Análise de Performance do PUSH em Conexões HTTP/2 no Carregamento de Páginas Web

Igor Nogueira de Oliveira[1], **Wesley Davison**[1], **Djamel Sadok**[1], **Patricia Takako Endo**[1 2]

[1]Grupo de Pesquisa em Redes de Computadores e Telecomunicações (GPRT)
Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

[2]Grupo de Estudos Avançados em Tecnologia da Informação e Comunicação (GREAT)
Universidade de Pernambuco (UPE)
Caruaru – PE – Brasil

{igor.nogueira, davison, jamel}@gprt.ufpe.br, patricia.endo@upe.br

Abstract. *Recently, the HTTP protocol was updated and received several modifications, focused mainly on improvements in the network respurce usage. Among these improvements one can cite the addition of push, a feature that allows the server to respond to a request with more than one resource simultaneously. This work presents a performance analysis of push feature on the transport of web pages on HTTP/2 connections. Therefore, experiments were conducted on a prototype using Total Download Time (TDT) as metric, and Web page requests with different amounts and sizes of objects.*

Resumo. *Recentemente, o protocolo HTTP foi atualizado e recebeu diversas modificações, direcionadas principalmente a melhorias na utilização dos recursos da rede. Dentre estas melhorias, pode-se citar a adição do recurso push, que permite que o servidor responda a uma solicitação com mais de um recurso simultaneamente. Este trabalho apresenta uma análise de desempenho do recurso push no transporte de páginas web em conexões HTTP/2. Para tanto, foram realizados experimentos em um cenário real, utilizando TDT (Total Download Time) como métrica de análise, e requisições de páginas web com diferentes quantidades e tamanhos de objetos.*

1. Introdução

Desde sua padronização definida em 1997 pela Internet Engineering Task Force (IETF) na RFC 2068 [Fielding et al. 1997], o protocolo Hypertext Transfer Protocol (HTTP) tem sido utilizado amplamente pela comunidade da Internet. Além de manter sua funcionalidade original de transmissão de arquivos hipertexto, o HTTP também provê o transporte de mídias mais dinâmicas, hoje denominadas hipermídias, tais como áudio e vídeo, não apenas em tempo real, como também de forma cada vez mais interativa. Mesmo com a diversidade de funcionalidades e características dos aplicativos web, o HTTP ainda continua a ser a escolha padrão para transporte de dados.

O HTTP 1.1 não proporciona a melhor utilização da rede possível por utilizar codificação de controle não compactado e em texto plano, sendo este último um fator incremental na complexidade computacional para análise de identificadores de quebra

de linha [Nielsen et al. 1998]. Além disso, segundo os mesmos autores, as restrições tornaram sua documentação desnecessariamente extensa e complexa, dificultando que suas implementações contivessem todas as funcionalidades definidas pelo padrão. Estes e outro pontos já eram identificados como falhos e que demandariam grandes alterações no protocolo.

Recentemente, uma nova proposta do HTTP foi definida [Belshe et al. 2015]. As principais melhorias propostas no Hypertext Transfer Protocol 2.0 (HTTP/2) são direcionadas ao melhor aproveitamento da rede. Para isso foram propostos o uso de codificação binária e uma estratégia específica de compactação para os dados de controle [Peon and Ruellan 2015]. Adicionalmente, a nova versão define multiplexação de requisições e respostas dentro de uma mesma conexão. Com isso, espera-se a diminuição do número de conexões ativas e consequentemente menor carga em vários pontos da rede.

Dentre as novas novas características do HTTP/2, o *push* permite o envio de recursos pelo servidor sem que haja necessidade de uma requisição explícita do cliente, de forma assíncrona. Esta funcionalidade já era, de certa forma, possível na versão 1.1 através da utilização de *pipelining* e técnicas de *polling*. Porém, ela ainda dependia da requisição explícita do cliente antes de efetuar o envio do conteúdo do recurso solicitado. Devido a natureza do modelo solicitação/resposta do HTTP, durante a utilização de *pipelining*, é possível que uma solicitação seja bloqueada por outra em casos de perda de dados ou variações na latência da rede. Este fenômeno é conhecido como Head-Of-Line Blocking (HOL Blocking).

Alguns dos protocolos utilizados como base para a definição do HTTP/2, como o SPDY (SPDY) e o Quick UDP Internet Connections (QUIC), já utilizam o recurso *push* em suas especificações. Contudo, como o HTTP/2 ainda é bastante recente, lançado em maio de 2015, alguns dos recursos definidos nesta versão ainda não são amplamente utilizados por servidores web. Dessa forma, o impacto que estes novos recursos causam ainda não foram extensamente analisados na prática.

Este trabalho apresenta uma análise inicial sobre o recurso *push* no transporte de páginas web em conexões HTTP/2 através de experimentos. Um servidor que implementa o HTTP/2 foi configurado para responder a solicitações de páginas web de diferentes características como quantidade e tamanho de objetos. Com isso foi possível duplicar a carga de rede gerada no carregamento de páginas web de diferentes configurações. A partir deste ambiente foi utilizado um cliente web compatível para acessar e capturar dados de temporização do carregamento de páginas Web em diferentes condições de rede.

Este artigo está estruturado da seguinte forma: a Seção 2 descreve conceitos básicos sobre HTTP/2; a Seção 3 apresenta os trabalhos relacionados a análise de desempenho do push; a Seção 4 descreve os experimentos realizados neste trabalho; a Seção 5 apresenta e discute os resultados obtidos; e por fim, a Seção 6 apresenta as conlusões e trabalhos futuros.

2. HTTP/2

De forma simplificada, o funcionamento do HTTP, independentemente da versão, dá-se da seguinte maneira: um cliente HTTP inicia uma solicitação através de uma conexão TCP para uma porta específica em um servidor HTTP (por padrão, porta 80 para HTTP, e

443 para HTTPS [Internet Assigned Numbers Authority (IANA)]). O servidor, ao receber o pedido, envia de volta um código de *status*, como por exemplo ”200 OK”, e, opcionalmente, o conteúdo no restante do corpo da resposta. O corpo desta mensagem é normalmente o recurso solicitado, apesar de uma mensagem de erro ou outras informações também poderem ser enviadas no cabeçalho. Os recursos HTTP são identificados e localizados na rede através de uma Uniform Resource Locators (URL), utilizando os esquemas de Uniform Resource Identifier (URI) http ou https para indicar o uso de conexões inseguras ou seguras, respectivamente.

A Figura 1 descreve um cenário simplificado de operação de uma sessão HTTP.

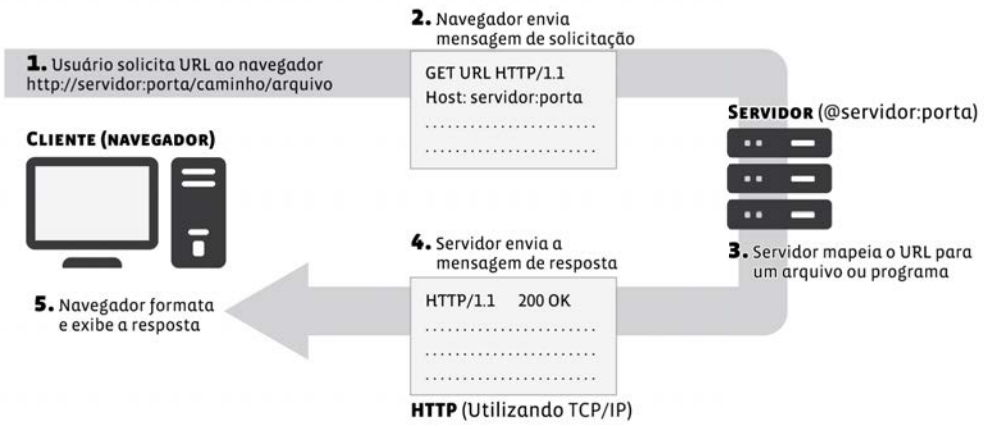

Figura 1. Visão geral de uma sessão HTTP. Adaptado de [Hock-Chuan 2009].

Métodos, como o GET exibido na Figura 1, são utilizados para indicar o tipo de ação em uma determinada solicitação. O HTTP 1.0 definiu 3 métodos, sendo eles: GET, POST e HEAD , utilizados para solicitar conteúdo, enviar conteúdo e solicitar apenas meta informações de um recurso, respectivamente. A versão 1.1 adicionou mais 5 métodos sendo eles: OPTIONS, PUT, DELETE, TRACE e CONNECT. A semântica utilizada para definir estes métodos permite que outros métodos sejam definidos por extensões.

Apesar de ser um protocolo considerado antigo, não houve grandes esforços no desenvolvimento de uma nova versão do HTTP. Apenas em 2007 foi formado pela IETF um novo grupo de trabalho, o HTTPBis. O grupo seria responsável por inicialmente revisar e atualizar todas as considerações definitivas do HTTP 1.1, que resultou nas RFCs 7230 à 7235.

De 2007 até 2012, foram analisados pelo HTTPBis alguns possíveis protocolos que poderiam ser utilizados como base para a versão 2.0. Em julho de 2012, um dos *feedbacks* ao *Call for Expressions of Interest in HTTP/2*[1], fornecido pela equipe de infraestrutura de rede do Facebook[2], relata quais características consideravam importantes na nova versão para atender as demandas em sua estrutura e quais protocolos já existentes estavam analisando.

[1]http://trac.tools.ietf.org/wg/httpbis/trac/wiki/Http2CfI
[2]http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0251.html

Dentre os possíveis protocolos a serem utilizados, como SPDY (formalmente FLIP, Google), HTTP Speed+Mobility (Microsoft) e WAKA (Roy Thomas Fielding, Adobe e Apache Software Foundation), apenas o **SPDY** atendia a maioria das características, além de já ser utilizado internamente na infraestrutura de rede do Facebook e em maior escala pela Google. Neste ponto da história, apenas o SPDY era suportado por dois do três navegadores Web mais populares, Google Chrome e Mozilla Firefox.

Em novembro de 2012, o primeiro *draft* do HTTP/2 foi publicado, sendo uma cópia direta da especificação do SPDY, até sua padronização oficial pela IETF em maio de 2015 através das Request For Commentss (RFCs) 7540 e 7541. Apesar da existência de diversas implementações e discussões pela comunidade ativa, o processo foi considerado muito rápido e possivelmente tendencioso até sua formalização [Kamp 2014].

As principais limitações do HTTP 1.1, como HOL Blocking, codificação em texto plano e falta de compressão de informações de controle(cabeçalhos), foram consideradas durante o processo de padronização da versão 2.0. Outros fatores técnicos de performance foram identificados para guiar o desenvolvimento do novo protocolo, especificamente, latência observada pelo usuário final, utilização de recursos de rede e recursos dos servidores. Além disso, o principal objetivo era implementar o uso de apenas uma conexão entre o navegador e o servidor Web[3].

Dentre os novos recursos do HTTP/2, este trabalho tem como foco o ***Server Push***. Através deste recurso é possível que o servidor responda a uma solicitação com mais de um recurso ao mesmo tempo. A implementação deste recurso pode ser descrita sucintamente através dos seguintes passos:

1. O servidor recebe uma solicitação que pode ter **recursos adicionais** como resposta;
2. No mesmo *stream* utilizado para responder à solicitação, o servidor envia um *frame* do tipo PUSH_PROMISE para cada recurso que deseja enviar via *push*. O *frame* PUSH_PROMISE contem um número de *stream* a ser utilizado pelo cliente para identificar futuros *frames* de dados do *push*;
3. O servidor considera como criado os *streams* informados no passo 2 e envia, assim que possível, *frames* de dados dos **recursos adicionais**;
4. O cliente, para cada PUSH_PROMISE recebido, deve optar por:
    - reservar um *stream* com o identificador sugerido pelo PUSH_PROMISE ou;
    - enviar um *frame* do tipo RST_STREAM informando que não irá aceitar o envio de *push*.

3. Trabalhos Relacionados

Estudos anteriores tais como [Padhye and Nielsen 2012] e [Podjarny 2012] analisam performance com uso do SPDY em relação ao HTTP e indicam resultados contraditórios, convergindo apenas no possível baixo desempenho do SPDY em redes móveis.

Em [Erman et al. 2013], os autores efetuam medições ao acessar os 20 sites mais populares do mundo, de acordo com o Alexa, utilizando SPDY e servidores *proxy* HTTP/1 em redes 3G. Eles atestam que o baixo desempenho do SPDY em redes móveis esta

[3]https://http2.github.io/

relacionado à forma como o Transmission Control Protocol (TCP) efetua o controle de congestionamento, em função de perdas e variância de latência da rede (*jitter*). Tendo em vista que o SPDY, assim como o HTTP/2, utiliza apenas uma conexão TCP, o impacto causado pelo controle de congestionamento é mais visível do que no HTTP, que utiliza várias conexões.

Ainda sobre o SPDY, [Wang et al. 2014] apresentou novas métricas e técnicas para caracterizar o desempenho deste protocolo. Através do isolamento do processo de carregamento de rede em relação aos demais processos envolvidos no carregamento de páginas Web, foi identificado que o SPDY possui melhor desempenho em relação ao HTTP. Porém, ao se considerar os demais fatores computacionais, o desempenho do SPDY é consideravelmente afetado. Apesar de propor formas de uso para o recurso *push*, a análise efetuada considera poucos casos, sendo focados números e tamanhos de objetos observados em um conjunto de 200 páginas.

Avançando para o HTTP, o trabalho apresentado em [Saxce et al. 2015] relata análises utilizando HTTP/2 em relação a diferentes fatores de rede como latência, perdas e largura de banda. Os resultados comprovam ganho de performance no uso do recurso *push* em um cenário simples de múltiplos objetos e tamanho fixo. Porém, os autores não estendem as análises em diferentes ambientes.

Por fim, [Han et al. 2015] descreve um *framework* que combina a utilização do *push* em conjunto com *Server Hint* para evitar uso duplicado na transmissão de objetos já armazenados em *cache* pelos clientes. Os comparativos de resultados do uso deste *framework* se baseiam apenas em configurações simples do uso do *push*.

Embora todos os trabalhos relacionado aqui descritos apresentem contribuições significativas, até onde sabemos, não foi encontrada uma análise específica sobre no impacto do uso do recurso *push* em relação aos fatores de tamanho e número de objetos em *streams* concorrentes. Os estudos efetuados consideram cópias de uma pequena fracão de páginas Web reais, que podem não representar de forma adequada características da população de Web sites existentes.

4. Experimentos

Os experimentos tem como objetivo comparar o comportamento das **variações no tempo** de carregamento de páginas Web através de conexões HTTP/2 com e sem a utilização do recurso *push*. Os resultados devem ser significativos o bastante para indicar quais fatores devem ser considerados ao se optar por este recurso.

Com o objetivo de recriar ao máximo a utilização de um servidor HTTP/2 em um ambiente real, a escolha da configuração do ambiente do experimento foi focada em soluções mais próximas possível do padrão dos protocolos envolvidos. Na Figura 2, pode-se visualizar as principais entidades envolvidas nos experimentos e como as mesmas estão interligadas.

O ambiente apresentado na 2 descreve um cenário de acesso a um servidor Web HTTP 1.1 utilizando um *proxy* reverso HTTP/2. Decidiu-se utilizar esta configuração de ambiente porque, durante o desenvolvimento deste artigo, o software HTTP/2 utilizado ainda não possuía suporte ao *backend* FastCGI[4] e, portanto, dificultaria a criação de

[4]http://www.fastcgi.com/drupal/node/6?q=node/15

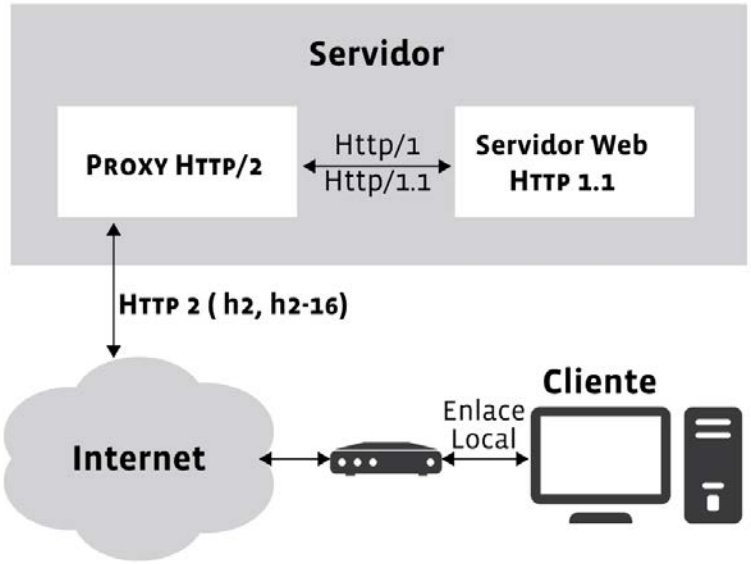

Figura 2. Entidades do ambiente

conteúdo de forma dinâmica.

Todas as requisições do cliente foram geradas por um navegador Web e possuem o mesmo servidor de destino para todos os objetos da página Web. A sequência de passos de uma requisição do experimento realizado são descritos no diagrama de sequência da Figura 3.

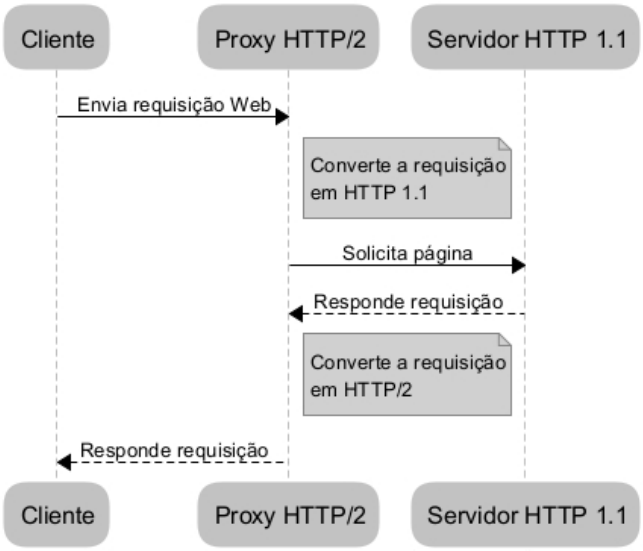

Figura 3. Diagrama de sequência das requisições do cliente

4.1. Infrastrutura de Software

O ***backend* de conteúdo do servidor Web HTTP 1.1** utiliza servidor Web Lighttpd versão 1.4.37, configurado na porta TCP 80 e com suporte a *backend* FastCGI habilitado.

Além disso, utiliza *scripts* PHP para gerar conteúdo dinâmico baseado em parâmetros da URL. Desta forma, é possível que um cliente solicite um recurso com tipo e tamanho de dados específico.

Para garantir que solicitações a um mesmo objeto não gerem armazenamento em *cache* pelo navegador Web, além de omitir cabeçalhos de controle de cache, o conteúdo é gerado de forma pseudo-aleatória, através da função openssl_random_pseudo_bytes. Esta função não requer acesso a disco para gerar conteúdo, o que garante que os dados gerados sejam rapidamente disponibilizados ao servidor Web.

O **servidor *proxy* HTTP/2** atua como *gateway* na conexão entre servidor Web e navegador Web do cliente. Este serviço implementa suporte ao protocolo HTTP/2 através do software Nghttp2, na versão 1.7.0, configurado na porta TCP 443, utilizando o servidor Web Lighttpd como *backend* HTTP. Bibliotecas desenvolvidas neste mesmo software são utilizados em outras ferramentas popularmente utilizadas como cURL[5] e Wireshark[6]. O servidor oferece suporte à extensão Application-Layer Protocol Negotiation (ALPN), provido pela camada de segurança Transport Layer Security (TLS). Através desta extensão, é possível informar quais os protocolos ofertados diretamente no final do *handshake* de segurança. Por padrão, o servidor suporta conexões em diferentes protocolos como HTTP/1.1, SPDY/3.1 e versões de desenvolvimento do HTTP/2 como h2-16 e h2-14 além da versão final h2. Para restringir o anuncio de protocolos não relacionados aos experimentos, o parâmetro de configuração "–npn-list=h2" foi utilizado ao iniciar o serviço.

O **cliente** utiliza o navegador Web Google Chrome na versão 48.0.2564.82 (64-bit) que foi escolhido por ofertar suporte nativo ao HTTP/2, além de facilidades para coleta de dados e manipulação interativa através de sua interface de depuração remota. A instalação deste não teve qualquer tipo de configuração adicional e não foram utilizados *plugins* ou modificações internas. A Tabela 1 resume as versões e configurações dos *softwares* do ambiente.

Entidade	Configuração do Software
Servidor Web	Lighttpd versão 1.4.37 TCP 80 *backend* FastCGI habilitado
Servidor Proxy	Nghttp2 versão 1.7.0 TCP 443 Servidor Web Lighttpd como *backend*
Cliente	Google Chrome versão 48.0.2564.82 (64-bit)

Tabela 1. Entidades de Software

As entidades em execução no servidor (servidor Web, servidor *proxy* e scripts PHP) foram compilados com base no sistema operacional Centos 7.0.1406 com o Kernel Linux 3.10.0123.el7.x86_64. A entidade no cliente, o Google Chrome, é executada no sistema operacional Fedora versão 22, utilizando o Kernel Linux 4.3.4200.fc22.x86_64.

[5]https://curl.haxx.se/

[6]https://www.wireshark.org/

4.2. Infraestrutura de Hardware

A estrutura de *hardware* do ambiente possui componentes capazes de executar os *softwares* de forma eficiente e sem gerar atrasos inesperados que poderiam influenciar na medição dos experimentos. A Tabela 2 resume as configurações de *hardware* utilizadas.

Componete	Processador	Memória	Rede
Servidor	Dual Xeon L5410	12 GigaBytes	Ethernet 100Mb/s
Cliente	Quad core Intel i7	16 Gigabytes	Ethernet 100Mb/s

Tabela 2. Componentes de Hardware

A interconexão entre os componentes é feita através da Internet, que não possui estrutura fixa de roteamento. Entretanto, durante a execução dos experimentos, foi possível observar um número constante de 13 saltos de rota e latência média entre 80 e 100 milissegundos.

4.3. Métrica, Fatores e Níveis

Este experimento definiu como métrica o **intervalo de tempo para conclusão do *download* de todos os objetos das páginas Web**, denominado **Total Download Time (TDT)**. Esta métrica considera apenas fatores relacionados a transações de rede, excluindo processos de renderização, validação e análise do conteúdo. A Tabela 3 apresenta todos os fatores e seus respectivos níveis utilizados nos experimentos para a análise da métrica definida.

Fator	Níveis
Tamanho	100KB a 2000KB com incremento de 100KB
Número de Objetos	10 a 200 com incremento de 10

Tabela 3. Fatores e Níveis

Alguns valores de níveis foram baseados em médias mineradas a partir do repositório Http Archive[7], no intervalo de janeiro de 2013 a dezembro de 2015. O sistema de coleta utilizado pelo Http Archive utiliza arquivos HAR para obter seus dados que não possuem poder descritivo para identificar como os objetos estão apresentados no documento HyperText Markup Language (HTML). Dessa forma, eles serão considerados nos experimentos como sendo o pior cenário possível, onde todos os objetos do tipo *JavaScript* são externos e causam bloqueio no processamento do Document Object Model (DOM).

Para definir o número de objetos do tipo *JavaScript* nos experimentos, utilizou-se como base os dados da Figura 4, que representa a frequência de números de objetos *JavaScript* e, a partir dela, fixou-se o limite de 200 objetos para o experimento.

Já para definir o tamanho dos objetos do tipo JavaScript, o nível do fator foi baseado na Figura 5, que representa a frequência de tamanhos de objetos também minerados do repositório Http Archive, coletados de janeiro de 2013 a dezembro de 2015.

[7]http://httparchive.org

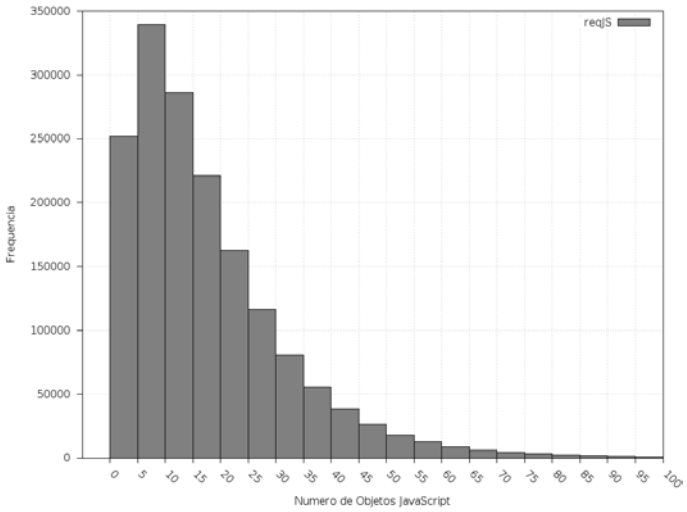

Figura 4. Frequência de número de objetos JavaScript

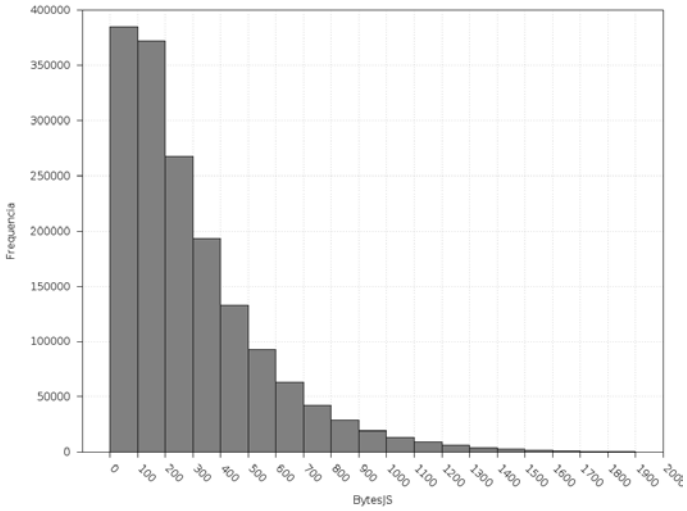

Figura 5. Frequência de tamanho de objetos JavaScript

Por não existir um modelo descritivo padrão para representar páginas Web, outros estudos como [Butkiewicz et al. 2011] e [Sivakumar et al. 2014] utilizam cópias de conteúdo de sites reais em um determinado período do tempo e os reutilizam durante os experimentos. Normalmente, os sites são selecionados com base em classificação de popularidade e tendem a representar o comportamento de uma população maior de sites.

5. Resultados

Os valores obtidos nos experimentos pela variação do tamanho de objetos são apresentados na Figura 6. Os grupos de experimentos apresentam incremento linear em função do tamanho dos objetos. O padrão visual dos intervalos de desvio padrão em relação as médias de cada grupo de experimentos indicou uma possível igualdade estatística. Para confirmação desta hipótese, o teste estatístico ANOVA foi executado[8]. Sendo o nível de significância considerado em 6%, 70% dos experimentos apresentam similaridade.

[8]Dados do teste não apresentados por falta de espaço

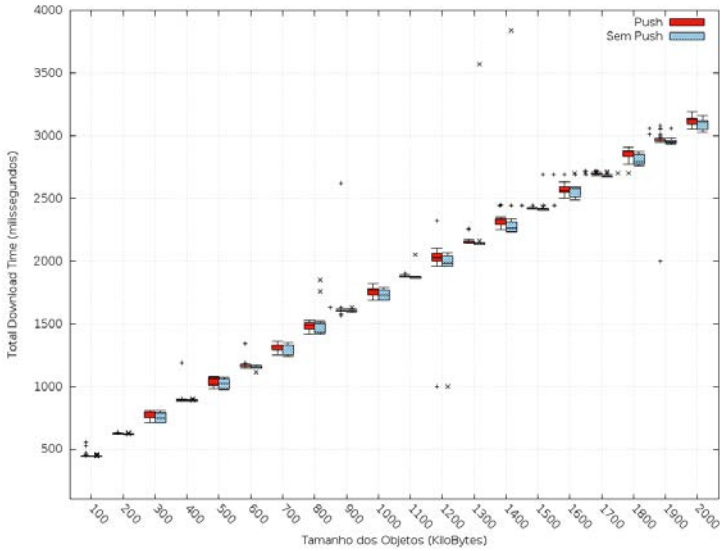

Figura 6. Comparativo do tamanho dos objetos para TDT

A Figura 7 apresenta os dados coletados nos experimentos com variação do fator número de objetos. Os valores apresentam grande dispersão, sendo este efeito mais frequente no uso do recurso *push* no intervalo de 17 a 110 objetos. Apesar desta dispersão, o teste estatístico ANOVA apresentou, para o nível de significância em 5%, que apenas 9 dos 20 experimentos são similares.

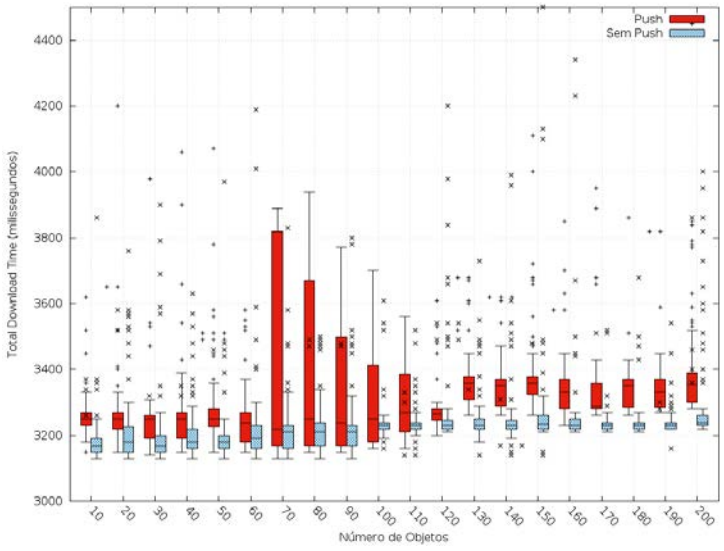

Figura 7. Comparativo do número de objetos em TDT

Os resultados obtidos pela métrica TDT indicam que **a quantidade de *streams* concorrentes, independentes do tipo, não afetam de forma crítica o tempo no processo de rede**. Apesar de não haver confirmação estatística de igualdade nos *streams* do tipo *push* e não *push*, ambos apresentam comportamento similar nos diferentes cenários. Esta caraterística é esperada devido o *design* do protocolo HTTP/2, uma vez que o mesmo utiliza multiplexação, e também indica que a compactação de cabeçalhos não

causa *overhead*.

Os níveis de dispersão apresentados com maior evidência durante o uso do recurso *push* nos experimentos com maiores números de objetos podem estar relacionados ao grande número de conexões efetuadas entre o servidor *proxy* e o servidor Web em um curto intervalo de tempo. Este evento ocorre pois para cada objeto a ser enviado através do recurso *push*, uma nova solicitação é gerada ao servidor Web, sendo necessário que o servidor *proxy* aguarde o recebimento do código de *status* antes de efetuar o envio dos respectivos *frames* do tipo PUSH_PROMISE.

6. Conclusões e Trabalhos Futuros

As limitações e resultados observados neste estudo revelam a possibilidade de continuação desta mesma análise considerando outros fatores e novas métricas, como o intervalo de tempo decorrido entre a solicitação das páginas experimentais e o evento de conclusão de carregamento do DOM, denominada PLT (*Page Load Time*).

Além disso, a variação de fatores da rede como latência e perdas devem ser analisadas, especialmente em ambientes reais, a exemplo de redes móveis. Também pode ser considerada a existência de múltiplos servidores como origem de conteúdos de uma mesma página, visto que a utilização de Content Delivery Networks (CDNs) deve continuar a ser fortemente utilizados.

Agradecimentos

Este trabalho foi apoiado pelo RLAM Innovation Center, Ericsson Telecomunicações S.A., Brasil

Referências

Belshe, M., Peon, R., and Thomson, M. (2015). Hypertext transfer protocol version 2 (http/2). RFC 7540, RFC Editor.

Butkiewicz, M., Madhyastha, H. V., and Sekar, V. (2011). Understanding website complexity: Measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 313–328, New York, NY, USA. ACM.

Erman, J., Gopalakrishnan, V., Jana, R., and Ramakrishnan, K. K. (2013). Towards a spdy'ier mobile web? In *ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, New York, NY, USA. ACM.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T. (1997). Hypertext transfer protocol – http/1.1. RFC 2068, RFC Editor.

Han, B., Hao, S., and Qian, F. (2015). Metapush: Cellular-friendly server push for http/2. In *5th Workshop on All Things Cellular: Operations, Applications and Challenges*, AllThingsCellular '15, New York, NY, USA. ACM.

Hock-Chuan, C. (2009). In introduction to http basics. `https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html`. [accessed: 2015-01-20].

Kamp, P.-H. (2014). Http/2.0 - the ietf is phoning it in. *Queue*, 13(1):10:10–10:12.

Nielsen, H. F., Spreitzer, M., Janssen, B., and Gettys, J. (1998). Http-ng overview - problem statement, requirements, and solution outline. Internet draft. `https://tools.ietf.org/html/draft-frystyk-httpng-overview-00`.

Padhye, J. and Nielsen, H. F. (2012). A comparison of spdy and http performance. *Microsoft Res*.

Peon, R. and Ruellan, H. (2015). Hpack: Header compression for http/2. RFC 7541, RFC Editor.

Podjarny, G. (2012). Not as spdy as you thought.

Saxce, H., Oprescu, I., and Chen, Y. (2015). Is http/2 really faster than http/1.1? In *IEEE Conference on Computer Communications Workshops*.

Sivakumar, A., Puzhavakath Narayanan, S., Gopalakrishnan, V., Lee, S., Rao, S., and Sen, S. (2014). Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 325–336, New York, NY, USA. ACM.

Wang, X. S., Balasubramanian, A., Krishnamurthy, A., and Wetherall, D. (2014). How speedy is spdy? In *USENIX Conference on Networked Systems Design and Implementation*, NSDI'14.