servem para ilustrar que existem ainda outras diferenças entre as versões 1.0 e 1.1 do protocolo que para os fins deste trabalho elas não são relevantes.~~



Quadro 4 – Mudanças introduzidas no HTTP/1.1

Cabeçalhos	
Etiqueta	Descrição
<i>Accept-Encoding*</i>	Lista de codificações aceitas
<i>Age</i>	O tempo que o conteúdo está salvo no <i>proxy</i> em segundos
<i>Cache-Control</i>	Notifica todos os mecanismos de <i>cache</i> do cliente ao servidor se o conteúdo deve ser salvo
<i>Connection</i>	Controla a conexão atual
<i>Content-MD5</i>	Codificação binária de base-64 para verificar conteúdo da resposta
<i>Etag</i>	Identificador único de um conteúdo
<i>Host</i>	Indica o endereço e a porta do servidor que deve ser utilizado pela mensagem
<i>If-Match</i>	Realize a ação requisitada se, e somente se, o conteúdo do cliente é igual ao conteúdo do servidor
<i>If-None-Match</i>	Retorna código 304 se o conteúdo não foi modificado
<i>If-Range</i>	Se o conteúdo não foi modificado, envie a parte solicitada, se não, envie o conteúdo novo
<i>If-Unmodified-Since</i>	Envie o conteúdo se, e somente se, ele não foi modificado na data esperada
<i>Proxy-Authentication</i>	Pede uma requisição de autenticação de um <i>proxy</i>
<i>Proxy-Authorization</i>	Credenciais de autorização para se conectar a um <i>proxy</i>
<i>Range</i>	Faz a requisição de apenas uma parte de um conteúdo
<i>Trailer</i>	Indica que o grupo de cabeçalhos está presente em uma mensagem
<i>Transfer-Encoding</i>	A forma de codificação usada para se transferir o conteúdo para o usuário
<i>Upgrade</i>	Pede para o servidor atualizar para outro protocolo
<i>Vary</i> 	Informa o cliente sobre <i>proxies</i> pelos quais a resposta passou
<i>Via</i>	Informa o servidor de <i>proxies</i> pelos quais a requisição passou
<i>Warning</i>	Mensagem genérica de cuidado para possíveis problemas no corpo da mensagem
<i>WWW-Authenticate</i>	Indica tipo de autenticação que deve ser utilizada para acessar entidade requerida
Métodos	
Método	Descrição
<i>OPTIONS</i>	Requer informações sobre os recursos que o servidor suporta
Estados**	
Código	Descrição
100	Confirma que o servidor recebeu o cabeçalho de requisição e que o cliente deve continuar a enviar a mensagem desejada
206	O servidor está entregando apenas uma parte de um conteúdo por causa da etiqueta de <i>Range</i> na requisição do cliente
300	Indica as múltiplas opções disponíveis para o cliente
409	Indica que a requisição não pôde prosseguir por causa de um conflito
410	Indica que o conteúdo requisitado não está mais disponível e não estará disponível no futuro
Diretivas	
Diretiva	Descrição
<i>chunked</i>	Utilizado para envie de conteúdo em partes
<i>max-age</i>	Determina qual é o tempo máximo que um conteúdo deve ficar salvo em <i>cache</i>
<i>no-store</i>	Indica que o conteúdo não deve ser salvo em <i>cache</i>
<i>no-transform</i>	Indica que o conteúdo não deve ser modificado por <i>proxies</i>
<i>private</i>	Indica que o conteúdo não deve ser acessado sem autenticação
Tipos de MIME	
Tipo	Descrição
<i>multipart/byteranges</i>	Indica que o conteúdo que está sendo enviado é apenas uma parte de um todo
Funcionalidades	
Nome	Descrição
<i>Content negotiation</i>	Escolhe a melhor representação disponível para um conteúdo
<i>Persistent connection</i>	Após o término de uma requisição HTTP a conexão continua aberta e pode ser utilizada por outras requisições
<i>Pipeline</i>	O cliente não precisa esperar que a resposta de uma requisição retorne antes de enviar outra requisição

2.5 HTTP/1.1 VS HTTP/2

Com o lançamento do HTTP/1.1 apenas 3 anos depois do HTTP/1.0, o protocolo conseguiu se firmar e provou que seria o protocolo dominante da Internet. O HTTP/1.1 era robusto e flexível, permitindo que fosse utilizado em aplicações diversas de maneira eficiente. Com o passar dos anos, o IETF acrescentou algumas extensões ao protocolo para corrigir erros pontuais, mas o HTTP suportava as necessidades da rede mundial de computadores.

No início do século XXI, os *websites* começaram a mudar. Eles passaram a se tornar mais complexos e consequentemente maiores, muitas fontes e folhas de estilo eram utilizadas e cada página passou a possuir vários arquivos de JavaScript. Além disso, eles deixaram de ser estáticos e passaram a responder dinamicamente às ações dos usuários. Hoje em dia, muitas requisições HTTP são necessárias para se montar uma página *web* e essas requisições podem ser longas e demoradas. Finalmente, o HTTP/1.1 passou a ser visto como um gargalo de desempenho para *websites*, então em 2007 o IETF formou o grupo HTTPbis (onde o "*bis*" quer dizer "de novo" em latim). Mas o grupo só começou as discussões sobre a nova versão do protocolo em 2012, terminando de redigir as especificações em 2014. Após revisões, a especificação oficial da nova versão do HTTP ~~vou~~ foi aprovada no início do ano de 2015 e deve começar a ser utilizada em 2016. A nova versão do HTTP passou a se chamar HTTP/2, sem os decimais, caso mudanças sejam necessárias será lançada uma nova versão completa do protocolo.


O HTTP/2 veio para corrigir o problema de latência existente na versão anterior. Apesar do sistema de *pipeline*, o HTTP/1.1 é muito sensível à latência, ou seja, apesar de conseguir entregar ~~muitos dados~~, existem problemas quanto ao tempo de viagem das requisições e respostas. Isso acontece porque o *pipeline* do HTTP/1.1 é muito difícil de ser gerenciado e muitas vezes fecha conexões que deveriam ter ficado abertas. O problema é tão grande que (STENBERG, 2014) afirma que mesmo nos dias de hoje, muitos navegadores *web* preferem desativar o *pipeline*. Para corrigir este problema, o HTTP/2 propõe mudanças na forma como as informações são enviadas. O mais importantes destas mudanças é que, assim como ocorreu das versões 1.0 para 1.1, elas não podem alterar nenhum paradigma já existente no protocolo. As aplicações que utilizam o HTTP/1.1 devem continuar funcionando no HTTP/2, os formatos de arquivos, as URL e as URI devem ser mantidos e o usuário final não deve ter de fazer nada para poder aproveitar das melhorias do novo protocolo. Com isso, para tentar criar um protocolo que funcionasse no mundo real tanto quanto no teórico, o HTTPbis decidiu se inspirar no protocolo SPDY, criado pela Google, por este já ter se provado um conceito funcionou.

A primeira mudança no HTTP/2 está na forma como ele escreve suas requisições e respostas. Em suas versões anteriores, o protocolo optou por utilizar o formato

ASCII para estruturar suas requisições e respostas, mas era difícil separar as partes dos cabeçalhos e tratar espaços em branco indesejados. Para resolver esse problema, o HTTP/2 é um protocolo binário. Assim é mais simples quebrar requisições e respostas em quadros, ~~comparar quadros~~ e comprimir as informações. Entre as desvantagens dessa representação binária estão o fato de que os cabeçalhos HTTP não serão mais compreensíveis sem a ajuda de ferramentas e que a depuração do protocolo dependerá de analisadores de pacotes.

Utilizando a apresentação binária, o HTTP/2 possibilita a multiplexação de **streams** de dados. Como explica (STENBERG, 2014) ~~Uma stream é~~ uma associação lógica criação por uma sequência de quadros. No HTTP/2 uma conexão possui ~~várias streams~~ e por isso vários componentes podem ser transferidos ao mesmo tempo. Para esse processo funcionar, o protocolo multiplexa ~~essas streams~~ no momento do envio e as separa novamente na chegada. A Figura 5 ilustra a multiplexação de ~~streams~~.

Um problema no HTTP/1.1 era a garantia de que um componente A já teria sido baixado quando outro componente B que depende de A fosse executado. Apesar de o HTML garantir isso, essa limitação impedia que a paralelização de *downloads* fosse maior. No HTTP/2 ~~as~~ foram adicionados os mecanismos de prioridade e dependência. ~~Com eles é fácil~~ indicar quais ~~streams~~ devem ser baixados primeiro e quais são ~~as dependências entre elas~~. Dessa forma, os desenvolvedores podem paralelizar ao máximo seus componentes e o protocolo cuidará de evitar erros.



A compressão de dados é um fator importante para o aumento de desempenho do HTTP. Com o passar dos anos, as requisições e respostas aumentaram de tamanho, e as **compressões** criadas para o SPDY e o **HTTPS** não se provaram **eficientes** contra **ataques de terceiros**. Logo notou-se que este era um ponto crítico para a nova versão do protocolo, então o HTTPbis decidiu por criar o HPACK, ~~que será o~~ novo formato de compressão para cabeçalhos HTTP/2. Visando a robustez desse formato, foi criada uma nova especificação ~~exclusivamente~~ para o HPACK, que detalha como ele funciona e como deve ser usado (GROUP, 2015a).

Assim como na versão anterior, o HTTP/2 possui mecanismos para lidar com a *cache*. As etiquetas existentes no HTTP/1.1 continuam a existir, mas uma nova funcionalidade foi adicionada na nova versão, o *Server Push*. **A ideia por trás é** garantir que o cliente consiga componentes da maneira mais rápida. Então se o cliente requisitar um componente X, o servidor pode saber que é provável que ele vá requisitar o componente Y nos próximos **instantes, sendo** assim o servidor pode enviar o componente Y antes mesmo de receber o pedido por ele. Essa funcionalidade é algo que o cliente deverá especificar explicitamente, mas existe grande expectativa ~~quando as~~ melhorias que ela pode trazer. Além dessa melhoria no sistema de ~~cache~~, o HTTP/2 inclui uma nova etiqueta para impedir o desperdício de banda de transmissão. Se o servidor começar a

Figura 5 – Multiplexação de ~~streams~~ no HTTP/2 (a) ~~duas streams~~ separadas (b) ~~streams~~ multiplexadas



(a)



(b)

Fonte: Adaptado de [Stenberg \(2014\)](#)

enviar um componente com um tamanho específico e perceber que aquele componente não é mais útil, ele pode cancelar o envio com a etiqueta *RST STREAM*, evitando que a ~~stream~~ de envio fique ocupada com dados desnecessários por muito tempo.

Caso um *website* ou uma aplicação deseje transferir o cliente para outro servidor sem ser o requisitado, ou até mesmo para outra porta, ele poderá utilizar a etiqueta *Alt-Svc*. Com essa etiqueta o servidor informa ao cliente para onde ele deve ir, então o cliente deve tentar se conectar de maneira assíncrona no caminho sugerido pelo *Alt-Svc* e utilizar aquela conexão apenas se ela se provar confiável. Essa etiqueta foi criada com a intenção de informar clientes que os dados requisitados estão disponíveis também em

um servidor seguro.

~~Como explicado anteriormente, o HTTP/2 utiliza de *streams* para enviar e receber dados.~~ Esta técnica foi escolhida para aumentar a velocidade de envio do protocolo e a quantidade de dados que pode ser enviado de uma só vez. Cada uma dessas *streams* possui sua própria janela de fluxo independente, o que garantirá que se uma *stream* falhar as outras continuem funcionando. Para impedir o envio de dados e parar todas as *streams* abertas, o protocolo inclui a etiqueta *BLOCKED*, que informa que existem dados para serem enviados, mas algo está impedindo o processo de continuar.

O protocolo HTTP/2 trás grandes mudanças na estrutura dos dados que serão enviados e recebidos. A essência continua a mesma, um protocolo de requisições e respostas, mas a medida que o protocolo evolui novas formas de garantir o desempenho dos *websites* e das aplicações estão sendo acrescentadas ao HTTP. Neste trabalho foram detalhadas as mudanças que visaram melhoria de desempenho e que podem afetar a forma como a otimização de desempenho de *websites* é feita. O documento completo com toda a especificação pode ser encontrado em (GROUP, 2015b). O Quadro 5 mostra um resumo das mudanças detalhadas acima.

Quadro 5 – Mudanças introduzidas no HTTP/2

Funcionalidade	Descrição
HTTP/2 binário	Ao invés de utilizar caracteres ASCII para representar informações, o HTTP/2 é binário, o que facilita a comparação de informações, o envio de dados e outras funcionalidades
<i>Streams</i> multiplexadas	Se existem dois componentes para serem enviados, o protocolo pode optar por multiplexá-los em uma única <i>stream</i> e enviar os dois ao mesmo tempo
Prioridades e dependências	O cliente pode definir quais componentes possuem prioridade para serem baixados primeiro. Além disso, pode se existem dependências entre os componentes para garantir que quando um arquivo seja baixado todos os outros necessários para o seu funcionamento já tenham sido baixados
<i>HPACK</i>	Novo sistema de compressão de cabeçalhos para o HTTP/2
<i>RST_STREAM</i>	Uma maneira de cancelar o envio de componentes
<i>Server Push</i>	Habilidade do servidor de enviar um arquivo X para o cliente caso ele veja como provável que o cliente vai precisar desse arquivo no futuro próximo
Fluxo individual de <i>streams</i>	Cada <i>stream</i> de envio possui sua própria janela e pode ser gerenciada individualmente, assim caso uma falhe as outras continuam
<i>BLOCKED</i>	Forma do cliente ou servidor informar a outra parte que existe algo impedindo que o envio de dados continue
Alt-Svc	O servidor pode informar ao cliente de caminhos alternativos para acessar os dados requisitados

3 Trabalhos Relacionados

Desenvolvedores estão sempre a procura de maneiras para torna suas aplicações mais rápidas aos olhos dos usuários finais, isto não foi diferente com os *websites*. A área de otimização de desempenho para páginas *web* é tão antiga quanto a *World Wide Web*, mas por muito tempo os esforços foram dedicados à otimização de *back-end* o que (SOUDERS, 2007) provou não ser o melhor caminho para se atingir o objetivo desejado. De acordo com Souders (2007, p. 5): "Apenas 10-20% do tempo de resposta do usuário final são gastos baixando o documento HTML. Os outros 80-90% são gastos baixando todos os componentes da página." Então os desenvolvedores podem obter resultados mais significativos caso trabalhem para otimizar o desempenho do *front-end* de seus *websites* ao invés do *back-end*.

Neste capítulo serão apresentadas as ideias principais dos dois livros de Steve Souders que tornaram o conhecimento sobre a otimização de desempenho de *front-end* de *websites* acessíveis a todos os desenvolvedores e ajudaram a melhorar a *web* nos últimos anos.

3.1 *High Performance Web Sites*

Neste livro, publicado em Setembro de 2007, Steve Souders descreve 14 regras para a otimização de desempenho de *front-end* de *websites*. De acordo com Souders (2007, p. 5): "Se você seguir todas as regras aplicáveis à seu *website*, você vai fazer sua página 25-50% rápida e melhorar a experiencia do usuário.". As regras são muito variadas e cobrem configurações básicas de servidores, redução do tamanho de arquivos e melhores práticas de programação.

A seguir serão explicadas as 14 regras encontradas no livro, ordenadas, de acordo com Steve Souders, das que causam maior impacto no desempenho às que causam menor impacto.

- 3.1.1 Regra 1: Faça menos requisições HTTP
- 3.1.2 Regra 2: Use Redes de Entrega de Conteúdo (CDN)
- 3.1.3 Regra 3: Adicione cabeçalhos de expiração
- 3.1.4 Regra 4: Utilize *gzip* no componentes
- 3.1.5 Regra 5: Coloque folhas de estilo no topo da página
- 3.1.6 Regra 6: Coloque *scripts* no fim da página
- 3.1.7 Regra 7: Evite expressões CSS
- 3.1.8 Regra 8: Faça arquivo *JavaScripts* e folhas de estilo externos
- 3.1.9 Regra 9: Reduza o número de pesquisas de DNS
- 3.1.10 Regra 10: Mimifique arquivo *JavaScript*
- 3.1.11 Regra 11: Evite redirecionamentos
- 3.1.12 Regra 12: Remova *scripts* duplicados
- 3.1.13 Regra 13: Configure *ETags*
- 3.1.14 Regra 14: Habilite *cache* para *AJAX*

4 Metodologia

Inserir seu texto aqui...

4.1 Delineamento da pesquisa

Inserir seu texto aqui...

4.2 Coleta de dados

Inserir seu texto aqui...

5 Análise de Resultados

Inserir seu texto aqui...

5.1 Situação atual

Inserir seu texto aqui...

5.2 Análise dos dados coletados

Inserir seu texto aqui...

6 Conclusão

Referências

- ARCHIVE, H. **Http Archive Abril 2011 Query**. 2015. Disponível em: <<http://httparchive.org/interesting.php?a=All&l=Apr%2015%202011>>. Citado na página 1.
- ARCHIVE, H. **Http Archive Abril 2015 Query**. 2015. Disponível em: <<http://httparchive.org/interesting.php?a=All&l=Apr%2015%202015>>. Citado na página 2.
- CONNOLLY, D. **The birth of the web**. 2000. Disponível em: <<http://www.w3.org/History.html>>. Citado na página 1.
- GRIGORIK, I. **High Performance Browser Networking**. 1st. ed. [S.l.]: O'Reilly, 2013. Citado na página 5.
- GROUP, H. W. **HPACK - Header Compression for HTTP/2**. 2015. Disponível em: <<http://tools.ietf.org/html/draft-ietf-httpbis-header-compression-12>>. Citado na página 14.
- GROUP, H. W. **Hypertext Transfer Protocol version 2**. 2015. Disponível em: <<https://tools.ietf.org/html/draft-ietf-httpbis-http2-17>>. Citado 3 vezes nas páginas 3, 6 e 16.
- GROUP, N. W. **RFC 1945**. 1996. Disponível em: <<http://tools.ietf.org/html/rfc1945>>. Citado 2 vezes nas páginas 6 e 9.
- GROUP, N. W. **RFC 2616**. 1999. Disponível em: <<http://tools.ietf.org/html/rfc2616>>. Citado 3 vezes nas páginas 1, 6 e 9.
- KRISHNAMURTHY JEFFREY C. MOGUL, D. M. K. B. Key differences between http/1.0 and http/1.1. **Elsevier Science B.V**, 1999. Citado 2 vezes nas páginas 9 e 10.
- SAENZ, J. **Building Web Apps with Go**. [s.n.], 2014. Disponível em: <<https://www.gitbook.com/book/codegangsta/building-web-apps-with-go/details>>. Citado na página 6.
- SOUDERS, S. **High Performance Web Sites**. 1st. ed. [S.l.: s.n.], 2007. Citado 3 vezes nas páginas 2, 8 e 17.
- SOUDERS, S. **Even Faster Web Sites**. 1st. ed. [S.l.: s.n.], 2009. Citado na página 2.
- SOUDERS, S. Even faster websites. In: . [s.n.], 2009. Disponível em: <https://www.youtube.com/watch?feature=player_embedded&v=aJGC0JSIpPE>. Citado na página 3.
- SOUDERS, S. **Web Performance Daybook**. 1st. ed. [S.l.: s.n.], 2012. Two. Citado na página 3.
- STENBERG, D. Http2 explained. **ACM SIGCOMM Computer Communication Review**, 2014. Disponível em: <<http://daniel.haxx.se/http2>>. Citado 4 vezes nas páginas 3, 13, 14 e 15.
- TANENBAUM, D. J. W. A. S. **Computer Networks**. 5th. ed. [S.l.: s.n.], 2011. Citado 5 vezes nas páginas 1, 6, 7, 8 e 11.

Apêndices

APÊNDICE A – Nome do Apêndice

Inserir seu texto aqui...

APÊNDICE B – Nome do Apêndice

Inserir seu texto aqui...

Anexos

ANEXO A – Nome do Anexo

Inserir seu texto aqui...

ANEXO B – Nome do Anexo

Inserir seu texto aqui...