UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JAIME FREIRE DE SOUZA

AVALIAÇÃO DO PROTOCOLO HTTP 2.0

VITÓRIA DA CONQUISTA - BA
2015

JAIME FREIRE DE SOUZA

AVALIAÇÃO DO PROTOCOLO HTTP 2.0

Monografia apresentada ao Curso de Graduação em Ciência da Computação da Universidade Estadual do Sudoeste da Bahia, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.
Orientador(a): Prof. Me. Stenio Longo Araújo

VITÓRIA DA CONQUISTA - BA

2015

JAIME FREIRE DE SOUZA

AVALIAÇÃO DO PROTOCOLO HTTP 2.0

Aprovado em ____/____/_____

BANCA EXAMINADORA

_____

_____

_____

RESUMO

A Internet cresceu muito desde sua criação. Grande parte deste crescimento se deve a *World Wide Web*. Neste sentido, tem-se o protocolo HTTP como um dos protocolos mais importantes da grande rede mundial de computadores. Ao longo dos anos o ambiente de atuação do HTTP se tornou cada vez mais complexo, exigindo a evolução do protocolo. Com isso, diferentes versões do HTTP foram desenvolvidas até chegar a versão atual conhecida como HTTP 2.0. Essa nova versão do protocolo surge com objetivo de tornar a Web mais rápida e segura em meio a uma crescente demanda por recursos e alto desempenho. Este trabalho objetiva realizar um estudo e avaliação desta nova versão do protocolo HTTP, fazendo um comparativo com a versão anterior, para analisar se de fato, o HTTP 2.0 oferece um serviço com melhor performance. Na metodologia, foi realizada uma pesquisa bibliográfica sobre o protocolo HTTP 2.0, procurando identificar as suas características e também foram desenvolvidas páginas Web com diferentes níveis de complexidade. Essas páginas foram acessadas utilizando ambos os protocolos para identificar qual versão possui melhor desempenho no carregamento da página. Como resultado, concluiu-se que o HTTP 2.0 melhora significativamente a performance da Web, pois apresenta melhor desempenho do que HTTP 1.1 no carregamento de páginas que possuem muitos objetos.

Palavras-chave: World Wide Web; Protocolo HTTP; HTTP 2.0; Desempenho do HTTP 2.0.

ABSTRACT

The Internet has grown since its beginning. Much of this growth is due to World Wide Web. In this sense, we have the HTTP protocol as one of the most important protocols of the Internet. Over the years the HTTP execution environment has become increasingly complex, requiring the evolution of the protocol. Thus, different HTTP versions were developed to get the current version known as HTTP 2.0. This new version of the protocol appears in order to make Web faster and more secure amid a growing demand for resources and high performance. This work aims to conduct a study and evaluation of this new version of the HTTP protocol, comparing it with the previous version, to see whether in fact the HTTP 2.0 offers a service with better performance. In the methodology, a literature research about the HTTP 2.0 protocol was carried out, trying to identify their design and technical gols, and have also developed Web pages with different levels of complexity. These pages were accessed using both protocols to identify which version has better performance on page load. As a result, it was found that HTTP 2.0 significantly improves the performance of the web, since it presents better performance than HTTP 1.1 in loading pages that have many objects.

Keywords: World Wide Web; HTTP protocol; HTTP 2.0; HTTP 2.0 performance.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ARPA	*Advanced Research Projects Agency*
ASCII	*American Standard Code for Information Interchange*
CR	*Carriadge Return*
CSS	*Cascading Style Sheets*
GZIP	*GNU zip*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IETF	*Internet Engineering Task Force*
JPEG	*Joint Photographics Experts Group*
JS	*Javascript*
KB	*Kilobyte*
LF	*Line Fead*
MIME	*Multipurpose Internet Mail Extension*
RFC	*Request for Comments*
SPDY	*Speedy*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

O protocolo HTTP *(HyperText Transfer Protocol)* é um dos protocolos da camada de aplicação mais largamente utilizados na Internet. Desde sua publicação, o HTTP se tornou o principal agente impulsionador para o crescimento sem precedentes da grande rede mundial de computadores. É o protocolo que viabiliza o desenvolvimento e uso de uma gama de aplicações e serviços acessados por bilhões de dispositivos de diferentes formatos e tamanhos (GRIGORIK, 2013).

Considerado um protocolo bastante simples no início, fato que contribuiu para o sucesso da Web, o HTTP passou por um processo de evolução ao longo dos anos, tornando-se cada vez mais complexo. (MOGUL, 2002).

A evolução do HTTP foi impulsionada pelas mudanças ocorridas no próprio ambiente da Web. Surgiram novos serviços e aplicações mais robustas e complexas. Além disso, o protocolo teve que se adaptar ao crescimento da rede e às novas formas de conectividade.

Como afirma Salam *et al.* (2014), os novos paradigmas para aplicações web motivaram nos últimos anos, uma série de propostas e discussões para atualização dos protocolos de Internet. Surge então o HTTP 2.0, a nova versão do protocolo HTTP, que apresenta uma série de mudanças e melhorias, mas preserva a semântica das aplicações da versão anterior do protocolo.

De acordo com Grigorik (2015), os objetivos primários do HTTP 2.0 são a redução da latência, minimização do *overhead* do protocolo, suporte para priorização de requisições e otimização do envio dos recursos. Neste sentido a proposta do protocolo não se restringe apenas ao oferecimento de um serviço para transferência de página. Essa nova versão do HTTP se propõe a oferecer um serviço com a melhor performance.

Entre esses objetivos do protocolo, pode-se destacar a redução da latência como algo fundamental para a melhoria do desempenho no carregamento das páginas Web. Sieminski (2005) define a latência como o tempo entre a solicitação de uma página e sua total renderização no navegador. Em teoria, os mecanismos e mudanças do protocolo tendem a diminuir esse tempo, tornando as aplicações Web mais rápidas.

Surge, portanto, a necessidade de uma avaliação acerca da eficiência do protocolo HTTP 2.0 para verificar, se realmente, o protocolo cumpre com os objetivos propostos em sua especificação.

O objetivo deste trabalho é avaliar o desempenho do HTTP 2.0, fazendo um comparativo em relação ao HTTP 1.1. Esse tipo de avaliação consiste em testar o protocolo, utilizando páginas Web de diferentes tamanhos com o intuito de analisar o comportamento do protocolo à medida que a complexidade da página aumenta.

A grande maioria dos servidores Web ainda utilizam a versão 1.1 do HTTP, mas já é notável uma crescente utilização do HTTP 2.0. O processo de atualização do protocolo por parte dos servidores se dará de forma gradativa, pois há todo um legado do HTTP 1.1. Desta forma, os desenvolvedores têm as duas opções da versão do protocolo para utilizar. É necessário, portanto, avaliar se há a necessidade de uma imediata migração dos serviços já em uso ou de novos serviços para o protocolo HTTP 2.0. Pretende-se identificar as situações em que a utilização do HTTP 2.0 é mais vantajosa, ou seja, apresenta um melhor desempenho.

1.1 Objetivos

1.1.1 Objetivo Geral

Avaliar o desempenho do protocolo HTTP 2.0.

1.1.2 Objetivos Específicos

- Pesquisar o funcionamento do protocolo HTTP 1.1;
- Pesquisar o funcionamento do protocolo HTTP 2.0;
- Pesquisar ferramentas para análise de carregamento de páginas Web;
- Comparar o desempenho do HTTP 2.0 em relação ao HTTP 1.1;
- Analisar os resultados obtidos.

1.2 Metodologia

De acordo com Gil (2010), as pequisas podem ser classificadas em dois grandes grupos: razões de ordem intelectual e razões de ordem prática. Este trabalho apresenta as duas características, pois utiliza-se de um estudo para o conhecimento do objeto de pesquisa, bem como, pretende-se obter conclusões práticas como resultado do trabalho.

No processo de desenvolvimento deste trabalho, inicialmente foi realizada uma pesquisa bibliográfica com base em material já publicado. Como afirma Gil (2010), praticamente toda pesquisa acadêmica requer em algum momento a realização de trabalho que pode ser caracterizado como pesquisa bibliográfica. Essa metodologia fornece ao pesquisador a fundamentação teórica necessária para o conhecimento do objeto pesquisado, além de fornecer o histórico de evolução do ambiente deste objeto.

No contexto desta pesquisa que objetiva avaliar um protocolo de Internet há a necessidade do estabelecimento de métricas de avaliação e também do uso de ferramentas que permitam coletar os dados necessários. Para tanto, foram desenvolvidas páginas Web com diferentes características, com o intuito de analisar o comportamento do objeto estudado em aplicações com diferentes quantidades de recursos.

Foi definido um ambiente de testes que consiste em um servidor Web e um navegador, conectados via Internet. As páginas foram testadas utilizando as duas versões em uso do protocolo estudado e os dados foram captados por uma ferramenta inclusa no navegador. Os dados obtidos foram inseridos em uma planilha para geração dos gráficos, o que permitiu uma análise visual do comportamento.

1.3 Estrutura do Trabalho

O capítulo 2 apresenta os conceitos relacionados ao protocolo HTTP, bem como, sua evolução até a versão 1.1. O capítulo 3 apresenta uma visão geral do HTTP 2.0. O capítulo 4 discorre sobre o processo de testes e demonstra os resultados obtidos. Por fim, a conclusão e os trabalhos futuros são apresentados no capítulo 5.

2 REFERENCIAL TEÓRICO

2.1 A Internet

A Internet é um conjunto de redes de computadores, uma infraestrutura que configura-se como um complexo sistema de engenharia, provavelmente o maior já criado pela humanidade, com uma imensa quantidade de dispositivos conectados através de links de comunicação e comutadores, reunindo centenas de milhões de usuários ao redor mundo (KUROSE e ROSS, 2010).

De acordo com Tanenbaum (2003), a preocupação na criação de uma nova rede começa no final da década de 1950, no período da Guerra Fria, em que o Departamento de Defesa dos Estados Unidos queria uma rede de controle e comando capaz de sobreviver a uma guerra nuclear. Nessa época, a rede de comunicação dominante no mundo era a rede telefônica, entretanto, era considerada vulnerável. Anos mais tarde, foi criada a Agência de Projetos de Pesquisa Avançada (*Advanced Research Projects Agency - ARPA*).

Outra motivação para criação da Internet foi a preocupação com a falta de computadores de alta potência para os projetos de pesquisa da ARPA, como afirma Comer (2007), enfatizando que as primeiras redes de computadores foram projetadas para compartilhar poder computacional em grande escala, expandindo equipamentos de computação já existentes.

A ARPA iniciou um projeto de ligação em redes de dados, reunindo alguns dos melhores cientistas disponíveis para trabalharem na pesquisa em redes, o que resultou na criação da ARPANET, uma rede de dados que utiliza comutação de pacotes, sendo esta a base para as redes de dados posteriores (COMER, 2007).

As pesquisas em redes de dados continuaram, especialmente com a ligação inter-redes *(internet-working),* dando início à grande rede mundial de computadores ou Internet (COMER, 2007).

A Internet se baseia em protocolos organizados em camadas. Segundo Comer (2006), protocolos de rede proveem as regras sintáticas e semânticas para comunicação. Isso permite que diferentes componentes de hardware ou software possam trocar mensagens, pois os protocolos estabelecem padrões para o formato da mensagem, além de oferecer serviços como o tratamento de erros na transmissão e/ou recebimento dos dados.

A organização dos protocolos de rede em camadas permite dividir um problema grande e complexo em problemas menores e específicos, de forma que cada camada trate de uma questão específica e bem definida do sistema de uma rede de computadores. Desta forma, cada protocolo pertence a uma camada e executa uma determinada função dentro do seu escopo. Nesse modelo, uma camada oferece serviços à camada acima dela (KUROSE e ROSS, 2010).

As camadas podem oferecer serviços orientados a conexões e serviços sem conexão para as camadas superiores. No serviço orientado a conexões, inicialmente, é estabelecida uma conexão entre o transmissor e o receptor, e após a utilização desta conexão para a troca de mensagens, a mesma é liberada. Por outro lado, no serviço sem conexão ocorre apenas o envio da mensagem do transmissor para o receptor sem nenhuma negociação prévia ou confirmação de recebimento (TANENBAUM, 2003).

Como afirma Kurose e Ross (2010), os protocolos das várias camadas, quando tomados em conjunto, são denominados pilha de protocolos, sendo esta formada por cinco camadas: física, enlace, rede, transporte e aplicação. A camada física cuida da transmissão dos bits por um canal de comunicação. A camada de enlace divide os dados em quadros e os transmite de um elemento da rede a outro. A camada de rede cuida do endereçamento dos dispositivos na rede e também roteia os pacotes conhecidos como datagramas, utilizando um caminho que passa por diferentes roteadores entre a origem e o destino. A camada de transporte transporta as mensagens entre o transmissor e o receptor. Há dois protocolos nesta camada: o TCP *(Transmission Control Protocol)* que provê serviços confiáveis e orientados a conexão e o UDP *(User Datagram Protocol)* que oferece um serviço não orientado a conexão. E por fim, a camada de aplicação contém os protocolos utilizados pelo usuário, ou seja, as aplicações da rede.

2.2 Arquitetura Cliente-Servidor

A arquitetura ou paradigma cliente-servidor classifica as aplicações da rede em dois tipos básicos: clientes e servidores. Um cliente é programa da camada de aplicação que solicita e recebe um determinado serviço oferecido por um programa servidor (COMER, 2006).

O cliente e o servidor, geralmente, estão em computadores ou dispositivos diferentes e interagem por meio da troca de mensagens através da rede. A Figura 1 ilustra esta interação.

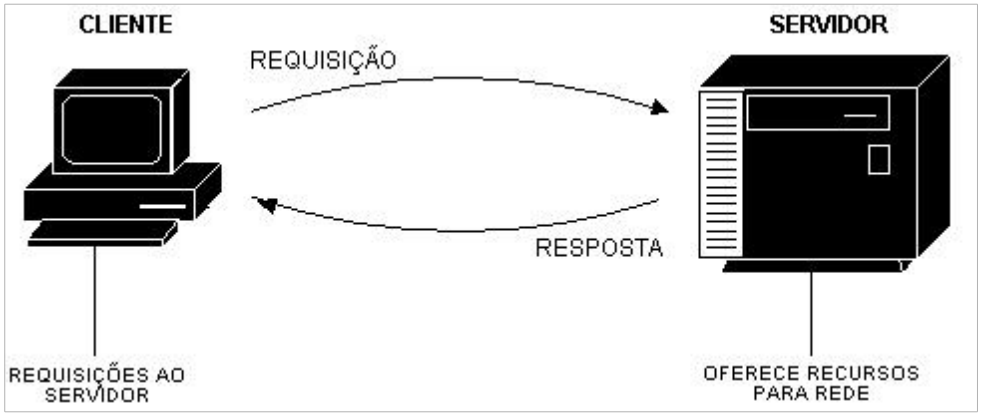

Figura 1: Interação entre dois computadores em uma rede. O primeiro possui um programa cliente que realiza uma requisição ao servidor, que por sua vez, envia a resposta.

Fonte: WIKILIVROS

O servidor é iniciado primeiro e aguarda as requisições do cliente. Quando uma requisição chega e é aceita, o servidor envia uma resposta para o requisitante e continua sua execução, aguardando novas requisições. E o cliente ao enviar uma requisição, fica na espera da resposta (COMER, 2007).

2.3 World Wide Web

Durante muito tempo, a Internet foi usada apenas para envio e recebimento de correio eletrônico e distribuição de dados científicos e acadêmicos por pesquisadores e estudantes universitários (KUROSE e ROSS, 2010). Portanto, seu uso era restrito às comunidades acadêmicas e de pesquisa.

Mas em 1991, surge uma nova aplicação para revolucionar e popularizar a Internet. Proposta pelo físico Tim Berners-Lee, entra em operação a *World Wide Web* (BERNERS-LEE, 1994).

Tanenbaum (2003) define a Web (também conhecida como WWW) como sendo uma estrutura arquitetônica que permite o acesso a documentos vinculados espalhados por milhões de máquinas na Internet. Portanto, a Web é uma aplicação

da Internet, que massificou o seu uso por todo o mundo, oferecendo uma imensa variedade de informações sobre topo tipo de assunto e apresentando uma nova experiência para o usuário, com ricos recursos gráficos e de interatividade.

O protocolo da Web é o Protocolo de Transferência de Hipertexto *(HyperText Transfer Protocol – HTTP)* (BERNERS-LEE, 1994). O HTTP utiliza o paradigma cliente-servidor, ou seja, é implementado em dois programas, permitindo que mensagens com estrutura padronizada sejam trocadas entre o programa cliente e o programa servidor, que se encontram em sistemas finais distintos (KUROSE e ROSS, 2010).

O programa servidor, conhecido como servidor Web, abriga os objetos dos documentos ou páginas Web, tais como arquivos HTML *(HyperText Markup Language)*, arquivos CSS *(Cascading Style Sheets)*, arquivos de script, arquivos de áudio ou vídeo, imagens, entre outros. Sua função é enviar um determinado documentado para um programa cliente, quando requisitado por este. Desta forma, os servidores Web implementam o lado servidor do HTTP. São exemplos de servidores Web o Apache, Nginx e Microsoft Internet Information Server (IIS) (KUROSE e ROSS, 2010).

O lado cliente do HTTP é implementado pelo navegador *(browser)*. Trata-se de um programa capaz de requisitar e exibir uma página Web, permitindo que o usuário clique com o mouse nos itens da página exibida (TANENBAUM, 2003). São exemplos de navegadores o Google Chrome, Mozilla Firefox, Internet Explorer, Safari e Opera.

Em geral, uma página Web é constituída de um arquivo base no formato HTML e diversos objetos referenciados (KUROSE e ROSS, 2010). Cada objeto da página, inclusive o arquivo base, é referenciado utilizando uma cadeia alfanumérica conhecida como *Uniform Resource Locator (URL)* (BERNERS-LEE; MASINTER; MCCAHILL, 1994).

A Figura 2 exibe a forma geral de uma URL HTTP.

<protocol>://<host>:<port>/<path>?<searchpart>

Figura 2: Forma geral de uma URL HTTP.

Fonte: Elaborada pelo autor

Segundo o formato mostrado na Figura 2, <*protocol*> é o nome do protocolo usado para acessar o documento. No contexto deste trabalho, o protocolo em questão é o HTTP. A segunda informação é o <*host*>, que especifica o nome do domínio do servidor (hospedeiro) que abriga o documento. O item <*port*> informa o número da porta do protocolo. Essa informação pode ser omitida na URL, caso seja usada a porta 80, que é a porta padrão do HTTP. O item seguinte é o <*path*>, que informa o caminho do objeto no servidor. Esse caminho é percorrido a partir do diretório raiz, que é informado na configuração do servidor. Além disso, o caminho pode ser omitido quando o objetivo requisitado encontra-se no diretório raiz e é o arquivo base da página Web, geralmente nomeado de *index.html*. Por fim, tem-se o <*searchpart*>, que é opcional. Este último item representa uma sequência de argumentos conhecidos como *query strings* (BERNERS-LEE; MASINTER; MCCAHILL, 1994).

Na Figura 3 tem-se o exemplo de uma URL do protocolo HTTP. Essa URL especifica o servidor de nome *www.uesb.br* e o caminho de um arquivo nomeado como *computacao/docentes.html*. Nota-se que a porta não foi informada, ou seja, utiliza a porta padrão, no caso, a porta 80. Além disso, não é utilizada nenhuma *query string*. Como mostrado no exemplo, uma URL contém as informações necessárias para que o navegador acesse um documento especificado e carregue uma página (COMER, 2007).

Figura 3: Exemplo de uma URL.
Fonte: Elaborada pelo autor

As URLs podem ser informadas pelo usuário a partir da barra de endereços do navegador ou através de *hiperlinks*, que são *links* ou vínculos para outras páginas em qualquer *host* (hospedeiro) que se encontra na Internet. Cada página pode conter vários hiperlinks. Esse modelo no qual uma página aponta para outra é conhecido como hipertexto (TANENBAUM, 2003).

2.4 Protocolo HTTP

O HTTP *(HyperText Transfer Protocol)* é o protocolo de transferência utilizado em toda a *World Wide Web,* sendo responsável pela especificação das mensagens que os navegadores podem enviar aos servidores e que respostas eles receberão, sendo, portanto, baseado em requisição e resposta (TANENBAUM, 2003).

O protocolo HTTP está em uso na Web desde de 1990. Sua primeira versão, definida como HTTP/0.9, era um protocolo simples para transferência de dados brutos na Internet (FIELDING *et al*., 1999). Os dados eram transferidos no formato de texto ASCII *(American Standard Code for Information Interchange)*.

Para expandir as possibilidades do protocolo, foi definida pela RFC 1945, a versão HTTP/1.0. Nesta versão, as mensagens são transmitidas no formato MIME *(Multipurpose Internet Mail Extension)* e também foram acrescentadas metainformações sobre as mensagens transferidas (BERNERS-LEE; FIELDING; FRYSTYK, 1996).

O HTTP 1.0 utilizava conexões não persistentes, ou seja, o cliente estabelecia uma conexão TCP com o servidor, enviava uma única solicitação e recebia uma única resposta, tendo a conexão encerrada após esse processo. Como as páginas Web eram constituídas apenas de textos HTML, esse método era bastante satisfatório. Entretanto, com o passar dos anos, as páginas Web se tornaram mais ricas em recursos, contendo grande quantidade de imagens e outros atrativos visuais. Desta forma, o estabelecimento de uma conexão para o transporte de cada objeto da página se tornou um modo de operação dispendioso (TANENBAUM, 2003).

Para resolver esse problema e ampliar as funcionalidades do protocolo, foi definido pela RFC 2616, o HTTP 1.1 (FIELDING *et al*., 1999). Como afirma Tanenbaum (2003), esta nova versão admite conexões persistentes, no qual uma mesma conexão pode conter solicitações adicionais e respostas adicionais, diminuindo o *overhead* do transporte via TCP.

O protocolo HTTP 1.1 utiliza o modelo *request-response* para realizar a troca de mensagens entre o navegador e o servidor. Inicialmente, o navegador estabelece uma conexão com o servidor. Após o estabelecimento da conexão, o navegador realiza uma ou mais requisições *(request)* ao servidor, que por sua vez, envia uma resposta *(response)* para cada requisição (FARIA e LOUREIRO, 1999).

As mensagens HTTP podem ser de dois tipos: requisição ou resposta. Ambas seguem padrões estruturais definidos pelo protocolo. De forma genérica, a mensagem é composta por uma linha inicial, zero ou mais linhas de cabeçalho, uma linha em branco que indica o fim do cabeçalho, e por fim o corpo da mensagem, que é opcional a depender da situação (FIELDING *et al*., 1999).

A Figura 4 descreve o formato geral da mensagem de requisição do protocolo HTTP 1.1.

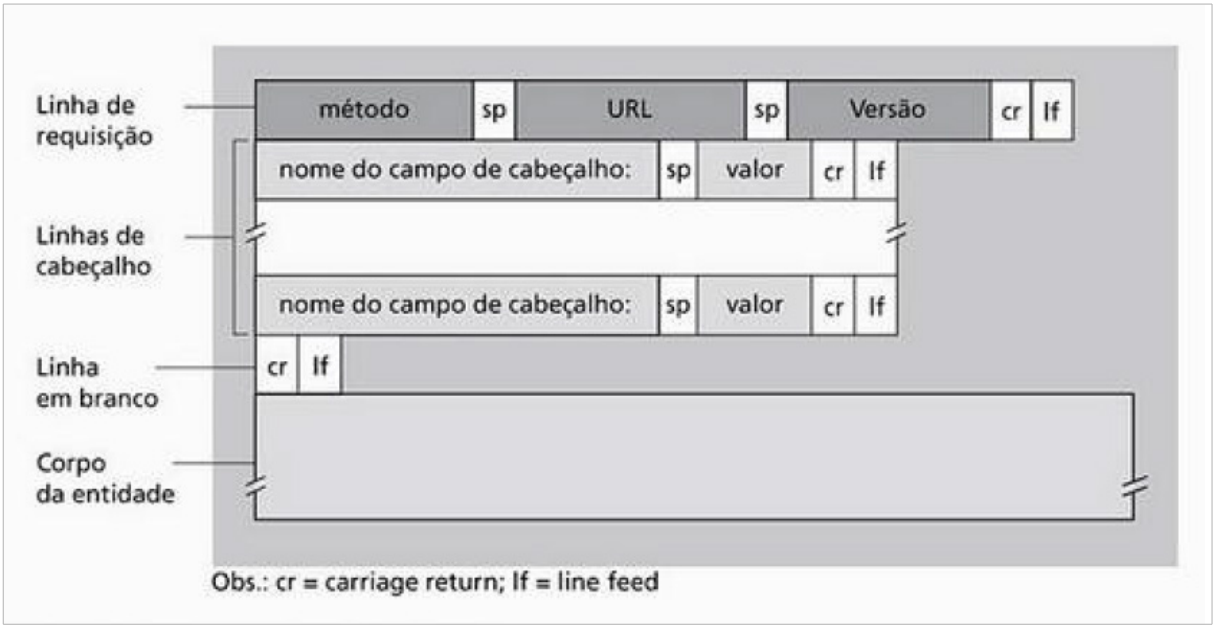

Figura 4: Formato geral de uma mensagem de requisição HTTP.

Fonte: KUROSE e ROSS, 2010.

A primeira linha da mensagem é a linha de requisição, que indica o método utilizado, o objeto requisitado e a versão do protocolo. O método informa qual a ação a ser realizada no recurso especificado. A maioria das mensagens de requisição utiliza o método GET, que é usado pelo navegador para solicitar um item específico do servidor (KUROSE e ROSS, 2010). Os outros métodos possíveis são HEAD, OPTIONS, POST, PUT, DELETE, CONNECT e TRACE (TANENBAUM, 2003).

O objeto requisitado é a parte da URL que especifica o caminho do recurso no servidor e a versão do protocolo é a versão em uso do HTTP que neste caso é HTTP/1.1. Na sequência estão as linhas de cabeçalho, que possuem informações adicionais sobre a mensagem. O Quadro 1 apresenta uma seleção dos cabeçalhos mais importantes. Os campos *cr* e *lf* são caracteres especiais que representam

c*arriadge return* e *line fead*, que fazem com que o cursor aponte para o início da próxima linha (KUROSE e ROSS, 2010).

Cabeçalho	Tipo	Conteúdo
User-Agent	Solicitação	Informações sobre o navegador e sua plataforma
Accept	Solicitação	O tipo de páginas que o cliente pode manipular
Accept-Charset	Solicitação	Os conjuntos de caracteres aceitáveis para o cliente
Accept-Encoding	Solicitação	As codificações de páginas que o cliente pode manipular
Accept-Language	Solicitação	Os idiomas com os quais o cliente pode lidar
Host	Solicitação	O nome DNS do servidor
Authorization	Solicitação	Uma lista das credenciais do cliente
Cookie	Solicitação	Envia um cookie definido anteriormente de volta ao servidor
Date	Ambos	Data e hora em que a mensagem foi enviada
Upgrade	Ambos	O protocolo para o qual o transmissor deseja alternar
Server	Resposta	Informações sobre o servidor
Content-Encoding	Resposta	Como o conteúdo está codificado (por exemplo, gzip)
Content-Language	Resposta	O idioma usado na página
Content-Length	Resposta	O comprimento da página em bytes
Content-Type	Resposta	O tipo MIME da página

Last-Modified	Resposta	Data e hora da última modificação na página
Location	Resposta	Um comando para o cliente enviar sua solicitação a outro lugar
Accept-Ranges	Resposta	O servidor aceitará solicitações de intervalos de bytes
Set-Cookie	Resposta	O servidor deseja que o cliente grave um cookie

Quadro 1: Alguns cabeçalhos de mensagens HTTP.

Fonte: TANENBAUM, 2003.

A Figura 5 apresenta um exemplo de uma típica mensagem de requisição HTTP, obtida através do *Wireshark* (WIRESHARK, 2015).

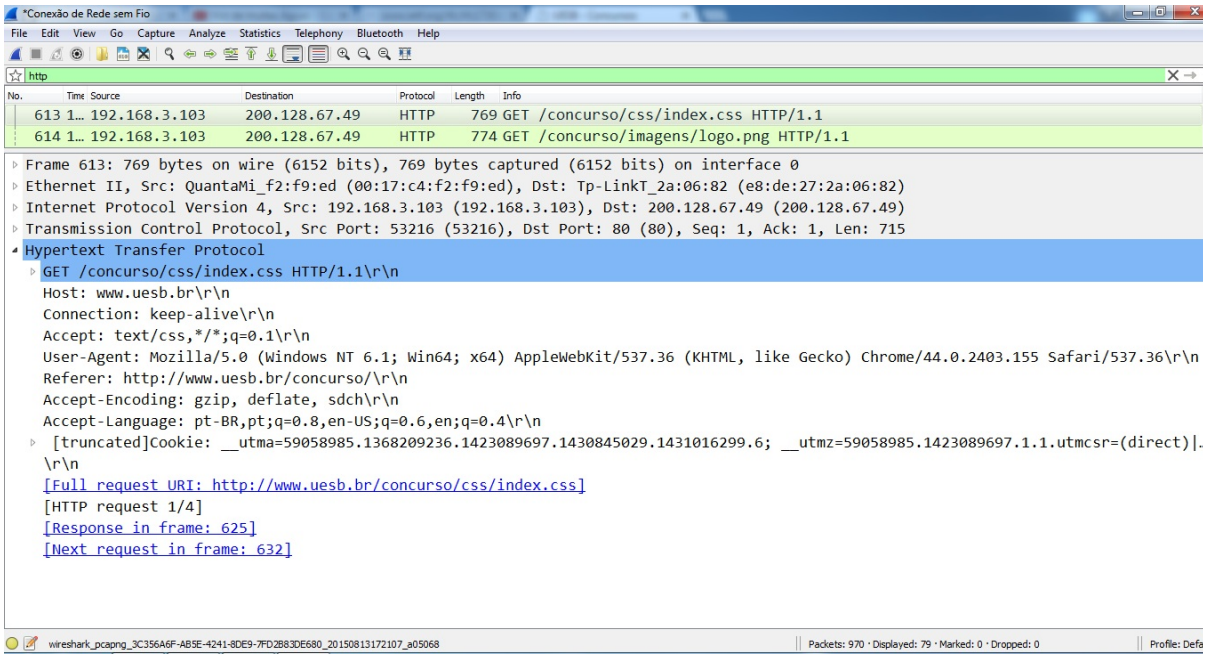

Figura 5: Exemplo de requisição HTTP.

Fonte: Elaborada pelo autor.

Neste exemplo, o navegador está requisitando o arquivo */concurso/css/index.css* ao servidor (*host*) endereçado pelo domínio *www.uesb.br*. As demais linhas de cabeçalho fornecem informações sobre a conexão, o navegador e os objetos aceitos. As informações do cabeçalho da mensagem de requisição são escritas em texto ASCII comum (KUROSE e ROSS, 2010).

A mensagem de resposta que o servidor envia ao navegador segue o mesmo padrão, diferindo na linha inicial e em alguns cabeçalhos que são específicos da resposta. Como pode ser verificado na Figura 6, a primeira linha da mensagem de resposta do HTTP 1.1, conhecida como linha de estado, possui três campos: o campo da versão do protocolo, um código de estado e uma mensagem de estado correspondente (KUROSE e ROSS, 2010).

O código de estado informa ao navegador se a solicitação foi atendida, ou seja, indica a resposta do servidor para a operação solicitada. Cada código de estado possui três dígitos. O primeiro dígito define a classe da resposta, podendo representar cinco categorias. Os dois últimos dígitos não possuem nenhuma regra de classificação (FIELDING *et al*., 1999).

O Quadro 2 mostra as cinco categorias e alguns exemplos de códigos de estado utilizados nas mensagens de resposta do HTTP.

Código	Significado	Exemplos
1xx	Informação	100 = o servidor concorda em atender a requisição do cliente
2xx	Sucesso	200 = requisição bem-sucedida 204 = sem conteúdo
3xx	Redirecionamento	301 = a página foi movida 304 = a página no cache ainda é válida
4xx	Erro do cliente	403 = página proibida 404 = página não encontrada
5xx	Erro do servidor	500 = erro interno no servidor 503 = tente de novo mais tarde

Quadro 2: Grupos de respostas de código de estado.

Fonte: TANENBAUM, 2003.

A Figura 6 descreve o formato geral da mensagem de resposta do protocolo HTTP 1.1.

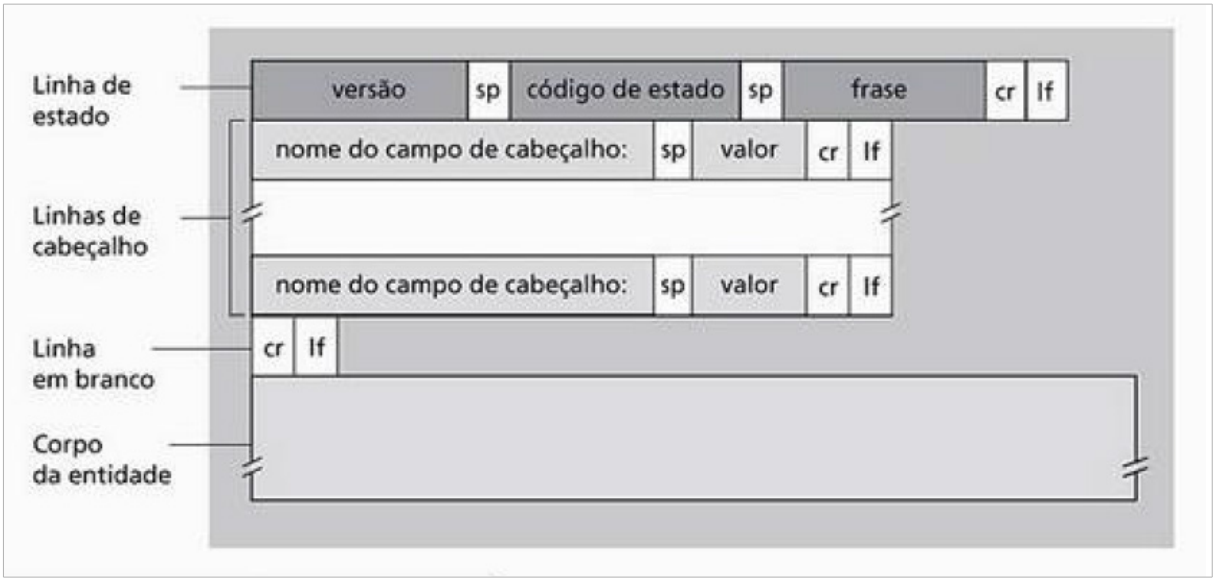

Figura 6: Formato geral de uma mensagem de resposta HTTP.

Fonte: KUROSE e ROSS, 2010.

O corpo da entidade ou mensagem contém o objeto que foi solicitado pelo navegador. A Figura 7 apresenta um exemplo de uma mensagem de resposta para a requisição feita na Figura 5, obtida através do *Wireshark*.

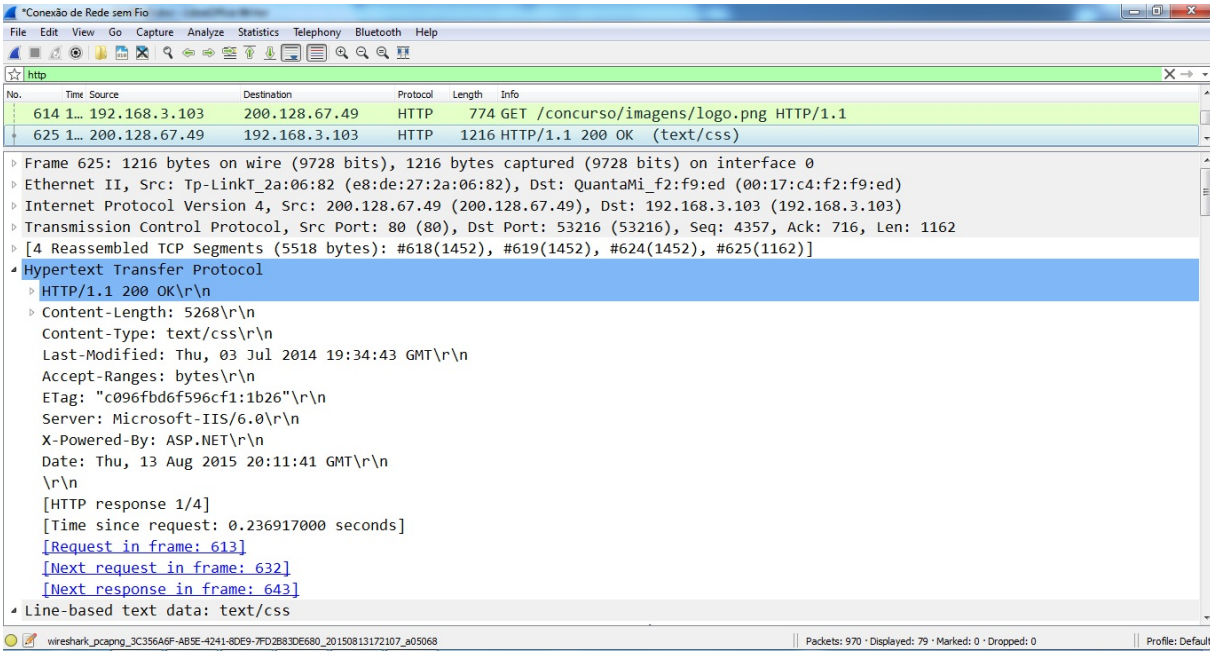

Figura 7: Exemplo de resposta HTTP.

Fonte: Elaborada pelo autor.

A linha de estado contém o código 200, informado que o arquivo solicitado foi encontrado no servidor e está sendo enviado na mesma mensagem. As linhas de cabeçalho apresentam informações sobre o servidor Web, data e hora em que a mensagem foi enviada, tipo MIME do arquivo, tamanho do arquivo em bytes, data e hora da última modificação no arquivo, entre outras informações (TANENBAUM, 2003).

O HTTP 1.1 também possui suporte para *caching* em navegadores. O armazenamento em *cache* é uma estratégia para melhorar o acesso a documentos, no qual o navegador armazena uma cópia de cada objeto requisitado e recebido em uma *cache* no disco local, permitindo acesso direto ao recurso quando ele for usado novamente, o que elimina a necessidade de uma nova requisição ou envio do mesmo objeto (COMER, 2007).

Outro forma de armazenamento em *cache* é a utilização de um servidor de *proxy*, que funciona como um intermediário entre o navegador e o servidor Web. O servidor de *proxy* é uma entidade da rede que tem seu próprio disco de armazenamento e mantém cópias dos objetos recentemente requisitados. O *proxy* é cliente e servidor ao mesmo tempo, pois atende as requisições do navegador, caso tenha uma cópia do objeto solicitado e requisita o objeto ao servidor, quando não possui uma cópia deste objeto armazenada localmente (KUROSE e ROSS, 2010).

O protocolo HTTP possui uma versão conhecida como HTTPS *(HyperText Transfer Protocol Secure)*, que fornece transferências seguras. A Figura 8 exibe uma mensagem HTTPS, obtida através do *Wireshark*.

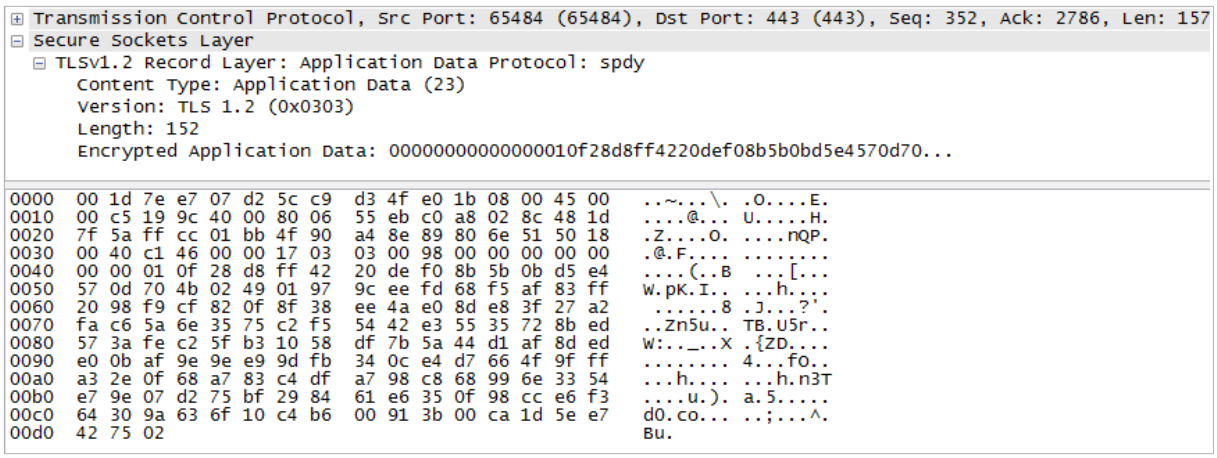

Figura 8: Mensagem HTTPS.

Fonte: Elaborada pela autor.

O HTTPS utiliza criptografia para garantir a confidencialidade dos dados transmitidos. Esse protocolo executa na porta 443 e sua implementação utiliza a conexão HTTP sobre uma camada segura SSL *(Secure Sockets Layer)* ou TLS *(Transport Layer Security)* (KHARE e LAWRENCE, 2000).

2.5 Trabalhos relacionados

Silva (2009) discute como se comportam, em termos de performance, duas das soluções mais usadas de aceleração HTTP cujas licenças são aprovadas pela Open Source Initiative: *Squid* e *Varnish.* O trabalho utiliza como métricas de avaliação o número de respostas HTTP por segundo e o tempo médio de conclusão de respostas. Como conclusão, comprova-se que ambas as soluções podem ser implementadas com sucesso como aceleradores HTTP. Mesmo implementando um *design* diferente, a performance de ambas as soluções é tecnicamente igual.

Padhye e Nielsen (2012) realizaram um estudo com testes controlados para comparar a performance do SPDY e HTTP em uma variedade de configurações. No trabalho é utilizado o tempo de carregamento de página como métrica de performance. Para alguns dos experimentos, foi utilizada uma página Web que consiste de várias imagens e arquivos CSS. Como resultado do trabalho, concluiu-se que o uso de otimizações pode tornar o desempenho do HTTP próximo ao do SPDY. Além disso, foi verificado que em páginas Web pequenas, o *overhead* do *handshake* SSL pode ter um impacto significativo no desempenho do SPDY.

3 PROTOCOLO HTTP 2.0

O HTTP surgiu como um protocolo aparentemente simples, fato que contribuiu para o sucesso da Web. Entretanto, o HTTP se mostrou bastante complexo à media em que evoluía, movido pelo aumento da complexidade do próprio ambiente no qual o protocolo é aplicado (MOGUL, 2002).

A versão 1.1 do HTTP foi lançada em 1999. Deste então, a forma como a Web é utilizada mudou bastante. As aplicações Web cresceram em funcionalidade, escopo e complexidade. Surgiram novos serviços como redes sociais, aplicações multimídia, modelos de negócios virtuais, sites mais elaborados, entre outros. Além disso, o próprio perfil de conectividade mudou com a utilização dos dispositivos móveis (GRIGORIK, 2013).

Em meio ao crescimento da Internet e mudanças significativas na forma de utilização da Web, o protocolo HTTP continuou o mesmo, se tornado um gargalo no desempenho das aplicações. Nesse cenário, usuários e desenvolvedores demandam uma performance próxima do tempo real para o protocolo HTTP 1.1, o que não é possível sem algumas modificações (GRIGORIK, 2013).

Com o intuito de prover uma solução para as limitações do HTTP 1.1, a Google propôs e desenvolveu o SPDY *(speedy)*, para ser o próximo protocolo de acesso a Web (MINEKI; UEMURA; HASEGAWA, 2013). O SPDY funciona como uma camada sobre o protocolo HTTP, pois as características básicas do protocolo, como cabeçalhos e métodos, não se alteraram. A diferença está na redução da latência do carregamento das páginas Web, usando um conjunto de novas técnicas para otimizar o transporte e a formatação dos dados.

O SPDY possui suporte em uma grande quantidade de navegadores, tais como Google Chrome, Mozilla Firefox, Internet Explorer e Opera. Além disso, é o protocolo padrão de grandes sites como Google, Twitter, Facebook, entre outros. É um protocolo que apresenta bons resultados, reduzindo o tempo de carregamento da página em cerca de 27 a 60%. Para obter essa melhoria, o SPDY introduz e combina uma série de técnicas, como multiplexação e priorização de requisições, compressão de cabeçalhos e envio de recursos do servidor para o navegador sem a necessidade de requisições adicionais (SALAM e*t al.*, 2014).

Baseando-se no SPDY, o *HTTP Working Group* do *Internet Engineering Task Force* (IETF) começou em 2012, a trabalhar na nova versão do HTTP, ou seja, as

especificações do SPDY foram adotadas como o ponto de partida para a discussão e desenvolvimento do protocolo HTTP 2.0, também escrito como HTTP/2. Ao longo dos anos seguintes, SPDY e HTTP 2.0 continuaram a evoluir em paralelo, com o SPDY sendo utilizado como um protocolo experimental, usado para testar as novas características e propostas para o padrão do HTTP/2 (GRIGORIK, 2015).

Após três anos de trabalho e vários esboços *(drafts)* intermediários, finalmente em 2015, foi revisado e aprovado o HTTP 2.0, definido pela RFC 7540. Trata-se de um protocolo que permite o uso mais eficiente dos recursos da rede, reduzindo a latência pela introdução de novas técnicas e características (BELSHE; PEON; THOMSON, 2015).

Como afirma Grigorik (2015), poucas semanas após a aprovação final do protocolo, muitos usuários já puderam aproveitar seus benefícios, pois diversos navegadores populares e muitos sites já dispunham de suporte total ao HTTP 2.0.

É importante ressaltar que a semântica do HTTP permanece inalterada. Os conceitos fundamentais como os métodos, códigos de estado, URLs e campos de cabeçalho continuam os mesmos. As mudanças no HTTP 2.0 ocorrem na maneira como os dados são formatados e transportados entre o navegador e o servidor. Portanto, as aplicações existentes continuarão a funcionar sem a necessidade de modificações (GRIGORIK, 2015).

3.1 Características do HTTP 2.0

O HTTP 2.0 traz uma série de mudanças positivas no modo de comunicação entre o navegador e o servidor, de forma a melhorar a performance da Web. Segundo Grigorik (2015), o HTTP 2.0 tornará as aplicações Web mais rápidas, mais simples e mais robustas, uma rara combinação.

Os objetivos principais do HTTP 2.0 são a redução da latência, minimização do *overhead* do protocolo, suporte para priorização de requisições e otimização do envio dos recursos necessários para o carregamento da página (GRIGORIK, 2015). Para alcançar esses objetivos, o HTTP 2.0 implementa um conjunto de técnicas e otimizações para oferecer um melhor desempenho na entrega dos recursos, bem como, garantir a segurança dos dados transmitidos.

3.1.1 Frame Binário

No HTTP 2.0, a comunicação é baseada na troca de *frames* entre o cliente e o servidor. Grigorik (2015), define o *frame* como a menor unidade de comunicação do protocolo. Nessa estrutura, os dados transmitidos são encapsulados e codificados no formato binário, possibilitando uma representação mais compacta da informação.

Como afirma Stenberg (2014), o HTTP 2.0 é binário para tornar o controle de *frames* mais fácil. Nas mensagens baseadas em texto ASCII é complicado implementar de forma correta a verificação de término de uma sequência de dados, bem como a utilização de espaços em branco opcionais, o que dificulta a identificação do início e fim de um *frame*. Já o formato binário permite implementações mais robustas, corretas e com ganho de performance (GRIGORIK, 2015).

A Figura 9 ilustra a camada de *frames* binários do HTTP 2.0 em comparação com uma típica mensagem do HTTP 1.1.

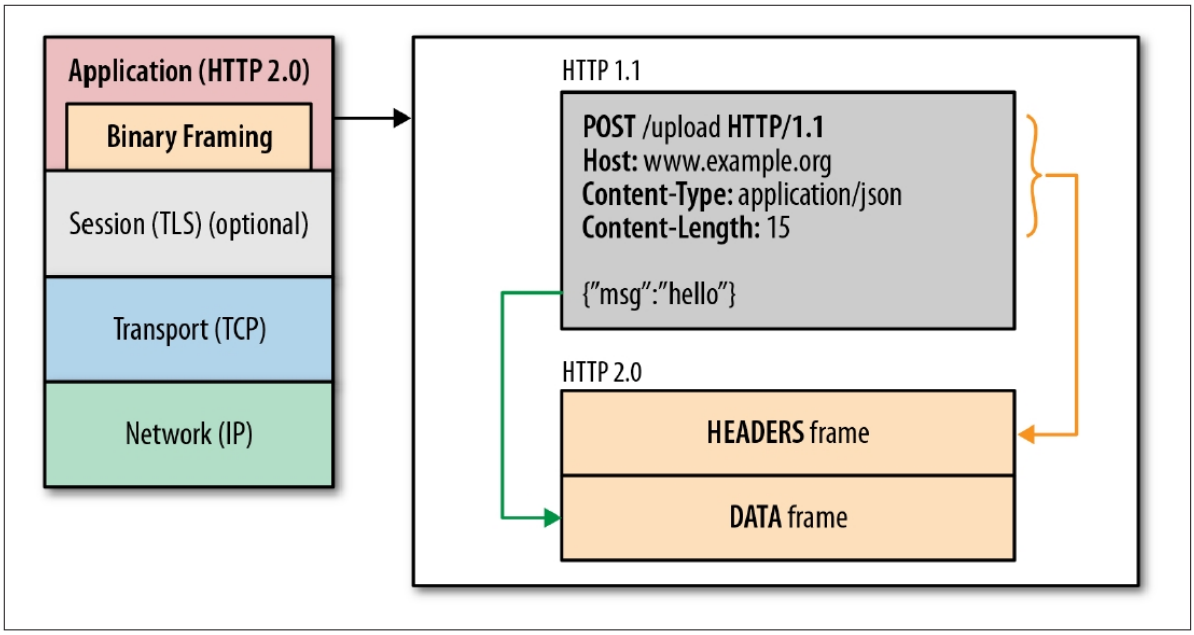

Figura 9: Camada de frames do HTTP 2.0.

Fonte: GRIGORIK, 2015.

No HTTP 1.1, uma única mensagem contém o cabeçalho e o conteúdo na mesma unidade de transmissão. Entretanto, no HTTP 2.0, as mensagens são divididas em diferentes tipos de *frames*, de acordo com a semântica dos dados. No exemplo da Figura 9, as linhas de cabeçalho foram encapsuladas em um *frame* específico para transmissão do cabeçalho da mensagem, enquanto o conteúdo foi encapsulado em outro *frame* específico para as informações do corpo da mensagem transmitida.

O HTTP 2.0 utiliza dez tipos de *frames*, que possuem funções diferentes na comunicação ente cliente e servidor. Cada *frame* do protocolo pode assumir os seguintes tipos:

- DATA: usado para transportar dados do corpo da mensagem HTTP;
- HEADERS: usado para transportar o cabeçalho de um determinado fluxo de mensagens;
- PRIORITY: usado para informar a prioridade de um fluxo de mensagens;
- RST_STREAM: usado para cancelar o envio de um fluxo de mensagens;
- SETTINGS: usado para informar os parâmetros de configuração para a conexão;
- PUSH_PROMISSE: usado para informar ao cliente, um determinado recurso que o servidor enviará, sendo que este recurso ainda não foi requisitado;
- PING: usado para medir o tempo de ida e volta;
- GOAWAY: usado para solicitar a parada na criação de fluxos de mensagens para a conexão em uso;
- WINDOW_UPDATE: usado para implementar controle de fluxo;
- CONTINUATION: usado para transporte da sequência de fragmentos de bloco de cabeçalho.

Todos os *frames* possuem um cabeçalho de nove bytes, seguido por uma carga útil *(payload)* de tamanho variável. As informações contidas no cabeçalho são: o tamanho do frame, seu tipo, um campo para *flags* e o identificador do fluxo de mensagens *(stream)* (BELSHE; PEON; THOMSON, 2015). A Figura 10 ilustra o formato típico do cabeçalho de um *frame*.

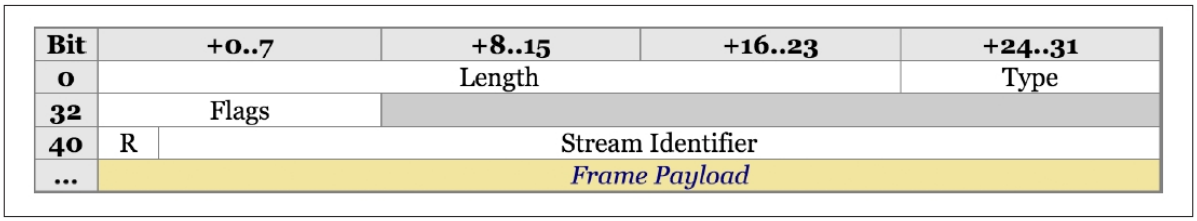

Figura 10: Cabeçalho de 9 bytes do frame.

Fonte: GRIGORIK, 2015.

Com base na estrutura ilustrada na Figura 10, verifica-se as seguintes conclusões a cerca do cabeçalho do *frame* HTTP 2.0:

- Os primeiros 24 bits são utilizados para informar o tamanho do *frame* em bytes, ou seja, permite representar a carga útil do *frame* em até 2^{24} bytes de dados;
- Os 8 bits seguintes informam o tipo do *frame*, o que determina o seu formato e a sua semântica;
- Na sequência são utilizados 8 bits para *flags* booleanas específicas do tipo do *frame*;
- O campo seguinte é reservado e possui 1 bit, sempre preenchido com valor zero;
- Os últimos 31 bits do cabeçalho informam o fluxo de mensagens *(stream)*, no qual o *frame* pertence.
- Os dados presentes na sequência após o término do cabeçalho compõem a carga útil do *frame*.

A Figura 11 apresenta um *frame* HTTP 2.0 do tipo DATA, ou seja, um *frame* de dados, obtido e decodificado através do *Wireshark*. Segundo Grigorik (2015), uma vez que uma nova *stream* é criada, e os cabeçalhos HTTP são enviados, *frames* do tipo DATA são usados para enviar a carga útil da aplicação, caso exista alguma. Analisando a Figura 11, nota-se os campos ilustrados na Figura 10, como tamanho do *frame* em bytes, tipo do *frame*, *flags* booleanas, campo reservado preenchido com zero, identificador da *stream*, e por fim, a carga útil transportada pelo *frame*.

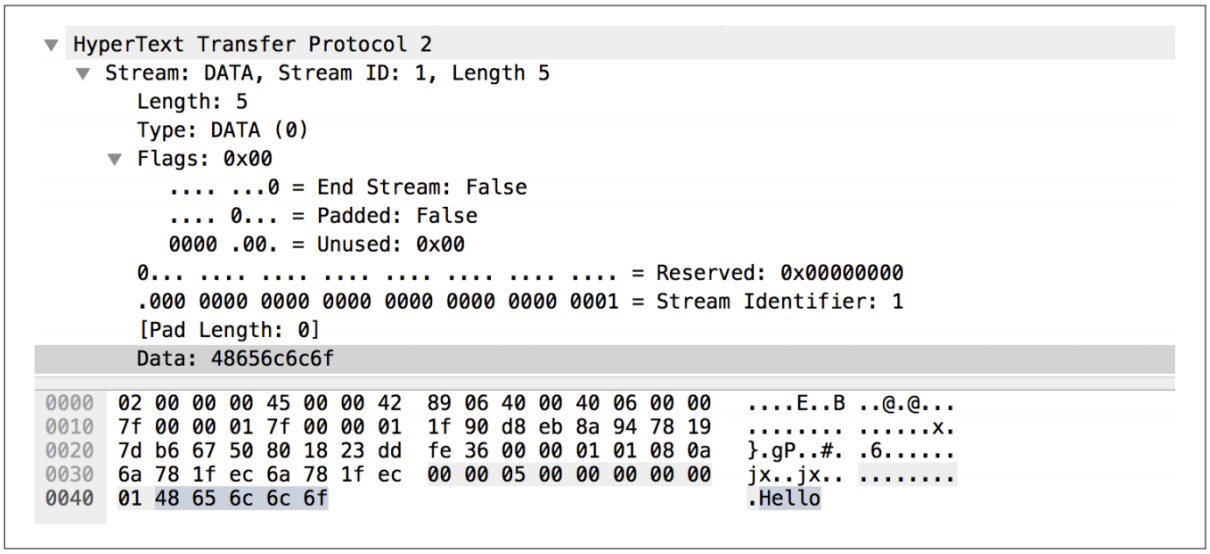

Figura 11: *Frame* de dados do HTTP 2.0.

Fonte: GRIGORIK, 2015.

De acordo com Grigorik (2015), a introdução do novo mecanismo de *frames* binários muda a forma como os dados são trocados entre o cliente e o servidor. Para entender o processo de comunicação no HTTP 2.0, é necessário o conhecimento das seguintes terminologias:

- *Stream:* trata-se de fluxo bidirecional de dados dentro de uma determinada conexão, que pode conter uma ou mais mensagens;
- *Mensagem:* é uma sequência de frames de compõem uma requisição ou resposta;
- *Frame:* como já falado anteriormente, é a menor unidade de comunicação do HTTP 2.0.

A Figura 12 ilustra a relação entre *frames*, mensagens e *streams*. Cada conexão TCP iniciada entre o cliente e o servidor, contém um conjunto de *streams*, sendo que cada *stream* possui um identificador único é composto por mensagens bidirecionais, ou seja, de requisição e resposta. Por sua vez, cada mensagem HTTP consiste de um ou mais *frames*.

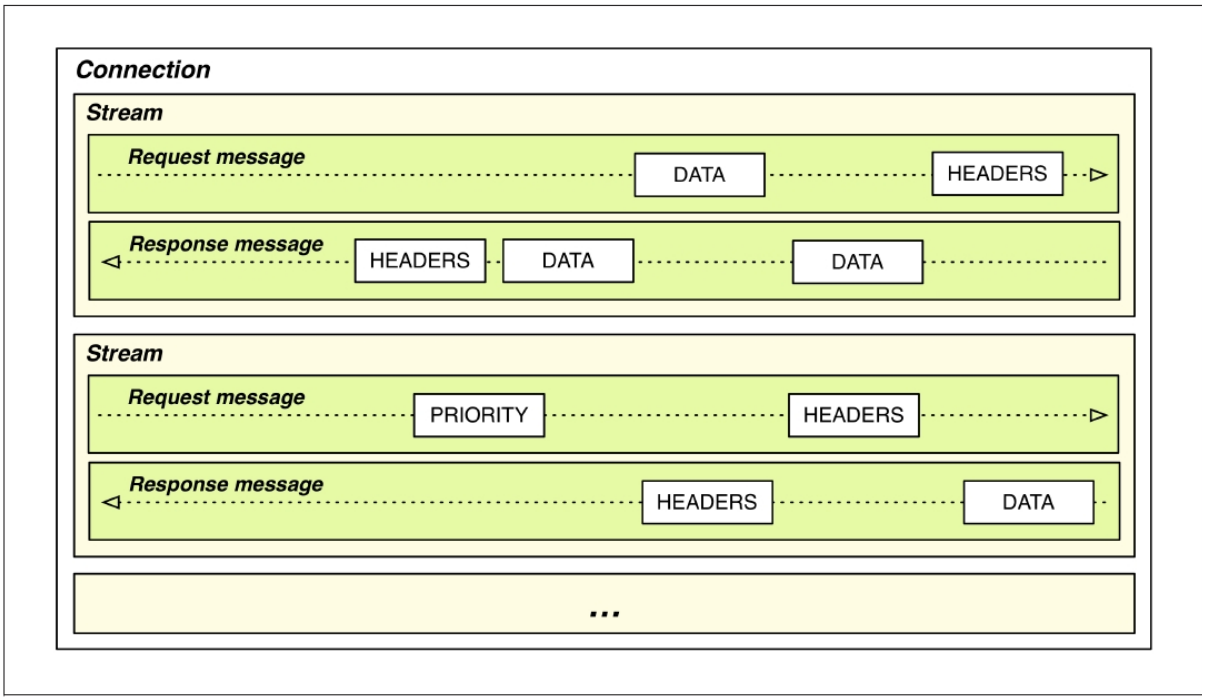

Figura 12: Relação entre frames, mensagens e streams.

Fonte: GRIGORIK, 2015.

A essência da arquitetura cliente-servidor da Web contínua a mesma. Entretanto, o HTTP 2.0 traz uma otimização na transmissão dos dados, ao dividir o conteúdo da mensagem em unidades menores e realizar o controle de *streams*. Essa nova estratégia de comunicação permite um melhor uso da conexão. Como afirma Grigorik (2015), este é o fundamento que permite todas as outras características e otimizações de performance providas pelo protocolo HTTP 2.0.

3.1.2 Multiplexação

Uma página Web é composta por diversos objetos como imagens, arquivos CSS, arquivos de *javascript*, arquivos de fontes, entre outros. Desta forma, o carregamento de uma típica página exige a realização de várias requisições ao servidor. Para que a página carregue mais rapidamente, é necessária a paralelização das requisições, ou seja, mais de um objeto pode ser requisitado e recebido por vez.

O HTTP 1.1 utiliza no cabeçalho a opção *keep-alive* para manter uma conexão TCP aberta e realizar diversas requisições na mesma conexão. Entretanto,

trata-se de um protocolo sequencial, no qual é preciso aguardar a resposta de uma requisição para iniciar uma nova. Para implementar o paralelismo de requisições, o HTTP 1.1 utiliza uma estratégia, que consiste no envio de múltiplos objetos sobre uma mesma conexão, permitindo que diferentes requisições sejam feitas em paralelo. Mas há um limite para a quantidade de requisições simultâneas que podem ser feitas, variando entre 4 a 8 requisições.

O HTTP 2.0 permite o paralelismo de requisições e respostas de forma automática em única conexão, através da multiplexação de *streams*. A Figura 13 ilustra esse mecanismo.

Figura 13: Multiplexação de requisições e respostas.

Fonte: Elaborada pelo autor.

A multiplexação funciona de forma assíncrona, ou seja, a realização de novas requisições independe do recebimento das respostas, que por sua vez, são enviadas à medida que forem processadas e concluídas. É necessária apenas uma única conexão para o carregamento paralelo e eficiente de todos os objetos da página, otimizando o uso da rede.

3.1.3 Controle de Fluxo

O uso de múltiplas *streams* paralelas em uma mesma conexão acarreta em disputas devido à concorrência, sendo necessário um esquema para controle de fluxo para impedir que as streams da conexão interfiram umas nas outras. O controle de fluxo do HTTP 2.0 é usado para *streams* individuais e para a conexão como um todo (BELSHE; PEON; THOMSON, 2015).

Além isso, o controle de fluxo previne o transmissor de sobrecarregar o receptor com dados que ele não queira ou não esteja habilitado a processar, pelo fato de estar ocupado com o processamento de outros recursos ou apenas esteja disposto a alocar uma quantidade fixa de recursos para uma *stream* específica (GRIGORIK, 2015).

O controle de fluxo é direcional e provido pelo receptor, que pode escolher a configuração de qualquer tamanho (em bytes) para a janela de recebimento de dados que deseja para cada *stream* e para a conexão inteira, sendo este limite rigorosamente repeitado pelo transmissor. À medida que os *frames* de dados vão sendo enviados, o valor da janela vai sendo reduzido. Para incrementar esse valor e permitir o envio de novos dados, o receptor pode informar o aumento da janela através do envio de um *frame* do tipo WINDOW_UPDATE (BELSHE; PEON; THOMSON, 2015).

O HTTP 2.0 não especifica nenhum algoritmo particular para a implementação do controle de fluxo. O protocolo apenas provê as ferramentas necessárias para a implementação de um algoritmo. Portanto, as diferentes abordagens ficam por conta dos clientes e servidores, que podem implementar estratégias para regular o uso e a alocação dos recursos, bem como otimizar a performance das aplicações (GRIGORIK, 2015).

3.1.4 Priorização de Requisições

Como já visto anteriormente, uma mensagem HTTP pode ser dividida em muitos *frames* individuais, que constituem diferentes *streams* ou fluxos de mensagens trocados em paralelo entre o cliente e o servidor. Neste contexto, a ordem na qual os *frames* são enviados dentro de uma conexão pode ser otimizada

para melhorar o carregamento e a performance da aplicação Web (GRIGORIK, 2013).

Segundo Stenberg (2014), as *streams* possuem uma prioridade, que é usada para informar ao servidor qual a *stream* considerada mais importante. Isso permite priorizar recursos que são fundamentais para o carregamento e utilização da página Web, ou seja, informa ao servidor quais requisições devem ser atendidas primeiro. A Figura 14 ilustra esse mecanismo de priorização de requisições.

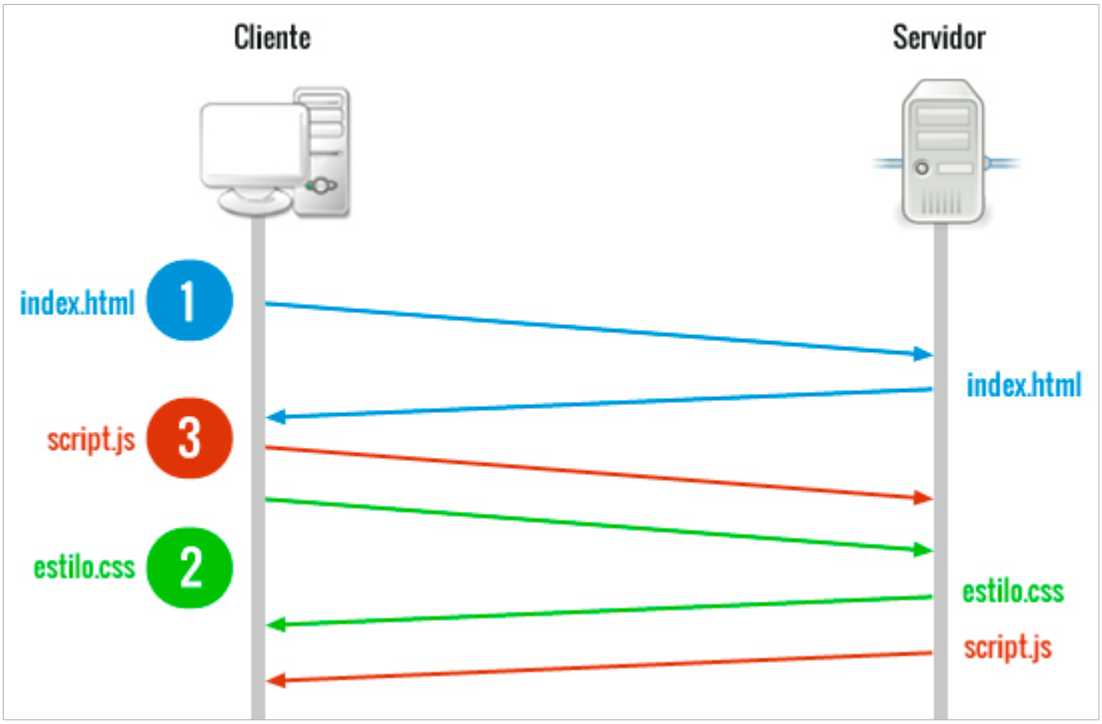

Figura 14: Exemplo de priorização de requisições.

Fonte: Elaborada pelo autor.

Como afirma Grigorik (2013), nem todos os recursos possuem a mesma prioridade na renderização de uma página no navegador. A Figura 14 exemplifica o carregamento de uma página constituída por três objetos. O arquivo HTML possui prioridade maior, pois contém a estrutura básica da página e referencia os demais recursos. O arquivo de *javascript* mesmo sendo requisitado primeiro, foi entregue depois do arquivo CSS, que possui maior prioridade em relação a este, pois é necessário para a formatação visual da página. Desta forma, objetos secundários como imagens ou outros arquivos de menor prioridade podem ser entregues depois dos objetos necessários para renderização inicial da aplicação.

3.1.5 Server Push

Segundo Wei e Swaminathan (2014), uma das novas características propostas no HTTP 2.0 é o mecanismo de *server push*, no qual o servidor envia alguns recursos para o cliente, sem que estes tenham sido requisitados ainda. É uma forma de o servidor enviar múltiplas respostas para uma única requisição do navegador.

Na ilustração da Figura 15, o navegador requisita o arquivo *index.html* ao servidor, que envia o arquivo solicitado e também envia os arquivos *estilo.css* e *script.js*. Neste modelo, o servidor infere requisições que seriam feitas logo em seguida, e pró-ativamente empurra os objetos para o cliente.

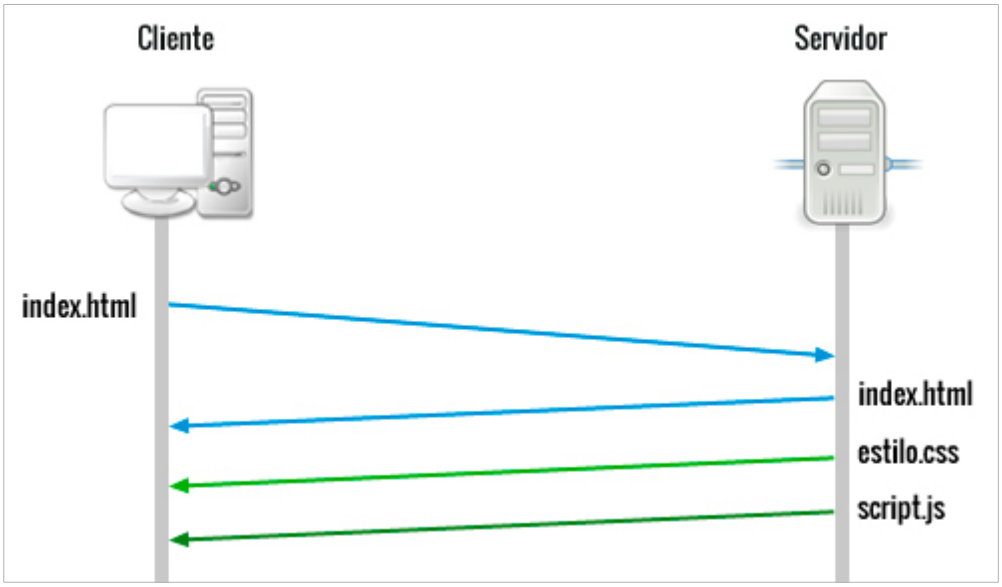

Figura 15: Exemplo do utilização do server push.

Fonte: Elaborada pelo autor.

Como afirma Stenberg (2014), isso ajuda o navegador a armazenar o recurso em *cache* para ser usado quando necessário. É também uma forma de priorização de elementos importantes para a renderização inicial da página, acabando com a necessidade do uso de *inline* de recursos, no qual um conteúdo CSS ou Javascript é embutido dentro do arquivo HTML, com o intuito de evitar uma nova requisição.

Esses recursos empurrados pelo servidor ao cliente podem ser reutilizados por diferentes páginas, multiplexados ao lado outros recursos e também podem

recusados pelo cliente quando este já possuir uma cópia em cache ou quando não forem utilizados na aplicação (GRIGORIK, 2015).

3.1.6 Compressão Automática de Dados

No HTTP 1.1, o uso do GZIP para a compressão dos dados enviados é opcional. O protocolo HTTP 2.0, porém, utiliza o GZIP como padrão e realiza a compressão dos dados de forma obrigatória. É uma boa opção para reduzir o tamanho dos *frames* de dados, diminuindo o volume de bytes que são trafegados na rede.

3.1.7 Compressão de Cabeçalho

Como afirma Grigorik (2015), cada transferência no HTTP carrega um conjunto de cabeçalhos que descrevem o recurso transferido e as suas propriedades. Desta forma, cada requisição e resposta do protocolo HTTP possui uma gama de metadados que muitas vezes se repetem em diferentes mensagens, provocando um *overhead* desnecessário.

Para reduzir o custo do envio de metadados e melhorar a performance do protocolo, o HTTP 2.0 comprime os cabeçalhos de requisição e resposta usando o algorítimo de compressão HPACK. Esse algoritmo utiliza código de Huffman para reduzir o tamanho dos dados e também adota uma estratégia para evitar o reenvio de valores de metadados já transmitidos anteriormente (GRIGORIK, 2015).

A compressão de cabeçalhos do HTTP 2.0 mantém uma lista indexada ou tabela dinâmica tanto no cliente, quanto no servidor. Essa tabela armazena os campos de cabeçalhos e seus devidos valores que são conhecidos e atualizados à medida que as requisições e respostas são transferidas em uma determinada conexão.

Analisando a Figura 16, é possível perceber de forma mais clara o funcionamento da tabela dinâmica. Nota-se que a primeira requisição carrega um cabeçalho com seis linhas de valores. Já na segunda requisição, a quantidade de linhas do cabeçalho enviado foi reduzida para apenas uma linha. Isso acontece, porque as informações do cabeçalho da primeira requisição já haviam sido armazenadas na tabela e seus valores não se alteraram na segunda requisição,

sendo desnecessário o seu reenvio. A única linha de cabeçalho que foi enviada na segunda requisição foi a linha que informa o recurso requisitado e trata-se de um valor diferente do anteriormente armazenado e conhecido.

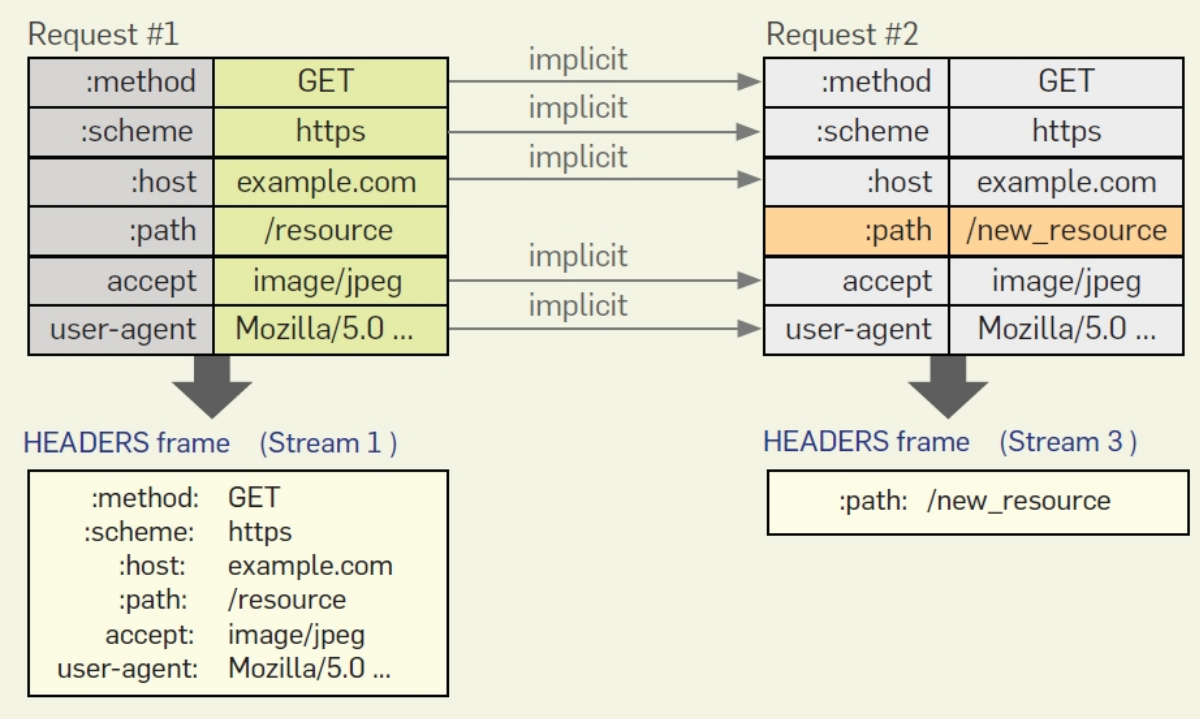

Figura 16: Exemplo do uso da compressão de cabeçalhos do HTTP 2.0.

Fonte: GRIGORIK, 2013.

Como resultado, o tamanho dos dados dos cabeçalhos enviados é reduzido pelo uso do código de Huffman que comprime as informações a serem transferidas, e pela utilização das tabelas presentes nos clientes e servidores, que armazenam os valores dos metadados que se repetem, evitando a necessidade de novas transmissões (GRIGORIK, 2015).

3.1.8 Criptografia e Segurança

O HTTP 2.0 não obriga o uso de TLS *(Transport Layer Security),* entretanto alguns dos principais navegadores como Google Chrome e Mozilla Firefox implementam o protocolo apenas sobre o TLS. Na prática, o uso de conexões seguras é a forma mais confiável para fazer a transição para o novo protocolo, uma vez que a Internet possui um grande número de intermediários como *proxies*, que

possuem aplicações e equipamentos que precisariam ser atualizados para entender a nova versão do HTTP (GRIGORIK, 2015).

Ao usar TLS, o cliente e o servidor podem negociar o uso do protocolo, possibilitando que navegadores antigos não sejam prejudicados e que os intermediários da rede também não sejam afetados pela mudança. Desta forma, o uso de conexões seguras do HTTP 2.0 permite também a compatibilidade do protocolo.

De qualquer forma, o uso do HTTPS para oferecer conexões seguras na implementação padrão do HTTP 2.0 é mais um ponto positivo do protocolo, pois contribui para a segurança e privacidade das aplicações Web.

O Quadro 3 apresenta um resumo com as principais diferenças entre os protocolos HTTP 2.0 e HTTP 1.1.

HTTP 1.1	**HTTP 2.0**
Protocolo textual.	Protocolo binário.
Protocolo sequencial. É necessário o uso de mais de uma conexão para simular o paralelismo de requisições.	Protocolo assíncrono. Utiliza multiplexação para realizar requisições paralelas em uma única conexão.
Não prioriza requisições.	Possui priorização de requisições.
Apenas o cliente pode iniciar uma requisição.	Possui o mecanismo de *server push*, que permite ao servidor inferir requisições futuras e realizar o envio antecipadamente.
Compressão de dados é opcional.	Compressão de dados é padrão e obrigatória.
Envia todos os dados de cabeçalho em cada mensagem.	Utiliza compressão de cabeçalhos para enviar apenas os dados de cabeçalho que sofreram alteração ou são desconhecidos na conexão.

Quadro 3: Principais diferenças entre HTTP 2.0 e HTTP 1.1.

Fonte: Elaborada pelo autor.

4 AVALIAÇÃO DE DESEMPENHO DO HTTP 2.0

Para avaliar o desempenho do protocolo HTTP 2.0 foram realizados alguns testes de carregamento de página, procurando obter dados comparativos em relação ao HTTP 1.1. Para tanto, foram utilizadas três métricas. A primeira métrica é o tempo de carregamento da página. Segundo Silva (2009), essa métrica sugere a capacidade de redução da latência experimentada pelo usuário. As métricas seguintes são o tamanho dos dados de requisição e tamanho dos dados de resposta, diretamente ligadas a quantidade de dados trafegados na rede. Como afirma Grigorick (2015), a redução do tamanho dos dados transmitidos acarreta em significativa melhoria na latência do carregamento da página.

4.1 Páginas utilizadas

Foram desenvolvidas dez páginas web com diferentes quantidades e tipos de objetos. O Quadro 4 apresenta a descrição das páginas, informando o tipo e a quantidade de cada arquivo de compõe a página, bem como, seus respectivos tamanhos em *kilobyte* (KB), sendo este tamanho um somatório para cada tipo de arquivo. As telas das páginas encontram-se no Apêndice A.

PÁGINA	OBJETOS
Página 01	1 HTML (1,07 KB)
Página 02	1 HTML (1,18 KB) 1 CSS (0,39 KB)
Página 03	1 HTML (1,24 KB) 1 CSS (0,39 KB) 1 JS (0,12 KB)
Página 04	1 HTML (1,23 KB) 1 CSS (0,39 KB) 1 JPEG (142 KB)
Página 05	1 HTML (1,29 KB) 1 CSS (0,39 KB)

	1 JS (277 KB) 1 JPEG (142 KB)
Página 06	1 HTML (1,60 KB) 2 CSS (1,87 KB) 2 JS (301 KB) 5 JPEG (367 KB)
Página 07	1 HTML (2,08 KB) 2 CSS (1,87 KB) 4 JS (309 KB) 10 JPEG (631 KB)
Página 08	1 HTML (2,82 KB) 3 CSS (3,16 KB) 6 JS (338 KB) 20 JPEG (2,75 MB)
Página 09	1 HTML (8,45 KB) 1 CSS (0,5 KB) 200 JPEG (7,02 MB)
Página 10	1 HTML (10,8 KB) 3 CSS (3,36 KB) 6 JS (338 KB) 220 JPEG (9,77 MB)

Quadro 4: Páginas desenvolvidas para os testes comparativos.

Fonte: Elaborada pelo autor.

As páginas possuem diferentes níveis de complexidade, partindo de uma página simples com apenas um arquivo HTML e aumentando a quantidade de recursos com a inclusão de imagens, arquivos CSS e arquivos de Javascript. Isso permite observar o desempenho do HTTP 2.0 à medida que página solicitada se torna mais pesada, exigindo mais requisições.

4.2 Ambiente de Testes

Os testes foram realizados utilizando duas máquinas conectadas via Internet. A primeira máquina é um servidor Web compartilhado, contratado a partir de um serviço de hospedagem, localizado nos Estados Unidos. A segunda máquina é um cliente localizado no Brasil, executando um navegador web.

4.2.1 Configuração do Cliente

O computador utilizado como cliente possui as seguintes configurações de hardware e software:

- Processador Intel Core i3 2.13 GHz
- Memória RAM de 3GB
- Disco Rígido de 250GB
- Sistema Operacional Windows 7 Home Premium 64 bits
- Navegador Web Mozilla Firefox 40.0.3

4.2.2 Configuração do Servidor

O computador utilizado como servidor é uma máquina compartilhada com as seguintes configurações de hardware e software para a conta de hospedagem contratada:

- Processador Intel Xeon 2.6 GHz
- Memória RAM de 1GB
- Disco Rígido de 6GB
- Sistema Operacional CloudLinux
- Servidor Web LiteSpeed 6.8

O servidor Web possui duas portas ativas para o acesso às páginas. Quando acessado na porta 80, o *LiteSpeed* executa o protocolo HTTP 1.1. Já na porta 443, o *LiteSpeed* oferece o serviço de transferência de página, executando o HTTP 2.0 sobre uma camada de segurança, ou seja, utiliza o protocolo sobre o HTTPS. O acesso ao servidor é feito através do domínio *sourcecoding.com.br*.

Para verificar o suporte do servidor ao HTTP 2.0 foi utilizada uma ferramenta online conhecida como *HTTP/2 Checker*. Essa ferramenta analisa o domínio e informada se o servidor oferece suporte para o HTTP 2.0 ou SPDY (HTTP/2... 2015).

A Figura 17 exibe o resultado da verificação do domínio *sourcecoding.com.br* no *HTTP/2 Checker*, evidenciando que o servidor oferece suporte ao HTTP/2 *draft* 14.

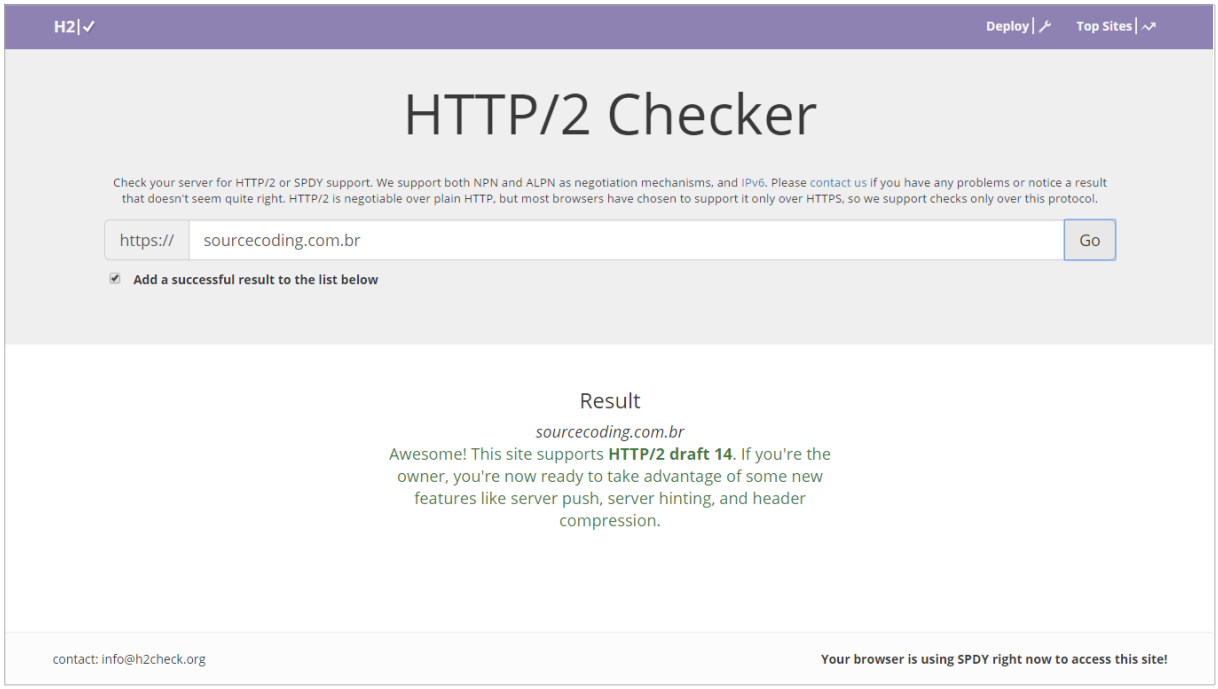

Figura 17: Verificação do servidor no HTTP/2 Checker.

Fonte: Http/2…, 2015.

4.3 Ferramenta Utilizada para Medição

Para fazer as medições foi utilizado o *HttpWatch*, que basicamente é um software para depuração de páginas Web. Esse software pode ser integrado a navegadores como Microsoft Internet Explorer e Mozilla Firefox, permitindo capturar e exibir as informações do HTTP ou HTTPS geradas pelo acesso a uma determinada página (HTTPWATCH, 2015).

Nos testes realizados para este trabalho, o *HttpWatch* foi utilizado em conjunto com o Mozilla Firefox para acessar e obter o tempo de carregamento das páginas, bem como, verificar o tamanho das mensagens de requisição e resposta. A Figura 18 exibe tela do *HttpWatch* quando uma página é acessada. Nota-se que a

mesma página foi acessada duas vezes, alternando entre o HTTP 2.0 e o HTTP 1.1, permitindo a comparação entre os dois protocolos.

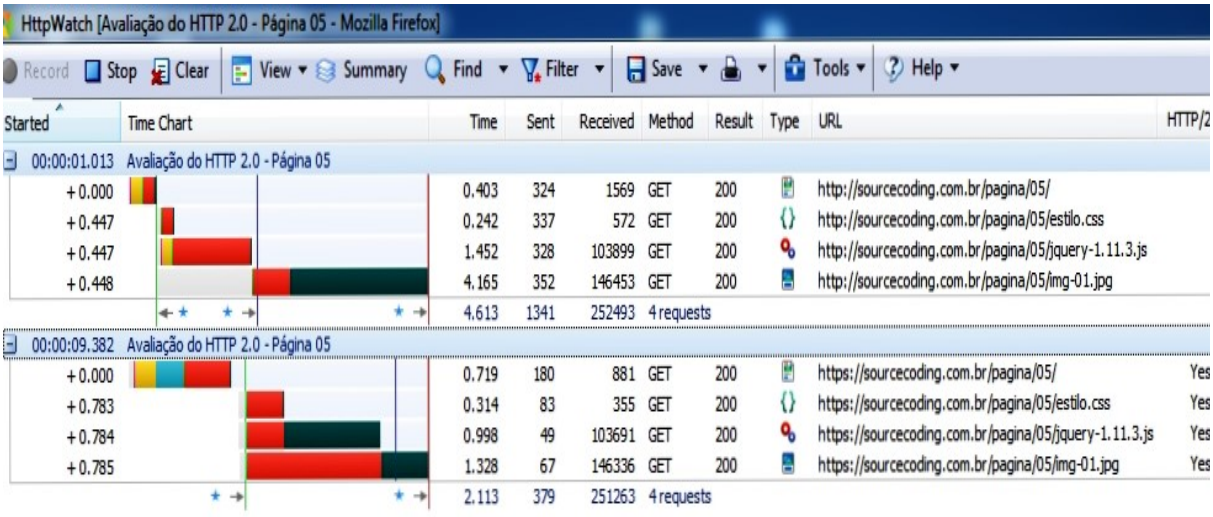

Figura 18: HttpWatch no Mozilla Firefox.

Fonte: Elaborada pelo autor.

A ferramenta mapeia cada objeto que compõe a página e gera também um gráfico, exibindo cada etapa do processo de carregamento dos arquivos e da página como um todo. Além disso, também informa se o protocolo HTTP 2.0 está sendo utilizado no acesso. O tempo de carregamento da página é medido em segundos e o tamanho dos dados enviados e recebidos é medido em *kilobytes* (KB).

4.4 Realização dos Testes

Os testes foram realizados em um ambiente real de uso, ou seja, as páginas hospedadas em um servidor web na Internet foram acessadas por meio de um navegador em um computador cliente. Isso permite observar o comportamento e o desempenho dos protocolos em um ambiente não controlado.

Cada página foi acessada três vezes para cada versão do protocolo HTTP. O horário de realização dos testes foi entre 1h e 2h. A utilização do HTTP 1.1 e HTTP 2.0 foi feita de maneira alternada. Os resultados foram tabelados, gerando uma média simples para representar o resultado final do conjunto de testes realizados.

Vale ressaltar que cada acesso foi feito após a limpeza completa da cache do navegador.

4.5 Resultados

Os testes realizados permitem comparar o desempenho do HTTP 2.0 e do HTTP 1.1, baseando-se em três métricas, como dito anteriormente. Foi avaliado o tempo de carregamento das páginas em cada protocolo, sendo que um menor tempo de carregamento representa um melhor desempenho. As outras métricas avaliadas estão diretamente ligadas a quantidade de dados trafegados na rede. Quanto menor o tamanho das mensagens de requisição e de resposta, menor a quantidade dados trafegados e desta forma, melhor é o desempenho do protocolo.

4.5.1 Tempo de Carregamento de Página

Segundo Bouch, Sasse e Demeer (2000), o tempo de carregamento de página é o intervalo de tempo desde o envio da requisição até a exibição da página correspondente no navegador. A Figura 19 ilustra os resultados obtidos nos testes

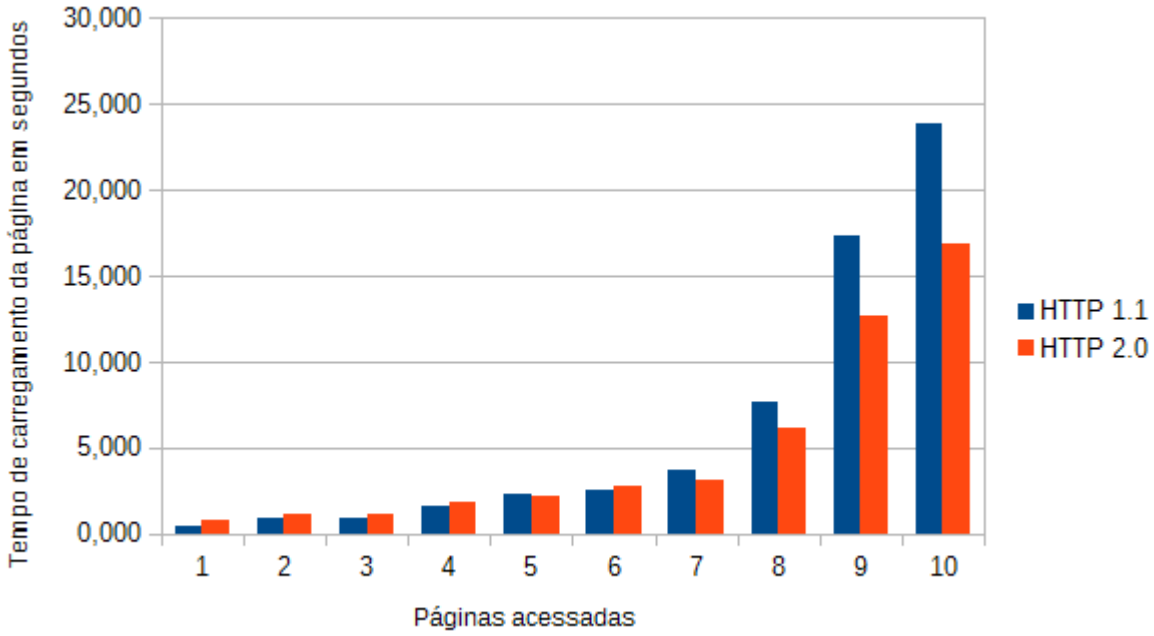

Figura 19: Tempo de Carregamento de Página.

Fonte: Elaborada pelo autor.

de tempo de carregamento de página em ambas as versões do protocolo HTTP avaliadas neste trabalho.

No gráfico é possível observar que em páginas mais simples, ou seja, páginas como menos recursos, o HTTP 1.1 apresentou um desempenho melhor que o HTTP 2.0. Isso acontece porque o HTTP 2.0 utiliza criptografia, e portanto, exige conexões TCP adicionais no processo de negociação TLS. Além disso, há o próprio *overhead* da encriptação e desencriptação dos dados.

Entretanto, a partir da página 7, o desempenho do HTTP 2.0 melhora consideravelmente, passando a apresentar tempos de carregamento de página muito menores do que o HTTP 1.1. O gráfico evidencia uma total superioridade do HTTP 2.0 em páginas carregadas com muitos objetos. De fato, essa nova versão do protocolo HTTP foi desenvolvida para melhorar o desempenho da Web, que se tornou mais complexa com o uso de novos serviços e aplicações mais ricas de recursos.

O *overhead* adicional da camada de segurança do HTTP 2.0 se torna praticamente desprezível no carregamento de páginas mais complexas, que exigem grande quantidade de requisições. Portanto, a superioridade de desempenho do HTTP 2.0 em relação ao HTTP 1.1 é proporcional à quantidade de objetos da página.

4.5.2 Tamanho das Mensagens de Requisição

A Figura 20 apresenta os resultados dos testes da análise do tamanho das mensagens de requisição. De acordo com o gráfico, o HTTP 2.0 supera e o HTTP 1.1 em todas as páginas testadas.

As mensagens de requisição são constituídas basicamente por informações de cabeçalho. Como visto no capítulo 4, o HTTP 2.0 utiliza o algorítimo de compressão HPACK para comprimir os cabeçalhos. Por isso o HTTP 2.0 possui cabeçalhos com tamanhos significativamente menores do que o HTTP 1.1, e portanto, menores mensagens de requisição.

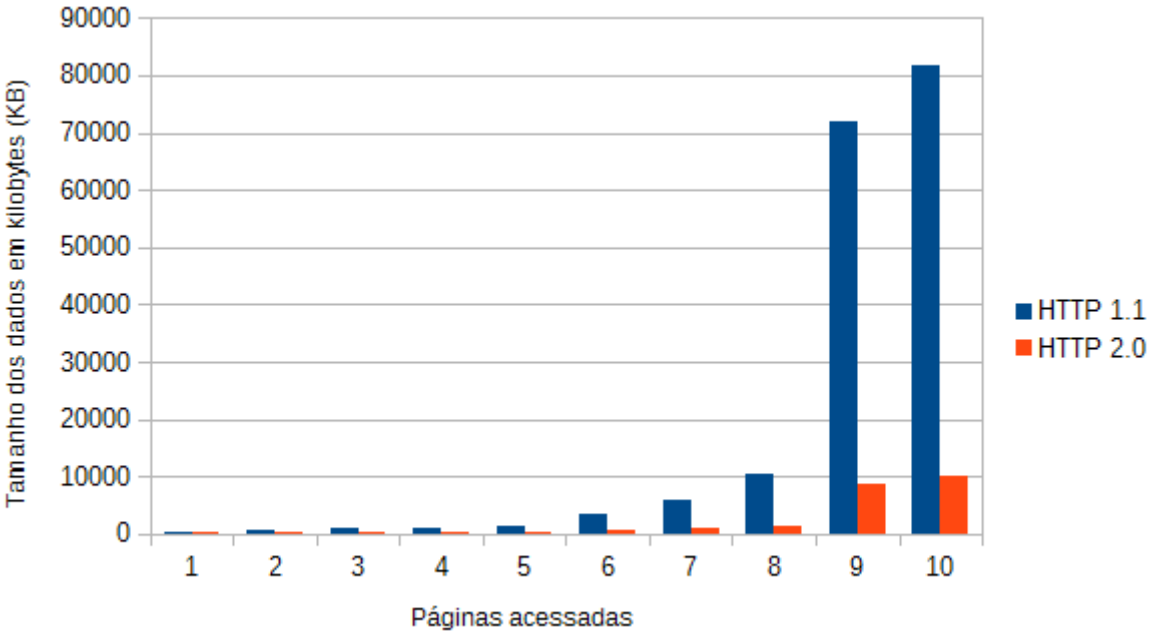

Figura 20: Tamanho das Mensagens de Requisição.

Fonte: Elaborada pelo autor.

4.5.3 Tamanho das Mensagens de Resposta

A Figura 21 exibe os resultados da verificação do tamanho das mensagens de resposta. O HTTP 2.0 apresenta uma breve vantagem em relação do HTTP 1.1. Neste caso, a maior parte dos dados transmitidos é o conteúdo dos objetos, apesar de também haver dados de cabeçalho.

Ambos os protocolos utilizam o GZIP para a compressão dos dados. Como visto no capítulo 5, a compressão de dados é padrão no HTTP 2.0 e opcional no HTTP 1.1, sendo utilizada nos testes realizados. Por isso, não há uma grande discrepância nos resultados dos dois protocolos, apesar de ainda haver uma sutil diferença por conta da compressão de cabeçalhos do HTTP 2.0, o que justifica seu melhor desempenho.

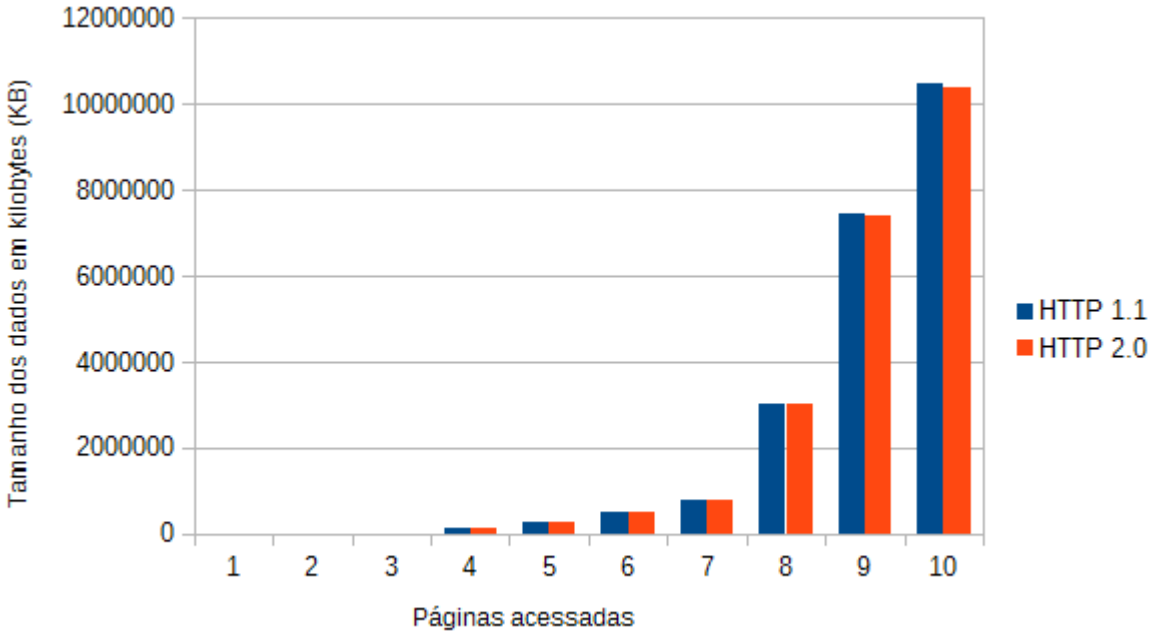

Figura 21: Tamanho das Mensagens de Resposta.

Fonte: Elaborada pelo autor.

5 CONCLUSÃO

De acordo com os resultados obtidos nos testes, o protocolo HTTP 2.0, de fato, apresenta um desempenho melhor do que o HTTP 1.1. Entretanto, vale salientar, que em algumas situações como no caso de páginas Web simples com poucos recursos, a situação se torna mais favorável para a versão 1.1 do protocolo HTTP. É inegável que o fato do HTTP 2.0 utilizar criptografia de forma intrínseca atribui um certo *overhead* que diminui o desempenho do protocolo, pois aumenta o tempo de carregamento das páginas.

Contudo, verifica-se que à medida que as páginas se tornam mais carregadas de recursos, o HTTP 2.0 passa a ter um desempenho muito melhor do que o HTTP 1.1. Esse cenário de páginas com muitos recursos é justamente, o cenário real de uso do protocolo HTTP, pois como foi dito, o ambiente Web ao longo dos anos foi se tornado cada vez mais complexo com o surgimento de novos serviços e aplicações que demandam grande quantidade e objetos visuais e arquivos de *scripts*.

Portanto, pode-se concluir que o HTTP 2.0 traz uma melhoria significativa na performance da Web. Além de otimizar o uso da rede e prover um serviço de entrega como menor latência, o protocolo ainda fornece a confidencialidade dos dados transmitidos.

Essa pesquisa observou o comportamento do HTTP 2.0 quando utilizado por um único usuário. Como trabalho futuro, seria interessante uma avaliação do desempenho do protocolo quando acessado por vários usuários, permitindo uma análise da escalabilidade do protocolo HTTP 2.0. Além disso, faz-se necessário realizar uma pesquisa acerca da performance do protocolo em dispositivos móveis, bem como, avaliar técnicas utilizadas em Javascript, CSS e HTML para melhorar o desempenho para o usuário final.

6 REFERÊNCIAS

BELSHE, M.; PEON, R.; THOMSON, M. **Hypertext Transfer Protocol Version 2 (HTTP/2)**. IETF, maio 2015. RFC 7540. Disponível em: <http://datatracker.ietf.org/doc/rfc7540/>. Acesso em: 03 jul. 2015.

BERNERS-LEE, T. **The Original HTTP as defined in 1991**. 1994. Disponível em: <http://www.w3.org/Protocols/HTTP/AsImplemented.html>. Acesso em: 10 jun. 2015.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. **Hypertext Transfer Protocol – HTTP/1.0**. IETF, maio 1996. RFC 1945. Disponível em: <https://tools.ietf.org/html/rfc1945>. Acesso em: 15 jun. 2015.

BERNERS-LEE, T.; MASINTER, L.; MCCAHILL, M. **Uniform Resource Locators (URL)**. IETF, dez. 1994. RFC 1738. Disponível em: <http://www.ietf.org/rfc/rfc1738.txt>. Acesso em: 11 jun. 2015.

BOUCH, A.; SASSE, M.A.; DEMEER, H. Of packets and people: a user-centered approach to quality of service. **2000 Eighth International Workshop On Quality Of Service. Iwqos 2000 (cat. No.00ex400),** [S.L.], p.189-197, jun. 2000. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/iwqos.2000.847955. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=847955>. Acesso em: 05 out. 2015.

COMER, Douglas E. **Interligação de redes com TCP/IP:** Volume 1 - Princípios, protocolos e arquitetura. 5. ed. Rio de Janeiro: Elsevier, 2006. Tradução de: Internetworking with TCP/IP, v.1, 5th ed.

COMER, Douglas E. **Redes de computadores e internet.** 4. ed. Porto Alegre: Bookman, 2007. Tradução de: Computer networks and internets with internet applications, 4th ed.

FARIA, Daniel B. de; LOUREIRO, Antonio Alfredo F. **Adaptando o Protocolo HTTP ao Ambiente de Computação Móvel**. XVII Simpósio Brasileiro de Redes de Computadores. 1999. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbrc/1999/017.pdf>. Acesso em: 22 jun. 2015.

FIELDING, R. et al. **Hypertext Transfer Protocol - HTTP/1.1**. IETF, jun. 1999. RFC 2616. Disponível em: <https://tools.ietf.org/html/rfc2616>. Acesso em: 15 jun. 2015.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 5ª edição. São Paulo: Atlas, 2010.

GRIGORIK, Ilya. **HTTP/2: A New Excerpt from High Performance Networking.** Sebastopol, CA: O'Reilly, 2015.

GRIGORIK, Ilya. Making the Web Faster with HTTP 2.0. **Queue – Development.** [S. L.], v. 11, n. 10, p.40-53, out. 2013. Disponível em: <http://dl.acm.org/citation.cfm?id=2542661.2555617&coll=DL&dl=ACM>. Acesso em: 30 jun. 2015.

HTTPWATCH. 2015. Disponível em: <http://www.httpwatch.com>. Acesso em: 30 set. 2015.

HTTP/2 Checker. 2015. Disponível em: <https://www.h2check.org>. Acesso em: 29 set. 2015.

KHARE, R.; LAWRENCE, S. **Upgrading to TLS Within HTTP/1.1**. IETF, mai. 2000. RFC 2817. Disponível em: <https://www.ietf.org/rfc/rfc2817.txt>. Acesso em: 27 jun. 2015.

KUROSE, James F.; ROSS, Keith W. **Redes de computadores e a internet:** Uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010. Tradução de: Computer networking a top-down approach featuing the Internet, 5th ed.

MINEKI, G.; UEMURA, S.; HASEGAWA, T. SPDY accelerator for improving Web access speed. **Advanced Communication Technology (ICACT), 2013 15th International Conference On.** Pyeongchang, p.540-544, jan. 2013. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6488246&queryText;=http/2>. Acesso em: 30 jun. 2015.

MOGUL, Jeffery C. Clarifying the fundamentals of HTTP. **Proceedings Of The Eleventh International Conference On World Wide Web.** Honolulu, Hawaii, USA, p.25-36, maio 2002. Disponível em: <http://dl.acm.org/citation.cfm?id=511446.511450&coll=DL&dl=ACM>. Acesso em: 30 jun. 2015.

PADHYE, Jitendra; NIELSEN, Henrik Frystyk. **A comparison of SPDY and HTTP performance**. July 26, 2012.

SALAM, A. Abdel et al. SPDY multiplexing approach on long-latency links. **2014 IEEE Wireless Communications And Networking Conference (WCNC).** Istanbul, p.3450-3455, abr. 2014. IEEE. DOI: 10.1109/wcnc.2014.6953140. Disponível em: <http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?tp=&arnumber=6953140&queryText;=http/2>. Acesso em: 30 jun. 2015.

SIEMINSKI, A. The impact of Proxy caches on Browser Latency. **International Journal of Computer Science & Applications**, II, n. II, p. 5 – 21, 2005.

SILVA, Lucas do R. B. Brasilino da. **Aceleração HTTP: Um comparativo de performance entre as soluções Squid e Varnish.** 2009. 65 f. Monografia (Especialização) - Curso de Administração em Redes Linux, Departamento de Ciência da Computação, Universidade Federal de Lavras, Lavras, 2009. Disponível em: <http://www.ginux.ufla.br/files/mono-LucasSilva.pdf>. Acesso em: 18 out. 2015.

STENBERG, Daniel. HTTP2 explained. **ACM Sigcomm Computer Communication Review.** [s.l.], v. 44, n. 3, p.120-128, 28 jul. 2014. Association for Computing Machinery (ACM). DOI: 10.1145/2656877.2656896. Disponível em: <http://dl.acm.org/citation.cfm?id=2656877.2656896&coll=DL&dl=ACM>. Acesso em: 30 jun. 2015.

TANENBAUM, Andrew S. **Redes de computadores.** 4. ed. Rio de Janeiro: Elsevier, 2003. Tradução de: Computer networks, 4th ed.

WEI, Sheng; SWAMINATHAN, Viswanathan. Cost effective video streaming using server push over HTTP 2.0. **2014 IEEE 16th International Workshop On Multimedia Signal Processing (MMSP).** Jakarta, p.1-5, set. 2014. IEEE. DOI: 10.1109/mmsp.2014.6958796. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6958796&queryText;=http+2>. Acesso em: 30 jun. 2015.

WIRESHARK. 2015. Disponível em: <https://www.wireshark.org/>. Acesso em: 17 out. 2015.

APÊNDICE A – Telas das Páginas Testadas

Página 1

Avaliação do HTTP 2.0

Página 01

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 2

AVALIAÇÃO DO HTTP 2.0

PÁGINA 02

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 3

AVALIAÇÃO DO HTTP 2.0

PÁGINA 03

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

A página 3 difere da página 2, por possuir arquivos de Javascript

Página 4

AVALIAÇÃO DO HTTP 2.0

PÁGINA 04

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 5

AVALIAÇÃO DO HTTP 2.0

PÁGINA 05

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

A página 5 difere da página 4, por possuir arquivos de Javascript

Página 6

AVALIAÇÃO DO HTTP 2.0

PÁGINA 06

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 7

AVALIAÇÃO DO HTTP 2.0

PÁGINA 07

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 8

AVALIAÇÃO DO HTTP 2.0

PÁGINA 08

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia

Página 9

AVALIAÇÃO DO HTTP 2.0

PÁGINA 09

Página 10

AVALIAÇÃO DO HTTP 2.0

PÁGINA 10

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na World Wide Web. Ele surgiu da necessidade de distribuir informações pela Internet e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

Fonte: Wikipedia