UFRJ – UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

POLI/UFRJ – ESCOLA POLITÉCNICA DA UFRJ

DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO

PROJETO DE PÓS-GRADUAÇÃO

ESTUDO COMPARATIVO DA PERFORMANCE DE MÉTODOS DE REQUISIÇÃO E RESPOSTA EM CONEXÕES WEB

Autor: Wallace de Souza Espíndola

Orientador: Heraldo Luís Silveira de Almeida

Coordenador da Disciplina: Sérgio Palma da Justa Medeiros

MBA ENGSOFT

Abril de 2018

Abril de 2018

ESTUDO COMPARATIVO DA PERFORMANCE DE MÉTODOS DE REQUISIÇÃO E RESPOSTA EM CONEXÕES WEB

Wallace de Souza Espíndola

Monografia submetida ao corpo docente da Escola Politécnica da Universidade Federal do Rio de Janeiro – POLI/UFRJ, como parte dos requisitos necessários à obtenção do diploma de MBA em Engenharia de Software.

Aprovada por:

_____

Orientador: Heraldo Luís Silveira de Almeida,
D.Sc., Prof., Poli/UFRJ.

_____

Coordenador da Disciplina: Sérgio Palma da Justa Medeiros,
D.Sc., Prof., Poli/UFRJ.

ESPINDOLA, WALLACE DE SOUZA

Estudo Comparativo Da Performance De Métodos De Requisição E Resposta Em Conexões Web [Rio de Janeiro] 2018.

IX, 67p. 29,7cm (POLI/UFRJ, Engenheiro, Engenharia de Software, 2018).

Projeto de Pós-Graduação – Universidade Federal do Rio de Janeiro – UFRJ.

1. Internet
2. Performance Web
3. Redes de Comunicação
4. Camada de Aplicação

I. POLI/UFRJ II. Título (série)

Dedicatória

Aos meus dois filhos queridos, Ana Luísa e Luís Eduardo, ofereço este trabalho como exemplo de dedicação, disciplina e persistência, evidenciando a importância do estudo, e como prova do que trabalho e esforço juntos podem realizar.

Agradecimentos

Primeiramente agradeço a Deus, por ter me guiado em cada passo da vida, tornando-me capaz de chegar até aqui.

Aos meus pais, que se sacrificaram e se esforçaram para que eu pudesse estudar e fosse vencedor, e por todo o apoio que me ofereceram durante toda a vida.

À minha esposa Janaína, pela dedicação, apoio, paciência e compreensão pelos momentos de ausência da convivência familiar.

Ao Prof. D.Sc. Heraldo Luís Silveira de Almeida, por ter sido sempre solícito quando necessário, e por ter me orientado neste trabalho de forma competente e profissional.

Ao CASNAV (Centro de Análises de Sistemas Navais) e à CPO (Comissão de Promoção de Oficiais) na Marinha do Brasil, por oferecer condições de trabalho e estrutura favoráveis, bem como aos meus colegas de projeto nestas organizações, pelo excelente trabalho de equipe, e por termos implementado juntos e com sucesso uma solução web inovadora e de alta performance baseada em WebSocket.

A todos aqueles que fazem parte da minha vida e que, de uma forma ou outra, me incentivaram e torceram pelo meu sucesso.

"Se a presença da eletricidade pode se tornar visível em qualquer parte de um circuito, não vejo razão por que a inteligência não possa ser transmitida instantaneamente pela eletricidade".

Samuel F. B. Morse, inventor do código Morse e do telégrafo.

"O que as redes de ferrovias, rodovias e canais foram em outra era, as redes de telecomunicações, informações e computação... são hoje em dia".

Bruno Kreisky, chanceler austríaco.

"...atualmente a questão é como acessar gigabits de informações através de linhas de velocidade ridícula".

J.C.R. Licklider, idealizador de importantes conceitos em computação e redes.

I. Resumo

O protocolo HTTP tem sido usado há mais de vinte anos na Internet, e apresenta certas limitações quando usado em conexões web de alta velocidade. Neste trabalho são abordados os efeitos de meios de requisição e resposta na transmissão de dados em aplicações web. Este trabalho apresenta um estudo comparativo entre os dois métodos de sondagem mais utilizados na web sobre HTTP, a sondagem padrão (Polling) e a sondagem longa (Long Polling), com o protocolo WebSocket uma alternativa com confiabilidade equivalente, mas com melhor performance e com menos sobrecarga de informação de cabeçalhos.

Inicialmente, as principais características do HTTP são apresentadas, assim como as dos métodos de sondagem e do WebSocket. As principais diferenças entre eles são mostradas. Então são realizados testes comparativos de desempenho dos métodos de requisição resposta selecionados. Os resultados são analisados detalhadamente, levando em consideração os tempos de processamento, atrasos na aquisição da resposta, e número de requisições necessárias para o mesmo resultado.

No geral, o método por conexão WebSocket apresentou o melhor desempenho nos cenários onde o tempo de processamento no servidor eram mais altos, enquanto que os métodos de sondagem curta e longa obtiveram resultados similares ao WebSocket em cenário de curto tempo de processamento com solução em chamada única.

Palavras Chave:

– HTTP

– Polling

– Long Polling

– WebSocket

– Web em Tempo Real

II. Abstract

The HTTP protocol has been used for more than twenty years on the Internet and it has certain limitations when used on high-speed web connections. In this work the effects of request and response means in the transmission of data in web applications are approached. This work presents a comparative study between the two most commonly used web-based probing methods over HTTP, Polling and Long Polling, with the WebSocket protocol, an alternative with equivalent reliability, but better performance and having less overhead of header information.

Initially, the main characteristics of HTTP are presented, as well as those of the Polling, Long Polling and WebSocket methods. The main differences between them are shown. Then, performance tests of the selected response request methods are performed. The results are analyzed in detail, considering the processing times, delays in the acquisition of the response, and the number of requisitions required for the same result.

In general, the WebSocket connection method presented a better performance in the scenarios where the processing time at the server was higher, while the short and Long Polling methods obtained results which are similar to the ones obtained in WebSocket approach in a short-time processing scenario with single call solution.

Keywords:

– HTTP

– Polling

– Long Polling

– WebSocket

– Real Time Web

III. Lista de Acrônimos

ACK	*Acknowledgement*
API	*Application Programming Interface*
CSV	*Comma Separated Values*
CERN	*Conseil Européen pour la Recherche Nucléaire*
DNS	*Domain Name Server*
FIN	*Finalize*
FTP	*File Transfer Protocol*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Secure Hypertext Transfer Protocol*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
NFS	*Network File System*
OSI	*Open Systems Interconnection*
POP3	*Post Office Protocol version 3*
PSH	*Push*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RFC	*Request For Comments*
RST	*Reset*
RTT	*Round Trip Time*
SMTP	*Simple Mail Transfer Protocol*
SSH	*Secure Shell*
SYN	*Synchronize*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
URG	*Urgent*

IV. Lista de Figuras

V. Lista de Tabelas

SUMÁRIO

1. Introdução

O mundo moderno não pode mais viver sem se comunicar, e as redes de comunicação desempenham um papel fundamental no cenário mundial atual. O enorme crescimento das demandas de informação levou a uma grande necessidade de elevação das taxas de transmissão de dados e da performance das aplicações web.

As redes gigabit chegam para resolver o problema atual de “gargalo” nos *back-bones* de comunicação e redes de alto tráfego, oferecendo uma banda passante extremamente larga, e, por conseguinte, a possibilidade de alcançar altíssimas taxas de dados.

Por outro lado, a existência destas redes também gera uma questão. Se estariam as tecnologias atualmente em uso evoluídas ao ponto de suportar essas altas taxas de dados. No presente momento a resposta certamente seria “não”.

1.1. Objetivo

Este trabalho tem como objetivo estudar e avaliar o desempenho dos métodos de requisição e resposta mais comumente utilizado atualmente em conexões HTTP, Polling e Long Polling, testando de forma experimental o seu desempenho, e comparando-os com uma alternativa mais moderna, o protocolo WebSocket [8]. Para tanto, serão levados em consideração o mecanismo de verificação por sondagem, comparando tempo de execução de requisições, tempo de execução de processo em back-end e o número de chamadas necessárias para finalizar o processo, sendo estes métodos de sondagem comparados com a conexão única que permanece em aberto do protocolo WebSocket, o qual é candidato a ser seu sucesso do HTTP na corrida pelo alcance de alta performance de conexões de dados na web.

O protocolo HTTP foi concebido na década de 90 e tem sido utilizado há quase 30 anos na Internet. Ele é a base da comunicação para a maioria das aplicações web utilizadas atualmente.

O mecanismo de sondagem trabalha para verificar assincronamente a execução de processos longos em indeterminados no servidor. No entanto, este

mesmo mecanismo gera alguns problemas de desempenho para web de alta velocidade devido à inundação da rede com pacotes em excesso, principalmente quando o produto banda x atraso é grande, pois a lentidão no envio e retransmissão dos pacotes provoca uma baixa utilização da banda passante disponível conexão [1].

Muitas tecnologias voltadas para a web de alta velocidades estão sendo propostas atualmente, visando o desenvolvimento de novas aplicações que envolvam a transmissão confiável de grande quantidade de dados, mas também uma melhor utilização das capacidades das redes atuais.

Neste trabalho são avaliados dois dos métodos mais tradicionalmente utilizados, Polling e Long Polling, além de um dos métodos mais modernos propostos, com o uso de WebSocket, comparando-os diretamente entre si.

1.2. Motivação

Com o advento das redes de alta velocidade, diversas aplicações tais como as aplicações multimídias distribuídas de tempo real e as aplicações de grade computacional de larga escala estão sendo viabilizadas [6]. De modo a utilizar essas e outras aplicações de grande consumo de banda em redes de longa distância como a Internet, grandes redes corporativas e outras redes com grande tráfego, a escolha de um mecanismo de requisição de dados mais adequado se torna de vital importância.

Uma internet de tempo real aproximado traz consigo a possibilidade do uso de novas aplicações, bem como melhoria na utilização das aplicações já existentes, mas também uma melhor distribuição dos recursos de rede. Por outro lado, tal recurso implica em grandes volumes de dados transitando numa rede, resultando em grande impacto no gerenciamento de tráfego, nos mecanismos de controle de fluxo e congestionamento dos protocolos, na oferta e gerenciamento de Qualidade de Serviço (QoS), e na escalabilidade de recursos [1].

A motivação deste trabalho se dá pelas grandes possibilidades do uso da web em tempo real e pelo fato de hoje em dia não haver ainda uma solução definitiva para o problema de excesso de trafego nas implementações de softwares que utilizam o

protocolo HTTP, o que mantém momentaneamente em aberto a solução a ser adotada, sendo esta certamente definida em breve.

Desta forma, um dos objetivos deste projeto é avaliar o desempenho de alguns dos protocolos citados, através experimentos, a fim de definir o método de requisição e resposta que mais se adequa às redes de alta velocidade. Assim sendo, a definição deste se torna de fundamental importância para viabilizar a utilização da banda usada na web de hoje de maneira mais eficiente em nas redes atuais, possibilitando a difusão em larga escala de serviços com alta exigência de banda e de alta performance de tempo de resposta, assim como o fim dos gargalos nos *back-bones* das grandes redes e da Internet.

1.3. Direcionamento do Trabalho

O presente trabalho está dividido em duas partes.

A primeira parte é baseada em uma pesquisa que abrange os fundamentos teóricos de dois protocolos, o TCP e HTTP, depois são descritos os métodos de sondagem sugeridos, Polling e Long Polling, e além destes, um estudo do protocolo WebSocket, para compor uma solução para o problema apresentado.

A segunda parte está voltada para testes reais de funcionamento, visando avaliar o desempenho dos três métodos citados, utilizando-se para tanto dois microcomputadores conectados que efetuarão conexões e transmissões de dados. Assim são constatadas as diferenças e similaridades entre tais métodos, visando à indicação de um que seja mais adequado a ser utilizado nas conexões em aplicações web de alta velocidade.

Este trabalho está organizado da seguinte forma: o Capítulo 2 as tecnologias envolvidas no presente estudo, mostrando suas principais características de funcionamento e seus mecanismos. A seguir, no Capítulo 3, algumas possibilidades da web em tempo real são apresentadas, com descrição de suas ideias gerais e demandas necessárias. No Capítulo 4 é feito um estudo comparativo, onde são descritos os testes realizados com as três abordagens técnicas alvo deste estudo, os resultados obtidos são apresentados, e uma análise detalhada dos dados gerados é realizada. Por fim, no Capítulo 6, conclusões gerais sobre o estudo feito e os testes

realizados são apresentadas, e são também levantadas propostas de estudo em trabalhos futuros.

2. Descrição de Tecnologias Web e Técnicas

2.1. A Web

A web se fundamenta fortemente nos protocolos TCP (camada de transporte), IP (camada de rede) e HTTP (camada de aplicação).

O protocolo TCP (*Transmission Control Protocol*) [12] tem sido utilizado há mais de vinte anos na Internet. Ele foi criado por dois pesquisadores chamados Vinton Cerf e Robert Khan na década de 70, antes mesmo da invenção dos PCs e estações de trabalho, antes das redes Ethernet e de outras tecnologias de redes locais, e logicamente, antes da *Web*. A princípio ele foi vinculado ao protocolo IP (*Internet Protocol*) [11], apresentado como TCP/IP.

"*O TCP é um protocolo de transferência confiável de dados que é implementado sobre uma camada de rede fim a fim não confiável (IP)*" [1].

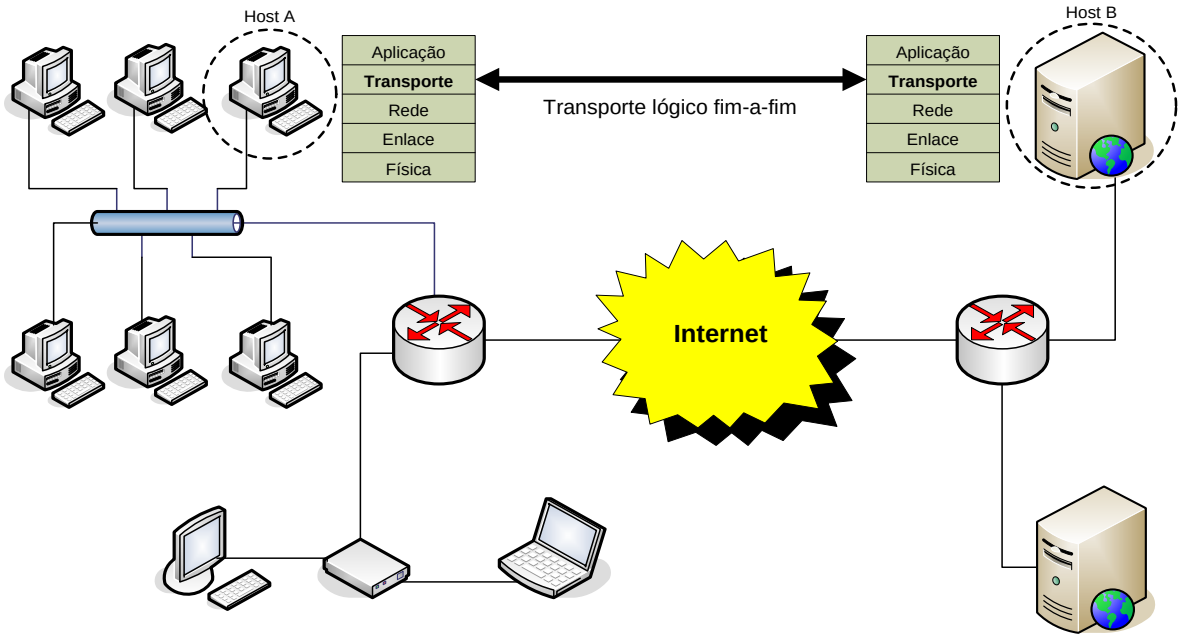


Figura 1: Exemplo de transporte lógico fim-a-fim na web usando o TCP/IP.

Na tabela abaixo pode ser visualizada uma lista de aplicações comuns na internet e seus protocolos de aplicação, onde a Web (WWW) tem papel essencial por ser a preponderante em termos de uso com acesso direto pelos usuários nos diversos tipos de dispositivos fixos ou móveis atualmente.

Tabela 1: Aplicações populares da Internet e seus protocolos de aplicação [1].

Aplicação	Protocolo de Aplicação
Correio eletrônico	SMTP [13, 14]
Acesso terminal remoto	Telnet [15]
Web (WWW)	HTTP [16]
Transferência de arquivos	FTP [17]
Servidor remoto de arquivos	NFS [18]
Multimídia em tempo real	Quase sempre proprietário (Ex.: *Real Networks*)

2.2. O Protocolo TCP

O protocolo TCP é orientado à conexão e fornece um serviço de transferência confiável de dados fim a fim entre *hosts* remotos.

A conexão TCP fornece transferência de dados *full-duplex*, ou seja, pode haver simultaneamente na rede segmentos enviados pelos dois lados conectados, gerando assim um melhor aproveitamento da rede [1].

Uma conexão TCP é sempre ponto-a-ponto, ou seja, só há o envolvimento de um remetente e um destinatário, ou um cliente e um servidor, diferente do que ocorre em conexões *multicast*, onde é possível ter um único remetente para vários destinatários, em uma única operação.

O protocolo TCP é orientado à conexão, o que significa que ele sempre estabelece uma sessão entre as partes comunicantes, de antes de enviar e receber segmentos dados.

O estabelecimento de uma conexão TCP se dá em três etapas, e é comumente conhecido como apresentação em três vias (*3-way handshake*), onde são enviados três segmentos especiais, entre cliente (quem solicita conexão) e servidor (quem responde a essa solicitação) da conexão, antes de quaisquer segmentos com carga útil de dados.

A finalização de uma conexão TCP se dá forma simples, com um pedido de finalização (FIN) enviado pelo cliente, reconhecimento (ACK) e envio do pedido também pelo servidor (outro FIN), com reconhecimento pelo cliente. Em seguida há uma temporização de fechamento. Após esse tempo todos os recursos estão totalmente liberados, inclusive os números de portas entre processos (*sockets*).

O TCP trabalha com *buffers* de transmissão e recepção, que são reservados durante o processo de estabelecimento da conexão, durante a apresentação dos *hosts*. De tempos em tempos o TCP retira dados dos *buffers*, seja para o envio (transmissão) seja para a recepção (entrega para a camada superior da pilha de protocolos – em se considerando o modelo Híbrido TCP/IP de 5 camadas, esta será a camada de aplicação).

A temporização de um segmento TCP enviado ou recebido é baseada na estimativa do RTT (tempo de ida e volta do pacote na rede). Conhecendo os tempos médios de ida e volta dos pacotes de dados, o TCP pode estipular estaticamente o valor da temporização sem que este seja longo demais, evitando baixa utilização da rede, nem curto demais, evitando retransmissões desnecessárias devido ao esgotamento prematuro da temporização.

A comunicação no TCP se dá entre processos de aplicação que rodam nos dois lados (*hosts*) envolvidos na conexão. Os *sockets* são portas de comunicação entre os processos de aplicação que estão rodando nos dois lados. Quando um *host* A deseja enviar um dado, usando o protocolo de transporte, para um *host* B (considerando que haja uma conexão entre eles), ele simplesmente direciona estes à porta de comunicação ou *socket* do *host B*, entregando assim os dados à aplicação final em B. O mesmo processo vale se o *host* B deseja enviar um dado para o *host* A.

Cada número de porta é um número de 16 bits que vai de 0 a 65535. Os números de porta bem conhecidos são os que vão de 0 a 1023 [1]. Alguns exemplos de números de porta bem conhecidos são: FTP (portas 20-dados e 21-controle), SSH

(porta 22), Telnet (porta 23), SMTP (porta 25), DNS (porta 53), HTTP (porta 80), POP3 (porta 110) e HTTPS (porta 443) [9].

A Figura a seguir mostra um exemplo de utilização de sockets em uma conexão TCP. Como pode ser visto, existe um processo de aplicação rodando em ambos os lados da conexão. Os sockets funcionam como portas de comunicação entre a aplicação e a camada de transporte. Esta última, por sua vez, faz uma conexão lógica fim a fim entre cliente e servidor, de modo transparente ao usuário.

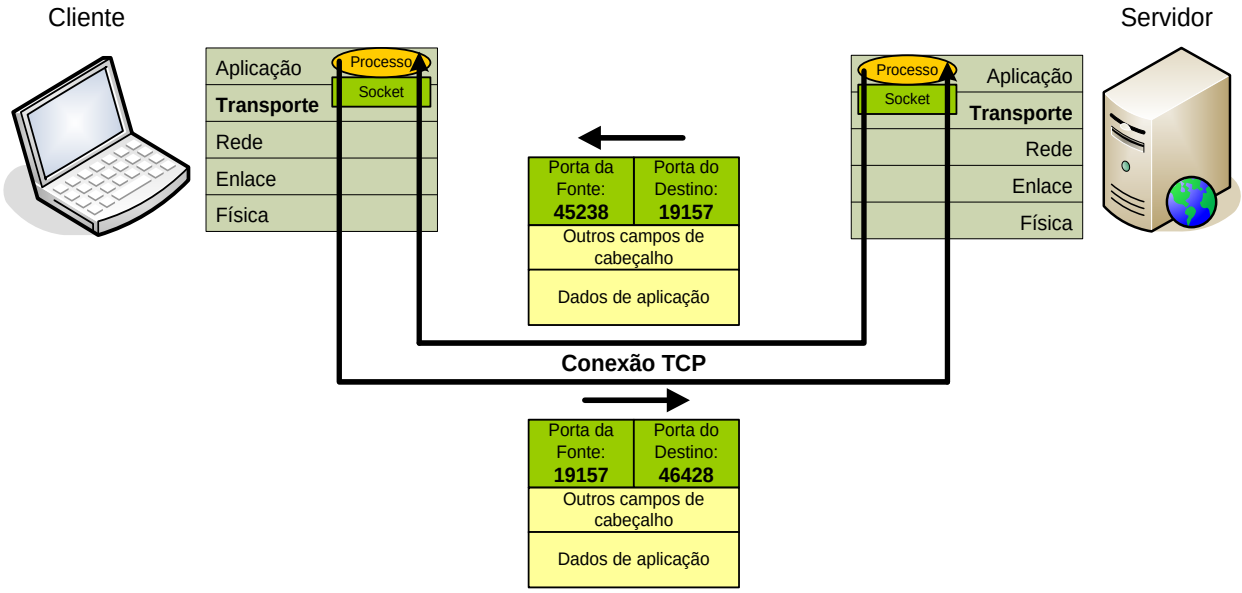


Figura 2: Exemplo do uso de *sockets* e processos de aplicação numa conexão.

Em resumo, uma conexão TCP é composta basicamente por *buffers* de transmissão e recepção, *sockets* de conexão (que são as portas entre processos de aplicação e transporte dos dois lados da conexão [1]), e variáveis que guardam o estado da conexão (variáveis de estado).

2.3. O Protocolo HTTP

O protocolo HTTP (*Hipertext Transfer Protocol*) [16] tem sido utilizado há quase 30 anos na Internet. Seu desenvolvimento foi iniciado por Tim Berners-Lee no CERN (em francês, *Conseil Européen pour la Recherche Nucléaire*) na década de 90. Tal protocolo teve papel fundamental e central no desenvolvimento da web

como se conhece hoje, sendo a base para a maioria das aplicações utilizadas através do browsers na web atualmente.

A Figura a seguir mostra a estrutura completa do pacote HTTP, seus campos, portas e tamanho em bits.

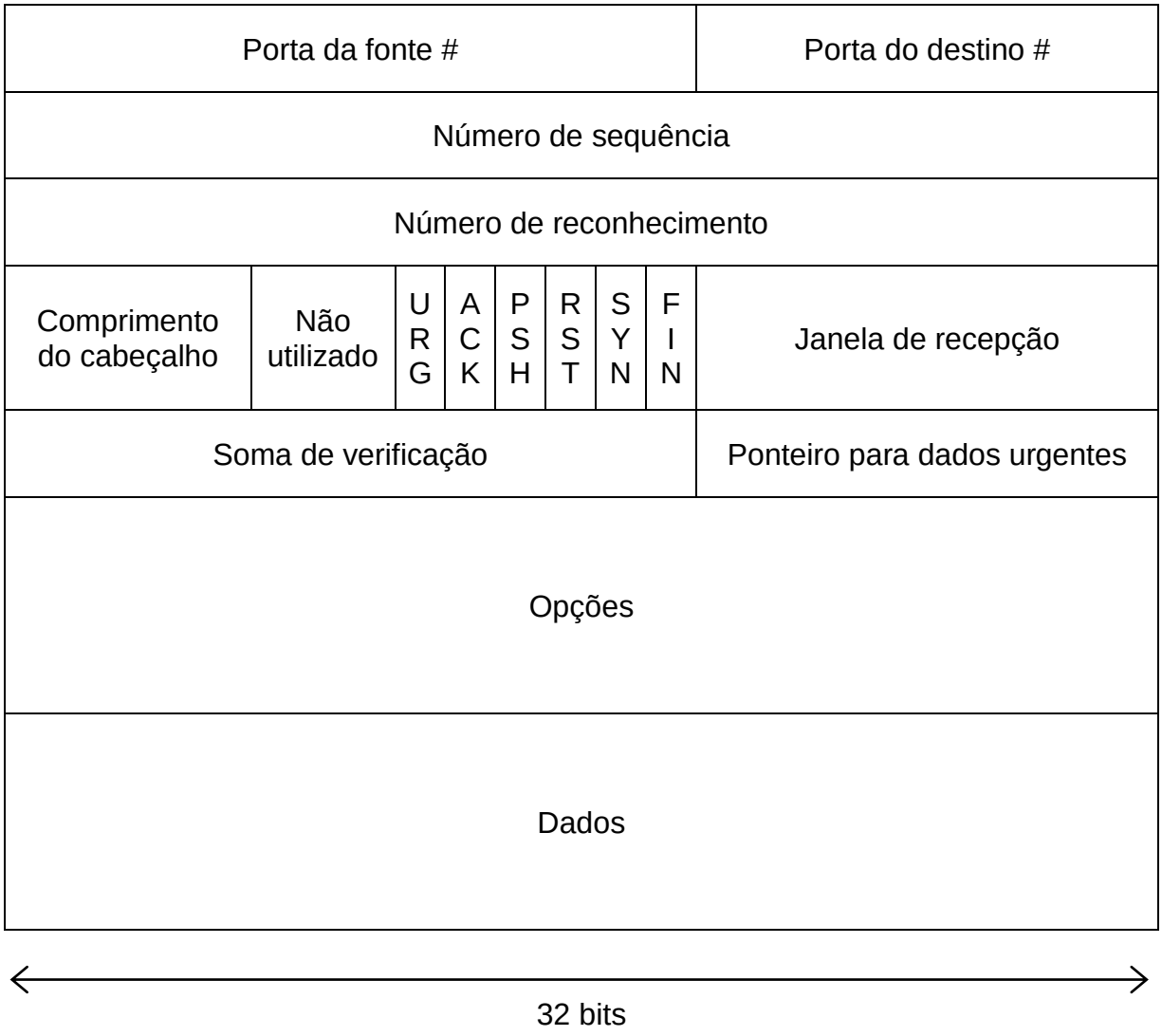


Figura 3: A estrutura de um pacote HTTP [1].

2.4. Sondagem Padrão ou Polling

Polling é o processo de verificar periodicamente o servidor para obter informações depois de realizar uma solicitação pelo cliente. Se a informação necessária estiver disponível, ela será enviada ao cliente na forma de resposta, caso contrário, o servidor enviará uma resposta vazia. Polling, numa página HTML é feito com a ajuda de funções Javascript como setInterval() e setTimeout().

2.5. Sondagem Longa ou Long Polling

No caso de Long Polling, o cliente faz uma solicitação ao servidor e, se a resposta estiver disponível, o servidor responde com as informações, como no Polling. Porém, se esta não está disponível, ele mantém o pedido até que as informações necessárias estejam disponíveis. Ao contrário da sondagem tradicional, onde o servidor teria respondido com uma resposta vazia.

2.6. O Protocolo WebSocket

O protocolo WebSocket, proposto através do RFC 6455 [7] de dezembro de 2011. Ele permite a comunicação bidirecional entre um cliente, executando código não confiável em um ambiente controlado, e um servidor remoto, que tenha optado por comunicações desse código. O modelo de segurança usado para isso é o modelo baseado em origem, que é comumente utilizado pelos navegadores web. O protocolo consiste em um “aperto de mão” de abertura de conexão (connection opening handshake), seguido por um enquadramento de mensagem básico, em camadas sobre TCP. O objetivo dessa tecnologia é fornecer um mecanismo para aplicações baseadas em navegador que precisam de comunicação bidirecional com servidores onde estas não dependem da abertura de várias conexões HTTP (por exemplo, usando XMLHttpRequest ou uso de blocos <iframe> e Long Polling).

Ao longo do tempo, a criação de aplicações web que precisam de comunicação bidirecional entre um cliente e um servidor (por exemplo: aplicativos de mensagens instantâneas e jogos) exigiu um uso excessivo de HTTP para realizar requisições ao servidor em busca de atualizações para que este enviasse notificações de resposta no sentido servidor-cliente utilizando chamadas HTTP distintas [19].

Isso resulta em vários problemas relacionados a performance e complexidade das conexões, podendo citar dentre eles:

- O servidor é forçado a usar várias conexões TCP subjacentes diferentes para cada cliente: uma para enviar informações para o cliente e uma nova para cada mensagem recebida.

- O protocolo de conexão tem uma alta sobrecarga, com cada mensagem cliente-servidor tendo um novo cabeçalho HTTP.

- O script do lado do cliente é forçado a manter um mapeamento das conexões de saída para a conexão de entrada para rastrear as respostas.

Uma solução mais simples seria usar uma única conexão TCP para o tráfego nas duas direções, ou seja, cliente-servidor e servidor-cliente. Isso é o que o protocolo WebSocket fornece. Combinado com a API WebSocket [20], ele fornece uma alternativa à sondagem (polling) HTTP para comunicação bidirecional de uma página web com um servidor remoto.

A mesma técnica pode ser usada para uma variedade de aplicações web: jogos, painéis de ações com atualização em tempo real, aplicações multiusuários com edição simultânea, interfaces de usuário expondo serviços do lado do servidor (back-end services) em tempo real, etc.

O protocolo WebSocket foi projetado para substituir as tecnologias de comunicação bidirecional existentes que usam o HTTP como uma camada de transporte para se beneficiar da infraestrutura pré-existente (proxies, filtragem, autenticação). Tais tecnologias foram implementadas como relações de troca entre eficiência e confiabilidade, porque o HTTP não foi inicialmente destinado a ser usado para comunicação bidirecional [19]. Este protocolo tenta endereçar os objetivos das tecnologias HTTP bidirecionais existentes no contexto da infraestrutura HTTP existente, e como tal, ele é projetado para funcionar nas portas HTTP 80 e 443 [9], bem como para suportar proxies e intermediários HTTP, mesmo que isso implique alguma complexidade específica para o ambiente atual. No entanto, o design não limita o WebSocket ao HTTP, e futuras implementações podem usar um método de conexão mais simples em uma porta dedicada sem reinventar todo o protocolo. Esse último ponto é importante porque os padrões de tráfego de mensagens interativas não correspondem muito ao tráfego HTTP padrão e podem induzir cargas incomuns em alguns componentes [7].

A possibilidade da existência de uma conexão única que não expira ou é terminada automaticamente após minutos ou horas após estabelecida é definida pelo RFC 6455 [7] onde se determina o uso de checagens (heartbeats), chamadas

verificadoras esporádicas, a fim de verificar a continuidade da conexão ou realizar a reconexão quando e se necessário.

A documentação técnica estabelece alguns quadros de controle para comunicar o estado sobre o WebSocket:

- Close: 0x8

- Ping: 0x9

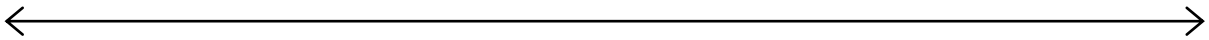
- Pong: 0xA

Ping e Pong são usados como heartbeats e permitem verificar se o cliente ainda está respondendo. "*Um quadro Ping pode servir como um keepalive ou como um meio de verificar se o endpoint remoto ainda está respondendo*" [7].

Quando o cliente recebe um Ping, um Pong deve ser enviado de volta ao servidor. *"Após o recebimento de um quadro Ping, um ponto final deve enviar um quadro Pong em resposta, a menos que já tenha recebido um quadro Close. Deve responder com quadro de Pong assim que for prático"* [7].

A Figura a seguir mostra a estrutura completa do pacote WebSocket, seus campos, portas e tamanho em bits.

<table>
<tr><td colspan="8">Porta da fonte #</td><td colspan="8">Porta do destino #</td></tr>
<tr><td colspan="16">Número de sequência</td></tr>
<tr><td colspan="16">Número de reconhecimento</td></tr>
<tr><td colspan="4">Comprimento do cabeçalho</td><td colspan="3">Não utilizado</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="4">Janela de recepção</td></tr>
<tr><td colspan="8">Soma de verificação</td><td colspan="8">Ponteiro para dados urgentes</td></tr>
<tr><td colspan="16">Dados</td></tr>
</table>



32 bits

Figura 4: A estrutura de um pacote WebSocket [7].

3. Algumas Possibilidades da Web em Tempo Real

Alguns tipos de aplicação podem ser projetados para utilizar a web em tempo real ou algo próximo disso, por necessitar de uma resposta rápida ou quase imediata para determinadas ações. Abaixo são dados alguns exemplos de possíveis usos:

3.1. Cenário 1: Execução de Processo Background Longo no Servidor

Um dado sistema pode ter a necessidade de executar um processo longo com duração desconhecida ou impossível de ser calculada com exatidão, como por exemplo a geração de um grande relatório, onde sejam necessários de alguns segundos a alguns minutos de processamento no servidor.

A abordagem tradicional seria a de realizar uma sondagem curta ou longa, para verificar quando tal operação de longa duração foi finalizada, afim de atualizar, por exemplo, uma página de resultados.

Utilizando uma tecnologia de tempo real de conexão única, tal problema se resumiria ao lado servidor responder ou realizar a notificação ao lado cliente para atualizar os dados imediatamente ao finalizar o processamento.

3.2. Cenário 2: Jogo On-Line em Tempo Real

Um sistema de jogo on-line pode possuir múltiplos usuários simultaneamente conectados ao servidor e estes poderão realizar diversas operações multiusuário simultâneas, porém cada usuário/jogador deve receber ou verificar os resultados das ações dos outros usuários/jogadores e do ambiente imediatamente, e caso o tempo de processamento no servidor ou o tempo de transporte dos dados seja insatisfatório, os atrasos de processamento inviabilizam a utilização de tal jogo.

Uma abordagem mais tradicional seria a de realizar múltiplas sondagens curtas ou longas para cada usuário partindo do lado cliente, de forma a verificar quando alguma ação do ambiente ou dos outros usuários/jogadores foi realizada, afim de atualizar a interface gráfica para mostrar a reação ou resultado visual gerado.

Ao se utilizar uma tecnologia de tempo real com baixa sobrecarga de rede nesse cenário, além de viabilizar o processo dinâmico e imediato do jogo, possibilitaria a otimização dos recursos de rede, já que nos jogos multiplayer massivos online atuais, como por exemplo Slither.io [21] ou Agar.io [22], podem existir de dezenas a até milhares de usuários/jogadores simultâneos.

3.3. Cenário 3: Sistema de Votação via Web

Um sistema de votação on-line consiste normalmente num sistema com a pauta de votação conduzida ou direcionada por um presidente de mesa/sessão e pode possuir de poucas dezenas a algumas centenas de usuário votantes conectados ao servidor e estes poderão realizar diversas operações em paralelo, com atualização dinâmica de um placar de votação mostrando os resultados atuais, preferencialmente de forma imediata casa haja usuários apuradores ou verificadores ou mesmo para validação dos componentes da votação, e caso o número de requisições e respostas seja muito alto neste sistema online, poderiam ocorrer gargalos na infraestrutura de rede e no processamento que poderiam inviabilizar a votação.

A abordagem tradicional para o placar na interface de todos os usuários conectados seria a de realizar múltiplas sondagens a partir do lado cliente, de forma a obter os resultados imediatamente após cada voto ser computado. Por exemplo, se as eleições presidenciais pudessem ser realizadas online haveriam milhões de brasileiros conectados simultaneamente ao sistema como votantes ou apuradores, gerando milhões sondagens (polling) a cada N segundos, sufocando totalmente a infraestrutura nacional de telecomunicações.

Ao se utilizar uma tecnologia de tempo real com baixa sobrecarga de rede e conexão única para cada usuário neste cenário, além de tornar o processo devido à maior velocidade e melhor performance, possibilitaria a otimização dos recursos de rede, já que seriam utilizadas conexões únicas para toda a comunicação necessária por cada usuário com o servidor, que uma vez abertas só seriam fechadas após o fim da votação ou desconexão individual de cada usuário.

3.4. Cenário 4: Barra de Notificação de Rede Social

Um dado sistema de rede social pode ter a necessidade de executar verificações constantes de novas mensagens ou novas amizades que solicitaram adição ou outras interações realizadas entre usuário as quais necessitam ser notificadas, de preferência imediatamente.

A abordagem mais comum seria a de realizar uma sondagem longa ao servidor a cada N segundos ou minutos, para verificar quando tais operações de interação ocorreram, afim de atualizar a barra de notificações.

Utilizando uma tecnologia de tempo real de conexão única, este problema se resumiria a cada usuário conectado ao sistema obter uma conexão única com o servidor e, quando necessário fosse, o servidor realizaria notificações push na direção servidor-cliente ou notificando os clientes conectados para atualizar barra de notificações devido a novos resultados a exibir.

3.5. Cenário 5: Sistema de Bate-Papo (Chat)

Um sistema de bate-papo via web normalmente tem a necessidade conectar múltiplos usuários, os quais realizam transferências de informações de forma bidirecional, ou seja, enviam e recebem mensagens.

A abordagem tradicional neste cenário seria a de realizar uma sondagem curta, para verificar quando novas mensagens foram recebidas, e assim enviar novas mensagens interativamente.

Utilizando uma tecnologia de tempo real de conexão única, tal problema se resumiria ao lado servidor responder com a nova mensagem propriamente dita assim que a tiver uma atualização ou a notificação ao lado cliente para atualizar os dados de mensagem na interface do usuário imediatamente após o servidor receber uma nova interação.

3.6. Cenário 6: Gráfico de Ações com atualização dinâmica

Um dado sistema online de visualização, compra e venda de ações na bolsa de valores normalmente tem a necessidade atualizar seus dados de gráficos de valores na interface do usuário assim que receba uma atualização, a qual normalmente tem seu momento de ocorrência desconhecido. Isso gera um sistema de escuta permanente do servidor pelo cliente na busca por novas atualizações.

A abordagem tradicional seria a de realizar sondagens curtas pelos clientes conectados, tão curtas quanto seja possível, para a atualização de resultados, onde a demora de 1 segundo na exibição de valores pode gerar prejuízos da ordem de milhões aos investidores e outros interessados.

Utilizando uma tecnologia de tempo real de conexão única, o problema se resumiria ao lado servidor responder atualizando ao lado cliente imediatamente a cada operação de compra ou venda processada, plotando novo ponto no gráfico e valores da última operação realizada.

4. Estudo Comparativo

4.1. Objetivo

O objetivo deste estudo é o de comparar as técnicas Polling, Long Polling e WebSocket, mostrando seus tempos de execução, número de chamadas realizadas para completar o processo.

Neste estudo foi escolhido o cenário 1 citado anteriormente, que era o de realizar a execução de um processo longo com duração desconhecida em background no servidor, por ter menor complexidade de implementação e maior simplicidade de análise, simples entendimento e de maior facilidade de execução com os recursos disponíveis.

4.2. Recursos Utilizados

Nos experimentos realizados neste trabalho foram utilizados computadores com os seguintes recursos:

Especificações do Computador Cliente [2]:

- Sistema Operacional: Windows 10 Home 64 bits
- Navegador: Google Chrome Versão 64.0.3282.186 64 bits
- Placa Mãe: Intel Kaby Lake-U Premium PCH
- Processador: Intel® Core™ i3-6006U CPU 2.0 GHz, 3MB L3 Cache
- Memoria: DDR4 8 GB (padrão)
- Armazenamento: 1 Drive 128 GB SSD + 1 Disco Rígido 1 TB
- Placa Gráfica: Intel® HD Graphics 520 - DDR4 Shared graphics memory
- Conectividade: 802.11ac wireless LAN / Gigabit LAN

Especificações do Computador Servidor [2]:

- Sistema Operacional: Windows 10 Home 64 bits
- Placa Mãe: Intel Kaby Lake-U Premium PCH
- Processador: Intel® Core™ i3-6006U CPU 2.0 GHz, 3MB L3 Cache
- Memoria: DDR4 8 GB (padrão)

- Armazenamento: 1 Drive 128 GB SSD + 1 Disco Rígido 1 TB
- Placa Gráfica: Intel® HD Graphics 520 - DDR4 Shared graphics memory
- Conectividade: 802.11ac wireless LAN / Gigabit LAN

Para executar os testes foi utilizada uma aplicação Web com HTML e Javascript no front-end e Java [3] no back-end fazendo uso do framework Spring Boot [4] com Servidor Apache Tomcat [5] integrado, possibilitando assim um desenvolvimento mais rápido, com menos configurações e uma análise de desempenho bastante eficiente através da geração de arquivos do tipo CSV (comma-separated values) para armazenamento dos resultados e futura geração de gráficos.

Os gráficos foram gerados com a utilização da ferramenta Excel 2016 da Microsoft, que é um aplicativo comumente utilizado para cálculos, manipulação ou criação de planilhas e gráficos de coleções de dados.

4.3. Configuração dos Experimentos

Servidor HTTP Apache Tomcat

Threads no Servidor = 1 thread principal + 1 thread para processo Fibonacci

Temporização de Polling = 2 seg

Temporização de Long Polling = 30 seg

Tempo de resposta do servidor = entre 0.01 e 92 seg

Nos testes foi utilizada uma topologia que consiste em dois computadores interligados através de um roteador em rede Wi-Fi de 1 Gbps. Nessa rede foi adotada a arquitetura cliente-servidor na conexão para a transmissão/recepção de dados.

4.4. Testes Realizados - Descrição dos Experimentos

Para realizar os testes foram criados alguns artefatos de software para automatização do processo, que contêm a execução dos testes propriamente ditas no lado cliente e lado servidor (vide Apêndices de A até I) da conexão, além da consolidação dos valores numéricos brutos, e geração dos valores finais escalonados (base para a geração dos gráficos apresentados).

Foram realizadas 10 sequências de conexões com transferências contínuas de dados nos testes, até o fim de 50 rodadas de Fibonacci, de forma a obter 50 tempos de execução distintos de um processo back-end no servidor, utilizando-se cada um dos 3 métodos avaliados.

No total foram realizados 150 testes, 50 para cada abordagem testada, acrescidos das repetições, com a geração de logs de execução em arquivos CSV, executados num tempo total aproximado de 3 horas contínuas de testes.

4.4.1. Experimento 1: Verificação de Andamento de Processo Longo em Background no Servidor com Polling

Abaixo segue uma imagem do sistema experimental proposto executando a verificação do Andamento de um Processo Longo em Background no Servidor experimental com Polling:

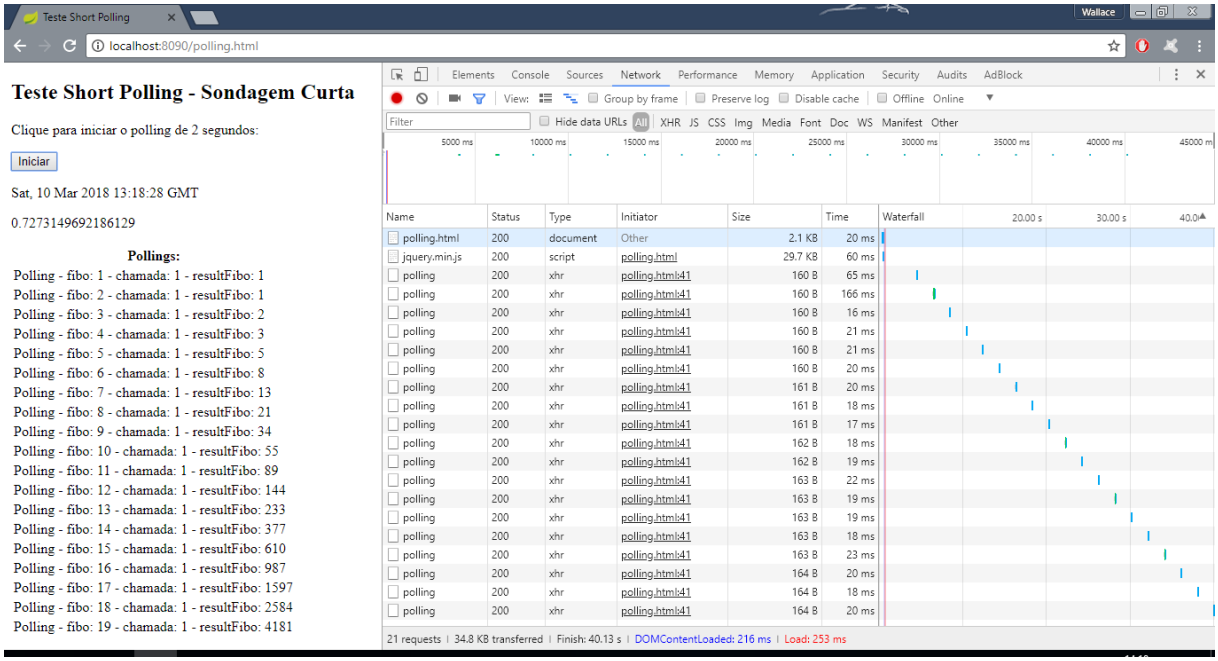


Figura 5: Experimento com Polling via browser, vide apêndice A.

4.4.2. Experimento 2: Verificação de Andamento de Processo Longo em Background no Servidor com Long Polling

Abaixo segue uma imagem do sistema experimental proposto executando a verificação do Andamento de um Processo Longo em Background no Servidor experimental com Long Polling:

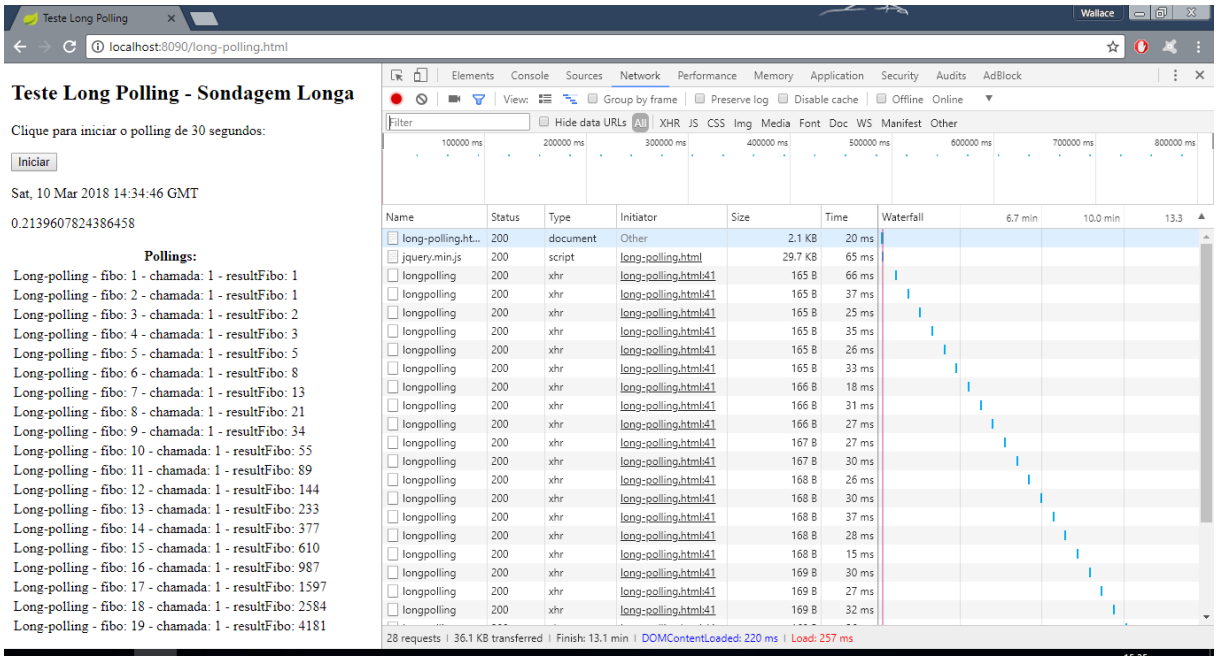


Figura 6: Experimento com Long Polling via browser, vide apêndice B.

4.4.3. Experimento 3: Verificação de Andamento de Processo Longo em Background no Servidor com WebSocket

Abaixo segue uma imagem do sistema experimental proposto executando a verificação do Andamento de um Processo Longo em Background no Servidor experimental com WebSocket:

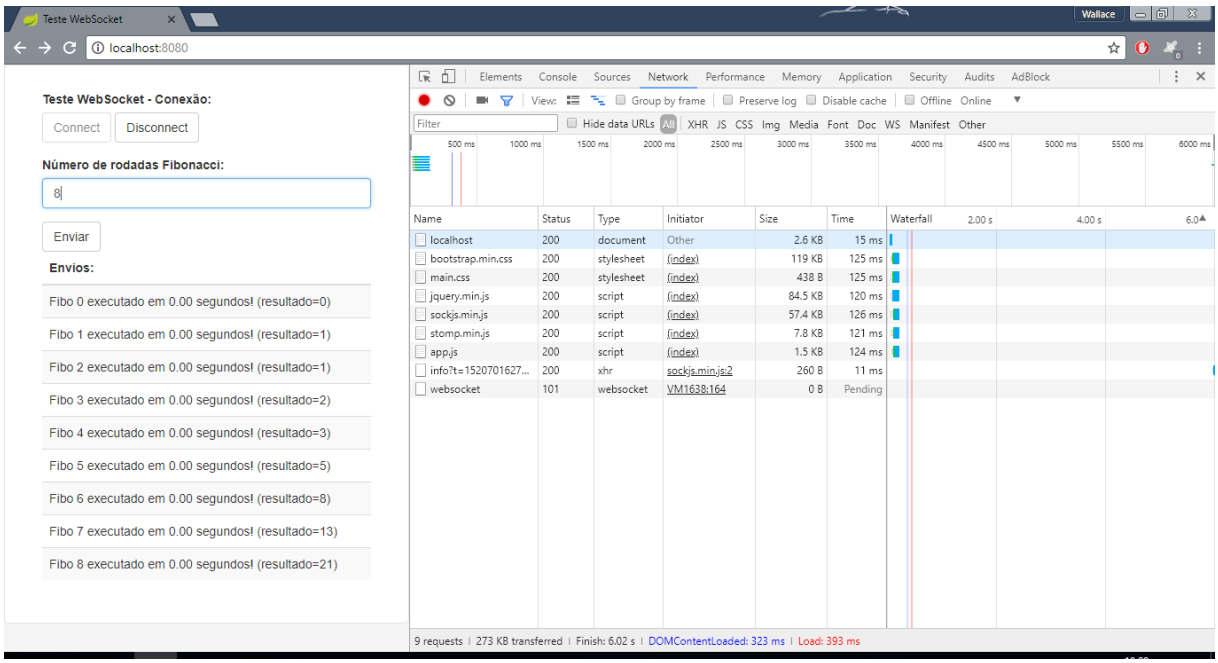


Figura 7: Experimento com WebSocket via browser, vide apêndice C.

4.5. Análise dos Resultados

Durante os testes realizados cada método de requisição e resposta foi submetido a diferentes cenários de tempo de execução de processo servidor. Nesses testes, foi possível comprovar a eficiência do uso de WebSocket frente aos métodos tradicionais de sondagem (polling).

Como já mencionado, foram colhidas de 10 amostras (transferências de dados) com 50 execuções cada. Para cada amostra dos experimentos foi utilizado um intervalo de confiança de 95% para a média (vide Apêndice Q), o que foi representado nos gráficos pelas barras de erro verticais.

Os resultados obtidos nos experimentos realizados são mostrados nas 3 Figuras a seguir.

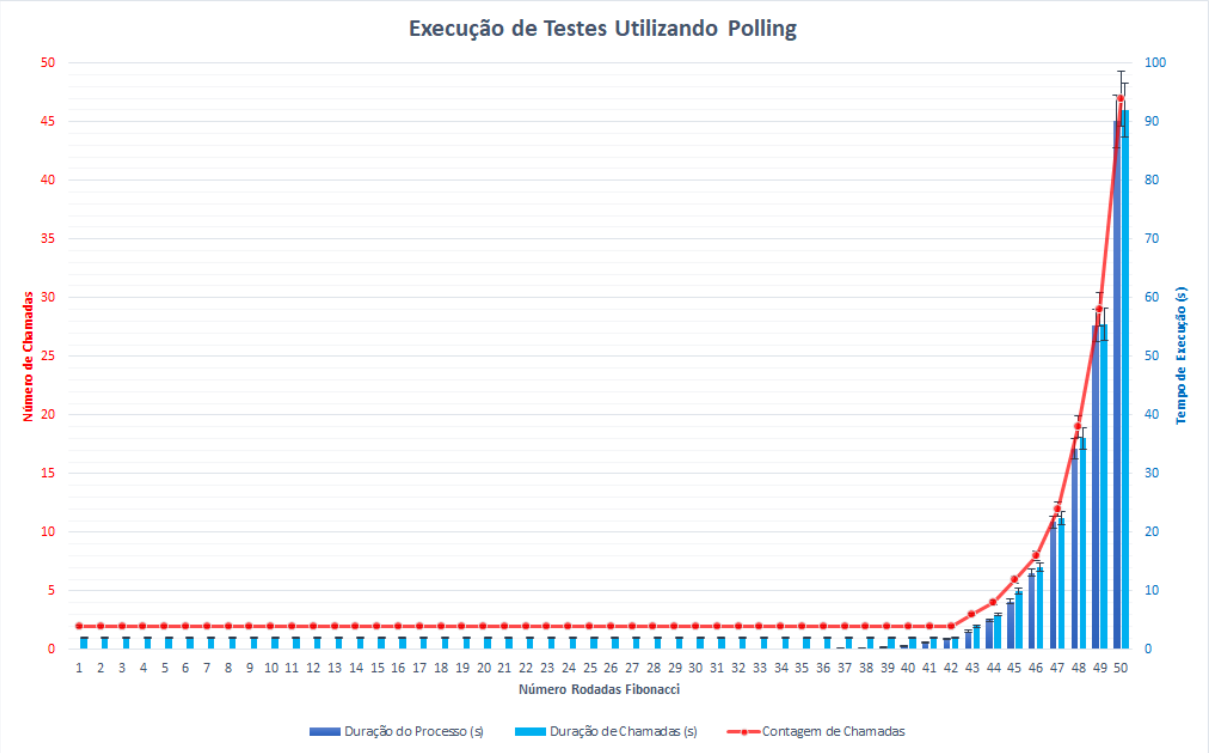


Figura 8: Gráfico com dados de testes utilizando a técnica Polling.

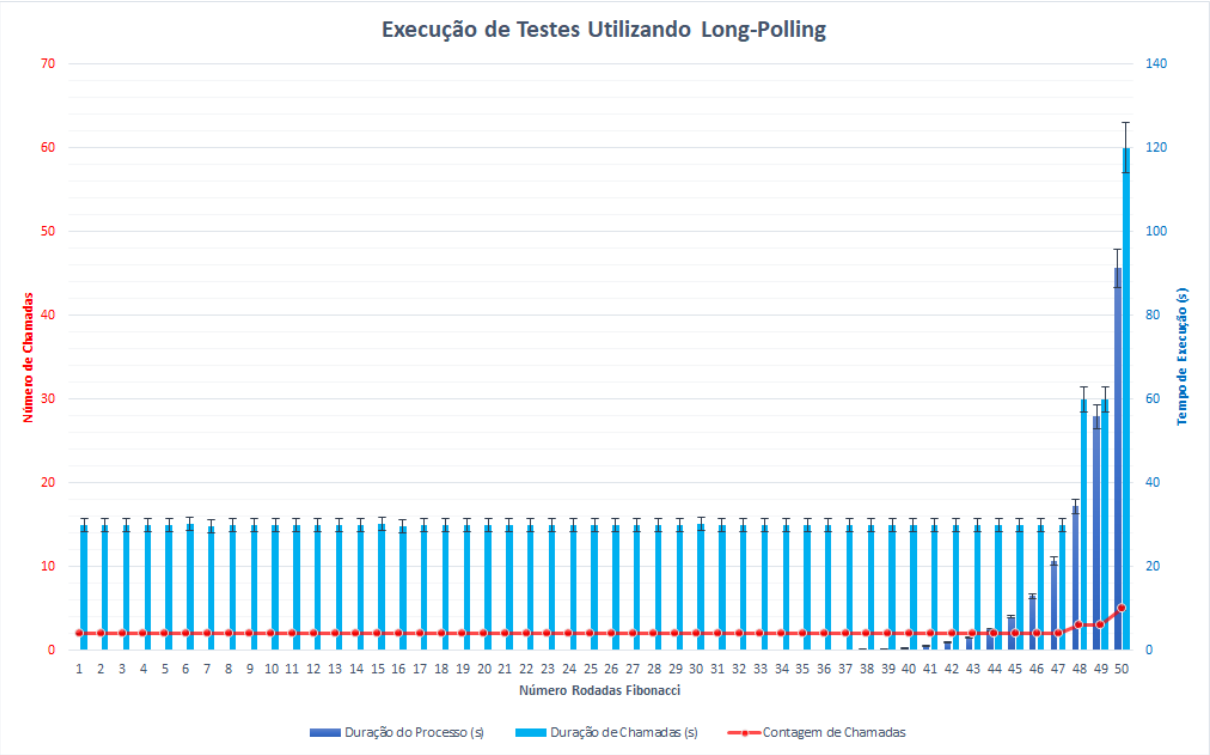


Figura 9: Gráfico com dados de testes utilizando a técnica Long Polling.

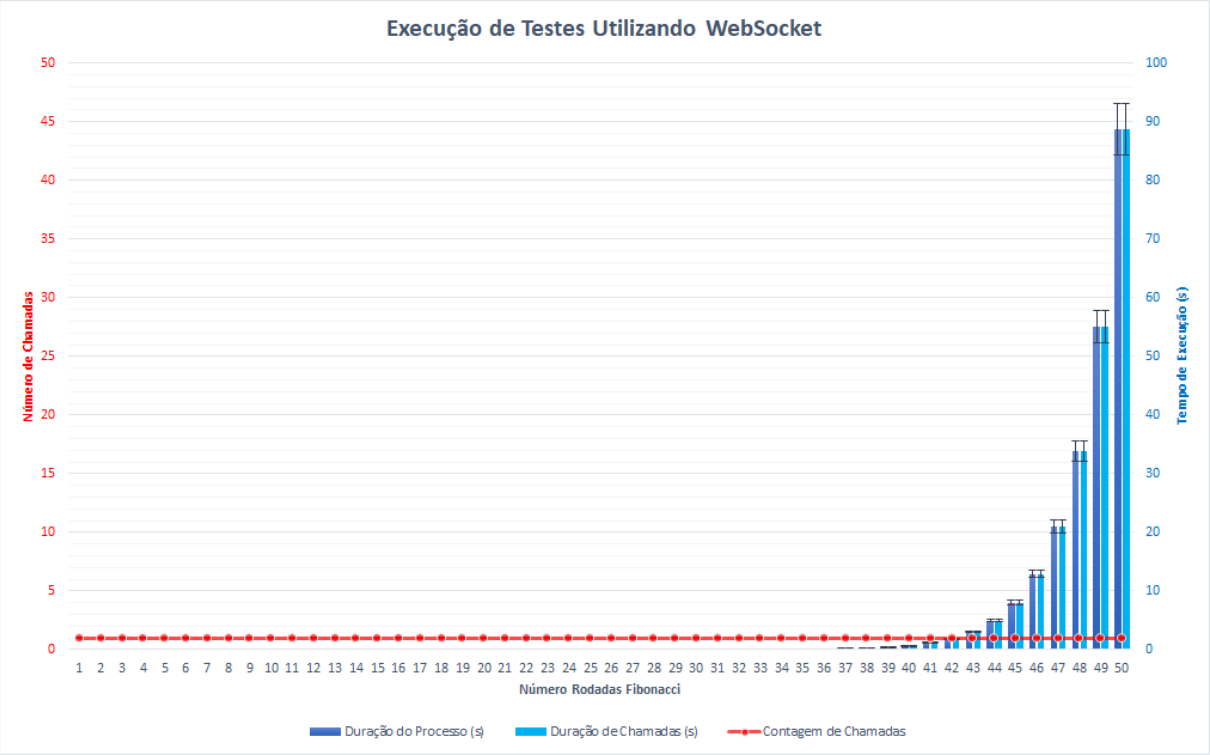


Figura 10: Gráfico com dados de testes utilizando a técnica WebSocket.

Os resultados mostram primeiramente que, como esperado, que o método Polling demonstra claramente uma alta taxa de utilização da rede através da repetição de excessiva de requisições, a qual aumenta conforme aumenta o tempo de processamento. Com o uso de WebSocket esse problema não ocorre, pois somente uma conexão única é estabelecida e mantida, com toda a comunicação necessária sendo realizada através desta.

Os gráficos mostram que, na comparação Polling x Long Polling, pelo simples fato de ser utilizada a sondagem longa no segundo cenário já houve um ganho considerável no numero de chamadas necessárias para verificar a completude do processamento em lado servidor, embora ainda haja um número considerável em se considerando a sobre carga de rede em um possível cenário com muitos usuários conectados.

Passando ao cenário com uso de WebSocket a maior otimização de recursos se torna visível, pois somente uma conexão foi aberta e mantida durante todo o tempo do processamento, não havendo necessidade de chamadas secundários, e sem outros envios e respostas de modo a gerar sobrecarga da infraestrutura utilizada.

5. Conclusões Gerais

Neste trabalho foi avaliado o desempenho de dois métodos de requisição e resposta comumente utilizados na web atualmente com protocolo alternativo cujo o uso está se tornando cada vez mais popular dada sua alta performance em conexões web de alta velocidade. Foi demonstrado que é possível utilizar a rede de maneira mais eficiente, com o uso do protocolo WebSocket.

Como problema conhecido, os métodos Polling e Long Polling sobrecarregam a rede com requisições excessivas em cenários onde manter uma conexão persistente não é possível ou viável. Este problema se refletiu claramente durante os testes realizados. Os métodos de sondagem com requisições sucessivas se comportaram de forma danosa em conexões de alta velocidade, não atendendo bem às demandas de performance atuais, por utilizar a banda da rede de forma ineficaz.

Por outro lado, o protocolo alternativo ao HTTP, o WebSocket, se mostrou como uma solução bastante eficiente para resolver o problema de má utilização da rede, já que apenas uma conexão ficou aberta durante todo o tempo dos testes, fazendo envios mínimos com seu pacote de dados reduzido. Os resultados experimentais demonstraram que é possível obter um resultado cerca de 100 vezes mais eficaz que os atingidos pelo método Polling, e 10 vezes mais eficaz do que o atingido pelo Long Polling, em cenários idênticos e com tempos de execução médios ou longos. Porém, para a execução de processos de curta duração, os resultados obtidos pelos três métodos avaliados demonstraram uma tendência de se igualar, como esperado.

Embora implementadas sobre uma arquitetura simples em rede local, as conexões de teste dos métodos em questão puderam ser avaliadas de forma consideravelmente eficiente, sem restrições de implementação ou teste, levando em conta diferentes cenários emulados, atingindo o objetivo proposto por este trabalho.

Como proposta de trabalhos futuros nesse assunto, sugere-se estudos mais aprofundados das conexões web utilizadas com a utilização de servidor remoto via internet e com tráfegos de fundo reais, com suas taxas e atrasos variados, causando efeitos de queda de desempenho nas conexões principais. Sugere-se ainda o estudo de cenários de múltiplas conexões simultâneas, testando o efeito de trafego paralelo

na performance. Testes com o uso de HTTP/2 também são recomendados, em cenário similares, já que este oferece transferência de dados binário e de alta performance, o que pode resultaria em conexões de altíssima performance, já que todo o trafego das conexões sugeridas seria ainda comprimido.

6. Referências Bibliográficas

[1] Kurose, James F. & Ross, Keith W. – “Redes de Computadores e a Internet: uma abordagem top-down” – 3ª Edição – Editora Pearson Addison Wesley – São Paulo, 2006.

[2] Acer – Especificações da Configuração do Acer Aspire 3 modelo A315-51-33TQ – URL: https://www.acer.com/ac/de/DE/content/model/NX.GNPEG.022 – Último acesso: Março/2018.

[3] Java – URL: https://www.java.com/pt_BR/ – Último acesso: Março/2018

[4] Spring Boot – URL: https://projects.spring.io/spring-boot/ – Último acesso: Março/2018

[5] Apache Tomcat – URL: http://tomcat.apache.org/ – Último acesso: Março/2018

[6] Costa, Luís Henrique M. K.; Rezende, José F. & Rubinstein, Marcelo G. – Artigo: “Avaliação Experimental e Simulação do Protocolo TCP em Redes de Alta Velocidade” – XXII Simpósio Brasileiro de Telecomunicações - SBrT'05, Campinas, SP – URL: https://goo.gl/1NTxjJ – Último acesso: Março/2018.

[7] I. Fette, Google Inc., A. Melnikov & Isode Ltd. – “The WebSocket Protocol”, RFC 6455, Dezembro 2011 – URL: https://tools.ietf.org/html/rfc6455 – Último acesso: Março/2018.

[8] Informações Gerais sobre a Tecnologia do Protocolo WebSocket – URL: http://www.websocket.org/index.html – Último acesso: Março/2018.

[9] IANA (Internet Assigned Names Authority) – “TCP and UDP Port Numbers” – URL: http://www.iana.org/assignments/port-numbers – Último acesso: Março/2018.

[10] R. K. Jain – “The art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling” – Editora John Wiley & Sons Inc. – USA, 1991.

[11] J. Postel – “Internet Protocol: DARPA Internet Program Protocol Specification”, RFC 791, Setembro 1981 – URL: http://www.rfc-editor.org/rfc/rfc791.txt – Último acesso: Março/2018.

[12] J. Postel – “Transmission Control Protocol”, RFC 793, Setembro 1981 – URL: http://www.rfc-editor.org/rfc/rfc793.txt – Último acesso: Março/2018.

[13] J. Postel – “Simple Mail Transfer Protocol”, RFC 821, Agosto 1982 – URL: http://www.rfc-editor.org/rfc/rfc821.txt – Último acesso: Março/2018 – Obsoleto pelo RFC 2821.

[14] J. Klensin – “Simple Mail Transfer Protocol”, RFC 2821, Abril 2001 – URL: http://www.rfc-editor.org/rfc/rfc2821.txt – Último acesso: Março/2018.

[15] J. Postel & J. Reynolds – “TELNET Protocol Specification”, RFC 854. Maio 1993 – URL: http://www.rfc-editor.org/rfc/rfc854.txt – Último acesso: Março/2018.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, R. Feilding – “Hypertext Transfer Protocol - HTTP/1.1”, RFC 2616, Junho 1999 – URL: http://www.rfc-editor.org/rfc/rfc2616.txt – Último acesso: Março/2018.

[17] J. Postel & J. Reynolds – “File Transfer Protocol (FTP)”, RFC 959, Outubro 1985 – URL: http://www.rfc-editor.org/rfc/rfc959.txt – Último acesso: Março/2018.

[18] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck – “Network File System (NFS) version 4 Protocol”, RFC 3530, Abril 2003 – URL: ftp://ftp.rfc-editor.org/in-notes/rfc3530.txt – Último acesso: Março/2018.

[19] Loreto, et al. – “Known Issues and Best Practice for the Use of Long Polling and Streaming in Bidirectional HTTP”, RFC 6202, Abril 2011 – URL: https://tools.ietf.org/html/rfc6202 – Último acesso: Abril/2018.

[20] Hickson, I., "The WebSocket API", W3C Working Draft, Setembro 2011 – URL: https://www.w3.org/TR/websockets/ – Último acesso: Abril/2018.

[21] Howse, S., “Jogo Online Slither.io", Game Multiplayer Massivo Online – URL: http://slither.io/ – Último acesso: Abril/2018.

[22] Valadares, M., “Jogo Online Agar.io", Game Multiplayer Massivo Online – URL: http://agar.io/ – Último acesso: Abril/2018.

7. Apêndices

7.1. Apêndice A: Página para testes Polling no cliente

http://localhost:8090/polling.html

```
<!--
################################################################################
#                                                                              #
# Código fonte para executar o teste com Polling ou sondagem padrão.           #
#                                                                              #
# Projeto de Pós-Graduação - Engenharia de Software - Poli/UFRJ                #
#                                                                              #
# Autor: Wallace de Souza Espindola                                            #
#                                                                              #
################################################################################
-->
<!DOCTYPE html>
<html>
<head>
<title>Teste Short Polling</title>
<script
	src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
<body>
	<h2>Teste Short Polling - Sondagem Curta</h2>
	<p>Clique para iniciar o polling de 2 segundos:</p>
	<button onclick="myPollFunction()">Iniciar</button>
	<script>
		function myPollFunction(){
			setInterval(function(){
				document.getElementById('parte1').innerHTML = (new
Date()).toUTCString();
				document.getElementById('parte2').innerHTML = Math.random();
				var xhttp = new XMLHttpRequest();
				xhttp.onreadystatechange=function() {
					if (this.readyState == 4 && this.status == 200) {
						$("#polling").append("<tr><td>" +
this.responseText + "</td></tr>");
					}
				};
				xhttp.open("GET", "http://localhost:8090/polling", true);
				xhttp.send();

			}, 2000); // 2 segundos
		}
	</script>
	<p id="parte1"></p>
	<p id="parte2"></p>
	<div>
		<table id="conversation">
			<thead>
				<tr>
					<th>Pollings:</th>
				</tr>
			</thead>
			<tbody id="polling">
			</tbody>
		</table>
	</div>
</body>
</html>
```

7.2. Apêndice B: Página para testes Long Polling no cliente

http://localhost:8090/long-polling.html

```
<!--
###############################################################################################
#                                                                                             #
# Código fonte para executar o teste com Long Polling ou sondagem longa.                      #
#                                                                                             #
# Projeto de Pós-Graduação - Engenharia de Software - Poli/UFRJ                               #
#                                                                                             #
# Autor: Wallace de Souza Espindola                                                           #
#                                                                                             #
###############################################################################################
-->
<!DOCTYPE html>
<html>
<head>
<title>Teste Long Polling</title>
<script
	src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
<body>
	<h2>Teste Long Polling - Sondagem Longa</h2>
	<p>Clique para iniciar o polling de 30 segundos:</p>
	<button onclick="myPollFunction()">Iniciar</button>
	<script>
		function myPollFunction() {
			setInterval(function() {
				document.getElementById('parte1').innerHTML = (new Date())
						.toUTCString();
				document.getElementById('parte2').innerHTML = Math.random();
				var xhttp = new XMLHttpRequest();
				xhttp.onreadystatechange = function() {
					if (this.readyState == 4 && this.status == 200) {
						$("#polling").append(
								"<tr><td>" + this.responseText +
"</td></tr>");
					}
				};
				xhttp.open("GET", "http://localhost:8090/longpolling", true);
				xhttp.send();

			}, 30000); // 30 segundos
		}
	</script>
	<p id="parte1"></p>
	<p id="parte2"></p>
	<div>
		<table id="conversation">
			<thead>
				<tr>
					<th>Pollings:</th>
				</tr>
			</thead>
			<tbody id="polling">
			</tbody>
		</table>
	</div>
</body>
</html>
```

7.3. Apêndice C: Página para testes WebSocket no cliente

http://localhost:8080/index.html

```
<!--
################################################################################
#                                                                              #
# Código fonte para executar o teste com WebSocket.                            #
#                                                                              #
# Projeto de Pós-Graduação - Engenharia de Software - Poli/UFRJ                #
#                                                                              #
# Autor: Wallace de Souza Espindola                                            #
#                                                                              #
################################################################################
-->
<!DOCTYPE html>
<html>
<head>
<title>Teste WebSocket</title>
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet">
<link href="/main.css" rel="stylesheet">
<script src="/webjars/jquery/jquery.min.js"></script>
<script src="/webjars/sockjs-client/sockjs.min.js"></script>
<script src="/webjars/stomp-websocket/stomp.min.js"></script>
<script src="/app.js"></script>
</head>
<body>
        <noscript>
                <h2 style="color: #ff0000">Seems your browser doesn't support
                        Javascript! Websocket relies on Javascript being enabled. Please
                        enable Javascript and reload this page!</h2>
        </noscript>
        <div id="main-content" class="container">
                <div class="row">
                        <div class="col-md-6">
                                <form class="form-inline">
                                        <div class="form-group">
                                                <label for="connect">Teste WebSocket -
Conexão:</label><br>
                                                <button id="connect" class="btn btn-default"
type="submit">Connect</button>
                                                <button id="disconnect" class="btn btn-default"
type="submit" disabled="disabled">Disconnect</button>
                                        </div>
                                </form>
                        </div>
                        <div class="col-md-6">
                                <form class="form-inline">
                                        <div class="form-group">
                                                <label for="name">Número de rodadas Fibonacci:
</label> <input type="text" id="name" class="form-control" placeholder="digite aqui...">
                                        </div>
                                        <button id="send" class="btn btn-default"
type="submit">Enviar</button>
                                </form>
                        </div>
                </div>
                <div class="row">
                        <div class="col-md-12">
                                <table id="conversation" class="table table-striped">
                                        <thead>
                                                <tr>
                                                        <th>Envios:</th>
                                                </tr>
                                        </thead>
                                        <tbody id="greetings">
                                        </tbody>
                                </table>
                        </div>
                </div>
        </div>
</body>
</html>
```

7.4. Apêndice D: Javascript para testes WebSocket cliente

app.js

```
var stompClient = null;

function setConnected(connected) {
    $("#connect").prop("disabled", connected);
    $("#disconnect").prop("disabled", !connected);
    if (connected) {
        $("#conversation").show();
    }
    else {
        $("#conversation").hide();
    }
    $("#greetings").html("");
}

function connect() {
    var socket = new SockJS('/gs-guide-websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/greetings', function (greeting) {
            showGreeting(JSON.parse(greeting.body).content);
        });
    });
}

function disconnect() {
    if (stompClient !== null) {
        stompClient.disconnect();
    }
    setConnected(false);
    console.log("Disconnected");
}

function sendName() {
    stompClient.send("/app/hello", {}, JSON.stringify({'name':
$("#name").val()}));
}

function showGreeting(message) {
    $("#greetings").append("<tr><td>" + message + "</td></tr>");
}

$(function () {
    $("form").on('submit', function (e) {
        e.preventDefault();
    });
    $( "#connect" ).click(function() { connect(); });
    $( "#disconnect" ).click(function() { disconnect(); });
    $( "#send" ).click(function() { sendName(); });
});
```

7.5. Apêndice E: Controller para testes Polling server

PollingController.java

```
package app;

import java.io.IOException;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import utils.CSVUtil;
import utils.ProcessaFiboThread;

/**
 * Controller para testes de Short Polling.
 * Projeto de Graduacao - Engenharia de Software - Poli/UFRJ
 * @author Wallace de Souza Espindola
 */
@RestController
public class PollingController {

	private final String NOME_TESTE = "Polling";
	private final int MAX_TESTE = 50;
	private int contChamada = 1;
	private int numFibo = 0;
	private long tInicioChamadas = 0;
	private long tChamadaAtual = 0;
	private boolean fimTeste = false;
	private ProcessaFiboThread fiboThread = new ProcessaFiboThread();

	@RequestMapping("/polling")
	public String polling() throws IOException {

		tChamadaAtual = System.currentTimeMillis();

		if (fiboThread.isProcessoFinalizado() & !fimTeste) {

			CSVUtil.salvarCSV(NOME_TESTE, numFibo, contChamada, tInicioChamadas, tChamadaAtual,
fiboThread.getInicioProcesso(), fiboThread.getFimProcesso(), fiboThread.getResultado());

			numFibo++;
			if (numFibo > MAX_TESTE) {
				fimTeste = true;
			}
			if (!fimTeste) {
				reiniciar();
			} else {
				reset();
				System.out.println("##### Fim do teste. #####");
			}
		}

		if (!fimTeste) {
			CSVUtil.salvarCSV(NOME_TESTE, numFibo, contChamada, tInicioChamadas, tChamadaAtual,
fiboThread.getInicioProcesso(), fiboThread.getFimProcesso(), fiboThread.getResultado());
		}

		String resultado;
		if (!fimTeste) {
			resultado = NOME_TESTE + " - fibo: " + numFibo + " - chamada: " + contChamada++ + " -
resultFibo: " + fiboThread.getResultado();
		} else {
			resultado = "Fim do teste.";
		}
		System.out.println(resultado);
		return resultado;
	}

	private void reiniciar() {
		fiboThread.interrupt();
		fiboThread = new ProcessaFiboThread();
		contChamada = 1; // reinicia contagem
		tInicioChamadas = tChamadaAtual;
		fiboThread.setNumRodadas(numFibo);
		fiboThread.start();
	}

	private void reset() {
		fiboThread.interrupt();
		contChamada = 0;
		tInicioChamadas = 0;
		tChamadaAtual = 0;
		fiboThread.setNumRodadas(0);
	}
}
```

7.6. Apêndice F: Controller para testes Long Polling server

LongPollingController.java

```
package app;

import java.io.IOException;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import utils.CSVUtil;
import utils.ProcessaFiboThread;

/**
 * Controller para testes de Long Polling.
 * Projeto de Graduacao - Engenharia de Software - Poli/UFRJ
 * @author Wallace de Souza Espindola
 */
@RestController
public class LongPollingController {

	private final String NOME_TESTE = "Long-polling";
	private final int MAX_TESTE = 50;
	private int contChamada = 1;
	private int numFibo = 0;
	private long tInicioChamadas = 0;
	private long tChamadaAtual = 0;
	private boolean fimTeste = false;
	private ProcessaFiboThread fiboThread = new ProcessaFiboThread();

	@RequestMapping("/longpolling")
	public String longPolling() throws IOException {

		tChamadaAtual = System.currentTimeMillis();

		if (fiboThread.isProcessoFinalizado() & !fimTeste) {

			CSVUtil.salvarCSV(NOME_TESTE, numFibo, contChamada, tInicioChamadas, tChamadaAtual,
fiboThread.getInicioProcesso(), fiboThread.getFimProcesso(), fiboThread.getResultado());

			numFibo++;
			if (numFibo > MAX_TESTE) {
				fimTeste = true;
			}
			if (!fimTeste) {
				reiniciar();
			} else {
				reset();
				System.out.println("##### Fim do teste. #####");
			}
		}

		if (!fimTeste) {
			CSVUtil.salvarCSV(NOME_TESTE, numFibo, contChamada, tInicioChamadas, tChamadaAtual,
fiboThread.getInicioProcesso(), fiboThread.getFimProcesso(), fiboThread.getResultado());
		}

		String resultado;
		if (!fimTeste) {
			resultado = NOME_TESTE + " - fibo: " + numFibo + " - chamada: " + contChamada++ + " -
resultFibo: " + fiboThread.getResultado();
		} else {
			resultado = "Fim do teste.";
		}
		System.out.println(resultado);
		return resultado;
	}

	private void reiniciar() {
		fiboThread.interrupt();
		fiboThread = new ProcessaFiboThread();
		contChamada = 1; // reinicia contagem
		tInicioChamadas = tChamadaAtual;
		fiboThread.setNumRodadas(numFibo);
		fiboThread.start();
	}

	private void reset() {
		fiboThread.interrupt();
		contChamada = 0;
		tInicioChamadas = 0;
		tChamadaAtual = 0;
		fiboThread.setNumRodadas(0);
	}
}
```

7.7. Apêndice G: Controller para testes WebSocket server

WebSocketController.java

```
package websocket;

import java.math.BigDecimal;

import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

import utils.CSVUtil;
import utils.Fibonacci;

/**
 * Controller para testes de WebSocket.
 *
 * Projeto de Graduacao - Engenharia de Software - Poli/UFRJ
 *
 * @author Wallace de Souza Espindola
 *
 */
@Controller
public class WebSocketController {

	private final String NOME_TESTE = "Websocket";

	@MessageMapping("/hello")
	@SendTo("/topic/greetings")
	public ConteudoMensagem executar(HelloMessage message) throws Exception {

		int numFibo = Integer.parseInt(message.getName());
		int contChamada = 1;
		long tInicioChamadas = System.currentTimeMillis();
		long tChamadaAtual = tInicioChamadas;
		long inicioProcesso = tInicioChamadas;
		long fimProcesso = 0;
		long resultado = 0;

		resultado = Fibonacci.fibo(numFibo); // processo a ser executado

		tChamadaAtual = System.currentTimeMillis();
		fimProcesso = System.currentTimeMillis();

		BigDecimal mil = new BigDecimal(1000);
		mil = mil.setScale(2, BigDecimal.ROUND_HALF_UP);

		long duracaoProcessoEmMilis = fimProcesso - inicioProcesso;
		BigDecimal decimalDuracaoProcessoEmMilis = new
BigDecimal(duracaoProcessoEmMilis);
		decimalDuracaoProcessoEmMilis = decimalDuracaoProcessoEmMilis.setScale(2,
BigDecimal.ROUND_HALF_UP);
		BigDecimal decimalDuracaoProcessoEmSeg =
decimalDuracaoProcessoEmMilis.divide(mil, BigDecimal.ROUND_HALF_UP);

		CSVUtil.salvarCSV(NOME_TESTE, numFibo, contChamada, tInicioChamadas,
tChamadaAtual, inicioProcesso, fimProcesso, resultado);

		return new ConteudoMensagem("Fibo " + message.getName() + " executado em " +
decimalDuracaoProcessoEmSeg + " segundos! (resultado=" + resultado + ")");
	}

}
```

7.8. Apêndice H: Classe thread para testes Polling server

ProcessaFiboThread.java

```
package utils;

/**
 * Codigo fonte para executar o teste com Polling numa thread rodando fibonacci.
 *
 * Projeto de Graduacao - Engenharia de Software - Poli/UFRJ
 *
 * @author Wallace de Souza Espindola
 *
 */
public class ProcessaFiboThread extends Thread {

	private boolean processoFinalizado = true;
	private int numRodadas = 0;
	private long inicioProcesso = 0;
	private long fimProcesso = 0;
	private long resultado = 0;

	public void run() {
		System.out.println("ProcessaFiboThread iniciando - rodadas fibo: " +
this.numRodadas);
		inicioProcesso = System.currentTimeMillis();
		processoFinalizado = false;
		resultado = Fibonacci.fibo(this.numRodadas); // processo alvo com tempo de
execucao x
		processoFinalizado = true;
		fimProcesso = System.currentTimeMillis();
		System.out.println("ProcessaFiboThread fim processamento");
	}

	public void setNumRodadas(int numRodadas) {
		this.numRodadas = numRodadas;
	}

	public boolean isProcessoFinalizado() {
		return processoFinalizado;
	}

	public long getInicioProcesso() {
		return inicioProcesso;
	}

	public long getFimProcesso() {
		return fimProcesso;
	}

	public long getResultado() {
		return processoFinalizado ? resultado : 0;
	}
}
```

7.9. Apêndice I: Classe com Algoritmo Fibonacci de teste

ProcessaFiboThread.java

```
package utils;

import java.math.BigDecimal;
import java.util.Date;

/**
 * Codigo fonte para executar o teste com o algoritmo de Fibonacci.
 *
 * Projeto de Graduacao - Engenharia de Software - Poli/UFRJ
 *
 * @author Wallace de Souza Espindola
 *
 */
public class Fibonacci {

	public static void main(String[] args) {

		int duracaoFibo = 60;

		BigDecimal mil = new BigDecimal(1000);
		mil = mil.setScale(2, BigDecimal.ROUND_HALF_UP);

		// Teste do programa, imprime os X primeiros termos.
		for (int i = 1; i <= duracaoFibo; i++) {
			long inicio = System.currentTimeMillis();
			System.out.println("-------------------------------------------------
-");
			System.out.println("Inicio fibo de [" + i + "] [" + new Date() + "]");
			System.out.println("fibo(" + i + "):" + fibo(i));
			System.out.println("Fim fibo de [" + i + "] [" + new Date() + "]");
			long fim = System.currentTimeMillis();

			long duracaoMillis = fim - inicio;

			BigDecimal decimalDuracaoEmMilis = new BigDecimal(duracaoMillis);
			decimalDuracaoEmMilis = decimalDuracaoEmMilis.setScale(2,
BigDecimal.ROUND_HALF_UP);
			BigDecimal decimalDuracaoEmSeg = decimalDuracaoEmMilis.divide(mil,
BigDecimal.ROUND_HALF_UP);

			System.out.println("Duracao fibonacci de [" + i + "] em segundos: " +
decimalDuracaoEmSeg);
		}
	}

	public static long fibo(int n) {
		return (n < 2) ? n : fibo(n - 1) + fibo(n - 2);
	}

}
```

7.10. Apêndice J: Dados do teste de Polling

A tabela abaixo representa os valores obtidos nos testes utilizando a técnica Polling. Tais valores foram filtrados, eliminando desta todas as linhas referentes às chamadas anteriores à chamada final que gerou o resultado do processo.

Tabela 2: Valores da execução de testes utilizando Polling.

Num Fibo	Cont Chamada	Inicio Chamadas	Chamada Final	Duracao Chamadas (ms)	Duracao Chamadas (s)	Inicio Processo	Fim Processo	Duracao Processo (ms)	Duracao Processo (s)	Result Fibo
1	2	1520687872926	1520687874898	1972	1.97	1520687872940	1520687872941	1	0.00	1
2	2	1520687874898	1520687876894	1996	2.00	1520687875050	1520687875050	0	0.00	1
3	2	1520687876894	1520687878896	2002	2.00	1520687876898	1520687876898	0	0.00	2
4	2	1520687878896	1520687880895	1999	2.00	1520687878901	1520687878901	0	0.00	3
5	2	1520687880895	1520687882897	2002	2.00	1520687880899	1520687880899	0	0.00	5
6	2	1520687882897	1520687884895	1998	2.00	1520687882901	1520687882902	1	0.00	8
7	2	1520687884895	1520687886894	1999	2.00	1520687884899	1520687884899	0	0.00	13
8	2	1520687886894	1520687888894	2000	2.00	1520687886898	1520687886899	1	0.00	21
9	2	1520687888894	1520687890895	2001	2.00	1520687888897	1520687888898	1	0.00	34
10	2	1520687890895	1520687892895	2000	2.00	1520687890900	1520687890900	0	0.00	55
11	2	1520687892895	1520687894895	2000	2.00	1520687892900	1520687892900	0	0.00	89
12	2	1520687894895	1520687896897	2002	2.00	1520687894902	1520687894902	0	0.00	144
13	2	1520687896897	1520687898897	2000	2.00	1520687896902	1520687896902	0	0.00	233
14	2	1520687898897	1520687900897	2000	2.00	1520687898902	1520687898902	0	0.00	377
15	2	1520687900897	1520687902897	2000	2.00	1520687900901	1520687900901	0	0.00	610
16	2	1520687902897	1520687904895	1998	2.00	1520687902903	1520687902903	0	0.00	987
17	2	1520687904895	1520687906895	2000	2.00	1520687904901	1520687904901	0	0.00	1597
18	2	1520687906895	1520687908899	2004	2.00	1520687906900	1520687906900	0	0.00	2584
19	2	1520687908899	1520687910896	1997	2.00	1520687908903	1520687908904	1	0.00	4181
20	2	1520687910896	1520687912896	2000	2.00	1520687910900	1520687910900	0	0.00	6765
21	2	1520687912896	1520687914900	2004	2.00	1520687912900	1520687912900	0	0.00	10946
22	2	1520687914900	1520687916898	1998	2.00	1520687914905	1520687914905	0	0.00	17711
23	2	1520687916898	1520687918896	1998	2.00	1520687916902	1520687916902	0	0.00	28657
24	2	1520687918896	1520687920896	2000	2.00	1520687918899	1520687918899	0	0.00	46368
25	2	1520687920896	1520687922894	1998	2.00	1520687920901	1520687920902	1	0.00	75025
26	2	1520687922894	1520687924896	2002	2.00	1520687922897	1520687922899	2	0.00	121393
27	2	1520687924896	1520687926897	2001	2.00	1520687924900	1520687924903	3	0.00	196418
28	2	1520687926897	1520687928897	2000	2.00	1520687926906	1520687926910	4	0.00	317811
29	2	1520687928897	1520687930897	2000	2.00	1520687928904	1520687928914	10	0.01	514229
30	2	1520687930897	1520687932898	2001	2.00	1520687930906	1520687930919	13	0.01	832040
31	2	1520687932898	1520687934894	1996	2.00	1520687932905	1520687932922	17	0.02	1346269
32	2	1520687934894	1520687936894	2000	2.00	1520687934898	1520687934926	28	0.03	2178309
33	2	1520687936894	1520687938895	2001	2.00	1520687936899	1520687936936	37	0.04	3524578
34	2	1520687938895	1520687940895	2000	2.00	1520687938905	1520687938973	68	0.07	5702887
35	2	1520687940895	1520687942894	1999	2.00	1520687940901	1520687940989	88	0.09	9227465
36	2	1520687942894	1520687944896	2002	2.00	1520687942907	1520687943037	130	0.13	14930352
37	2	1520687944896	1520687946894	1998	2.00	1520687944900	1520687945090	190	0.19	24157817
38	2	1520687946894	1520687948894	2000	2.00	1520687946898	1520687947190	292	0.29	39088169
39	2	1520687948894	1520687950895	2001	2.00	1520687948898	1520687949361	463	0.46	63245986
40	2	1520687950895	1520687952894	1999	2.00	1520687950906	1520687951642	736	0.74	102334155
41	2	1520687952894	1520687954895	2001	2.00	1520687952897	1520687954089	1192	1.19	165580141
42	2	1520687954895	1520687956895	2000	2.00	1520687954899	1520687956807	1908	1.91	267914296
43	3	1520687956895	1520687960895	4000	4.00	1520687956903	1520687960011	3108	3.11	433494437
44	4	1520687960895	1520687966894	5999	6.00	1520687960898	1520687965909	5011	5.01	701408733
45	6	1520687966894	1520687976895	10001	10.00	1520687966897	1520687975084	8187	8.19	1134903170
46	8	1520687976895	1520687990895	14000	14.00	1520687976899	1520687990043	13144	13.14	1836311903
47	12	1520687990895	1520688013417	22522	22.52	1520687990898	1520688012604	21706	21.71	2971215073
48	19	1520688013417	1520688049403	35986	35.99	1520688013417	1520688047718	34301	34.30	4807526976
49	29	1520688049403	1520688104895	55492	55.49	1520688049488	1520688104794	55306	55.31	7778742049
50	47	1520688104895	1520688196895	92000	92.00	1520688104899	1520688194958	90059	90.06	12586269025

7.11. Apêndice K: Dados do teste de Long Polling

A tabela abaixo representa os valores obtidos nos testes utilizando a técnica Long Polling. Tais valores foram filtrados, eliminando desta todas as linhas referentes às chamadas anteriores à chamada final que gerou o resultado do processo.

Tabela 3: Valores da execução de testes utilizando Long Polling.

Num Fibo	Cont Chamada	Inicio Chamadas	Chamada Final	Duracao Chamadas (ms)	Duracao Chamadas (s)	Inicio Processo	Fim Processo	Duracao Processo (ms)	Duracao Processo (s)	Result Fibo
1	2	1520691736144	1520691766126	29982	29.98	1520691736160	1520691736160	0	0.00	1
2	2	1520691766126	1520691796120	29994	29.99	1520691766140	1520691766140	0	0.00	1
3	2	1520691796120	1520691826120	30000	30.00	1520691796128	1520691796128	0	0.00	2
4	2	1520691826120	1520691856118	29998	30.00	1520691826129	1520691826129	0	0.00	3
5	2	1520691856118	1520691886121	30003	30.00	1520691856125	1520691856125	0	0.00	5
6	2	1520691886121	1520691916417	30296	30.30	1520691886133	1520691886133	0	0.00	8
7	2	1520691916417	1520691946121	29704	29.70	1520691916424	1520691916424	0	0.00	13
8	2	1520691946121	1520691976120	29999	30.00	1520691946130	1520691946130	0	0.00	21
9	2	1520691976120	1520692006118	29998	30.00	1520691976125	1520691976125	0	0.00	34
10	2	1520692006118	1520692036121	30003	30.00	1520692006125	1520692006125	0	0.00	55
11	2	1520692036121	1520692066119	29998	30.00	1520692036128	1520692036128	0	0.00	89
12	2	1520692066119	1520692096121	30002	30.00	1520692066125	1520692066125	0	0.00	144
13	2	1520692096121	1520692126124	30003	30.00	1520692096129	1520692096129	0	0.00	233
14	2	1520692126124	1520692156119	29995	30.00	1520692126133	1520692126133	0	0.00	377
15	2	1520692156119	1520692186400	30281	30.28	1520692156124	1520692156124	0	0.00	610
16	2	1520692186400	1520692216118	29718	29.72	1520692186407	1520692186407	0	0.00	987
17	2	1520692216118	1520692246121	30003	30.00	1520692216127	1520692216127	0	0.00	1597
18	2	1520692246121	1520692276122	30001	30.00	1520692246126	1520692246126	0	0.00	2584
19	2	1520692276122	1520692306119	29997	30.00	1520692276131	1520692276131	0	0.00	4181
20	2	1520692306119	1520692336121	30002	30.00	1520692306126	1520692306126	0	0.00	6765
21	2	1520692336121	1520692366119	29998	30.00	1520692336128	1520692336128	0	0.00	10946
22	2	1520692366119	1520692396119	30000	30.00	1520692366125	1520692366125	0	0.00	17711
23	2	1520692396119	1520692426116	29997	30.00	1520692396123	1520692396123	0	0.00	28657
24	2	1520692426116	1520692456118	30002	30.00	1520692426124	1520692426125	1	0.00	46368
25	2	1520692456118	1520692486118	30000	30.00	1520692456123	1520692456125	2	0.00	75025
26	2	1520692486118	1520692516120	30002	30.00	1520692486126	1520692486127	1	0.00	121393
27	2	1520692516120	1520692546122	30002	30.00	1520692516125	1520692516126	1	0.00	196418
28	2	1520692546122	1520692576120	29998	30.00	1520692546129	1520692546134	5	0.01	317811
29	2	1520692576120	1520692606119	29999	30.00	1520692576126	1520692576139	13	0.01	514229
30	2	1520692606119	1520692636413	30294	30.29	1520692606127	1520692606141	14	0.01	832040
31	2	1520692636413	1520692666406	29993	29.99	1520692636419	1520692636435	16	0.02	1346269
32	2	1520692666406	1520692696406	30000	30.00	1520692666409	1520692666426	17	0.02	2178309
33	2	1520692696406	1520692726405	29999	30.00	1520692696411	1520692696429	18	0.02	3524578
34	2	1520692726405	1520692756412	30007	30.01	1520692726414	1520692726449	35	0.04	5702887
35	2	1520692756412	1520692786404	29992	29.99	1520692756417	1520692756488	71	0.07	9227465
36	2	1520692786404	1520692816411	30007	30.01	1520692786413	1520692786529	116	0.12	14930352
37	2	1520692816411	1520692846403	29992	29.99	1520692816416	1520692816590	174	0.17	24157817
38	2	1520692846403	1520692876407	30004	30.00	1520692846408	1520692846674	266	0.27	39088169
39	2	1520692876407	1520692906415	30008	30.01	1520692876414	1520692876852	438	0.44	63245986
40	2	1520692906415	1520692936406	29991	29.99	1520692906422	1520692907148	726	0.73	102334155
41	2	1520692936406	1520692966416	30010	30.01	1520692936409	1520692937590	1181	1.18	165580141
42	2	1520692966416	1520692996406	29990	29.99	1520692966421	1520692968331	1910	1.91	267914296
43	2	1520692996406	1520693026406	30000	30.00	1520692996411	1520692999499	3088	3.09	433494437
44	2	1520693026406	1520693056406	30000	30.00	1520693026412	1520693031442	5030	5.03	701408733
45	2	1520693056406	1520693086408	30002	30.00	1520693056410	1520693064513	8103	8.10	1134903170
46	2	1520693086408	1520693116407	29999	30.00	1520693086412	1520693099438	13026	13.03	1836311903
47	2	1520693116407	1520693146407	30000	30.00	1520693116411	1520693137749	21338	21.34	2971215073
48	3	1520693146407	1520693206404	59997	60.00	1520693146410	1520693180807	34397	34.40	4807526976
49	3	1520693206404	1520693266405	60001	60.00	1520693206408	1520693262259	55851	55.85	7778742049
50	5	1520693266405	1520693386415	120010	120.01	1520693266412	1520693357673	91261	91.26	12586269025

7.12. Apêndice L: Dados do teste de WebSocket

A tabela abaixo representa os valores obtidos nos testes utilizando a técnica WebSocket. Tais valores foram filtrados, eliminando desta todas as linhas referentes às chamadas anteriores à chamada final que gerou o resultado do processo.

Tabela 4: Valores da execução de testes utilizando WebSocket.

Num Fibo	Cont Chamada	Inicio Chamadas	Chamada Final	Duração Chamadas (ms)	Duração Chamadas (s)	Inicio Processo	Fim Processo	Duração Processo (ms)	Duração Processo (s)	Result. Fibo
1	1	1520701635917	1520701635917	0	0.00	1520701635917	1520701635917	0	0.00	1
2	1	1520701638337	1520701638337	0	0.00	1520701638337	1520701638337	0	0.00	1
3	1	1520701642297	1520701642297	0	0.00	1520701642297	1520701642297	0	0.00	2
4	1	1520701645431	1520701645431	0	0.00	1520701645431	1520701645431	0	0.00	3
5	1	1520701648611	1520701648611	0	0.00	1520701648611	1520701648611	0	0.00	5
6	1	1520701651379	1520701651379	0	0.00	1520701651379	1520701651379	0	0.00	8
7	1	1520701654296	1520701654296	0	0.00	1520701654296	1520701654296	0	0.00	13
8	1	1520701656930	1520701656930	0	0.00	1520701656930	1520701656930	0	0.00	21
9	1	1520701714679	1520701714679	0	0.00	1520701714679	1520701714679	0	0.00	34
10	1	1520701717911	1520701717911	0	0.00	1520701717911	1520701717911	0	0.00	55
11	1	1520701722574	1520701722574	0	0.00	1520701722574	1520701722574	0	0.00	89
12	1	1520701728460	1520701728460	0	0.00	1520701728460	1520701728460	0	0.00	144
13	1	1520701738685	1520701738685	0	0.00	1520701738685	1520701738685	0	0.00	233
14	1	1520701745398	1520701745398	0	0.00	1520701745398	1520701745398	0	0.00	377
15	1	1520701749222	1520701749222	0	0.00	1520701749222	1520701749222	0	0.00	610
16	1	1520701752550	1520701752550	0	0.00	1520701752550	1520701752550	0	0.00	987
17	1	1520701760976	1520701760976	0	0.00	1520701760976	1520701760976	0	0.00	1597
18	1	1520701764428	1520701764428	0	0.00	1520701764428	1520701764428	0	0.00	2584
19	1	1520701768098	1520701768098	0	0.00	1520701768098	1520701768098	0	0.00	4181
20	1	1520701770921	1520701770921	0	0.00	1520701770921	1520701770921	0	0.00	6765
21	1	1520701773381	1520701773382	1	0.00	1520701773381	1520701773382	1	0.00	10946
22	1	1520701775350	1520701775350	0	0.00	1520701775350	1520701775350	0	0.00	17711
23	1	1520701777308	1520701777308	0	0.00	1520701777308	1520701777308	0	0.00	28657
24	1	1520701779304	1520701779304	0	0.00	1520701779304	1520701779304	0	0.00	46368
25	1	1520701781146	1520701781146	0	0.00	1520701781146	1520701781146	0	0.00	75025
26	1	1520701809628	1520701809629	1	0.00	1520701809628	1520701809629	1	0.00	121393
27	1	1520701841296	1520701841298	2	0.00	1520701841296	1520701841298	2	0.00	196418
28	1	1520701845260	1520701845262	2	0.00	1520701845260	1520701845262	2	0.00	317811
29	1	1520701861269	1520701861272	3	0.00	1520701861269	1520701861272	3	0.00	514229
30	1	1520701872738	1520701872744	6	0.01	1520701872738	1520701872744	6	0.01	832040
31	1	1520701876080	1520701876090	10	0.01	1520701876080	1520701876090	10	0.01	1346269
32	1	1520701890464	1520701890482	18	0.02	1520701890464	1520701890482	18	0.02	2178309
33	1	1520701893717	1520701893746	29	0.03	1520701893717	1520701893746	29	0.03	3524578
34	1	1520701896539	1520701896584	45	0.05	1520701896539	1520701896584	45	0.05	5702887
35	1	1520701899269	1520701899342	73	0.07	1520701899269	1520701899342	73	0.07	9227465
36	1	1520701901511	1520701901620	109	0.11	1520701901511	1520701901620	109	0.11	14930352
37	1	1520701903497	1520701903672	175	0.18	1520701903497	1520701903672	175	0.18	24157817
38	1	1520701906772	1520701907052	280	0.28	1520701906772	1520701907052	280	0.28	39088169
39	1	1520701909169	1520701909621	452	0.45	1520701909169	1520701909621	452	0.45	63245986
40	1	1520701915114	1520701915843	729	0.73	1520701915114	1520701915843	729	0.73	102334155
41	1	1520701918805	1520701919973	1168	1.17	1520701918805	1520701919973	1168	1.17	165580141
42	1	1520701922579	1520701924471	1892	1.89	1520701922579	1520701924471	1892	1.89	267914296
43	1	1520701926840	1520701929895	3055	3.06	1520701926840	1520701929895	3055	3.06	433494437
44	1	1520701943023	1520701947964	4941	4.94	1520701943023	1520701947964	4941	4.94	701408733
45	1	1520701951686	1520701959675	7989	7.99	1520701951686	1520701959675	7989	7.99	1134903170
46	1	1520701963148	1520701976071	12923	12.92	1520701963148	1520701976071	12923	12.92	1836311903
47	1	1520701979708	1520702000710	21002	21.00	1520701979708	1520702000710	21002	21.00	2971215073
48	1	1520702012346	1520702046197	33851	33.85	1520702012346	1520702046197	33851	33.85	4807526976
49	1	1520702061669	1520702116657	54988	54.99	1520702061669	1520702116657	54988	54.99	7778742049
50	1	1520702134081	1520702222755	88674	88.67	1520702134081	1520702222755	88674	88.67	12586269025

7.13. Apêndice M: Dados do gráfico do teste Polling

A tabela abaixo representa os uma visão dos valores filtrados para utilização no gráfico que demonstra a execução dos testes de Polling. Foi ajustada a duração mínima do processo para 0.01 para facilitar a visualização no gráfico.

Tabela 5: Valores da execução dos testes de Polling, para uso em gráfico.

Num Rodadas Fibo	Contagem de Chamada	Duração Chamadas (s)	Duração Processo (s)
1	2	1.97	0.01
2	2	2.00	0.01
3	2	2.00	0.01
4	2	2.00	0.01
5	2	2.00	0.01
6	2	2.00	0.01
7	2	2.00	0.01
8	2	2.00	0.01
9	2	2.00	0.01
10	2	2.00	0.01
11	2	2.00	0.01
12	2	2.00	0.01
13	2	2.00	0.01
14	2	2.00	0.01
15	2	2.00	0.01
16	2	2.00	0.01
17	2	2.00	0.01
18	2	2.00	0.01
19	2	2.00	0.01
20	2	2.00	0.01
21	2	2.00	0.01
22	2	2.00	0.01
23	2	2.00	0.01
24	2	2.00	0.01
25	2	2.00	0.01
26	2	2.00	0.01
27	2	2.00	0.01
28	2	2.00	0.01
29	2	2.00	0.01
30	2	2.00	0.01
31	2	2.00	0.02
32	2	2.00	0.03
33	2	2.00	0.04
34	2	2.00	0.07
35	2	2.00	0.09
36	2	2.00	0.13
37	2	2.00	0.19
38	2	2.00	0.29
39	2	2.00	0.46
40	2	2.00	0.74
41	2	2.00	1.19
42	2	2.00	1.91
43	3	4.00	3.11
44	4	6.00	5.01
45	6	10.00	8.19
46	8	14.00	13.14
47	12	22.52	21.71
48	19	35.99	34.30
49	29	55.49	55.31
50	47	92.00	90.06

7.14. Apêndice N: Dados do gráfico do teste Long Polling

A tabela abaixo representa os uma visão dos valores filtrados para utilização no gráfico que demonstra a execução dos testes de Long Polling. Foi ajustada a duração mínima do processo para 0.01 para facilitar a visualização no gráfico.

Tabela 6: Valores da execução dos testes de Long Polling, para uso em gráfico.

Num Rodadas Fibo	Contagem de Chamada	Duração Chamadas (s)	Duração Processo (s)
1	2	29.98	0.01
2	2	29.99	0.01
3	2	30.00	0.01
4	2	30.00	0.01
5	2	30.00	0.01
6	2	30.30	0.01
7	2	29.70	0.01
8	2	30.00	0.01
9	2	30.00	0.01
10	2	30.00	0.01
11	2	30.00	0.01
12	2	30.00	0.01
13	2	30.00	0.01
14	2	30.00	0.01
15	2	30.28	0.01
16	2	29.72	0.01
17	2	30.00	0.01
18	2	30.00	0.01
19	2	30.00	0.01
20	2	30.00	0.01
21	2	30.00	0.01
22	2	30.00	0.01
23	2	30.00	0.01
24	2	30.00	0.01
25	2	30.00	0.01
26	2	30.00	0.01
27	2	30.00	0.01
28	2	30.00	0.01
29	2	30.00	0.01
30	2	30.29	0.01
31	2	29.99	0.02
32	2	30.00	0.02
33	2	30.00	0.02
34	2	30.01	0.04
35	2	29.99	0.07
36	2	30.01	0.12
37	2	29.99	0.17
38	2	30.00	0.27
39	2	30.01	0.44
40	2	29.99	0.73
41	2	30.01	1.18
42	2	29.99	1.91
43	2	30.00	3.09
44	2	30.00	5.03
45	2	30.00	8.10
46	2	30.00	13.03
47	2	30.00	21.34
48	3	60.00	34.40
49	3	60.00	55.85
50	5	120.01	91.26

7.15. Apêndice O: Dados do gráfico do teste WebSocket

A tabela abaixo representa os uma visão dos valores filtrados para utilização no gráfico que demonstra a execução dos testes de WebSocket. Foi ajustada a duração mínima do processo para 0.01 para facilitar a visualização no gráfico.

Tabela 7: Valores da execução dos testes de WebSocket, para uso em gráfico.

Num Rodadas Fibo	Contagem de Chamada	Duração Chamadas (s)	Duração Processo (s)
1	1	0.00	0.01
2	1	0.00	0.01
3	1	0.00	0.01
4	1	0.00	0.01
5	1	0.00	0.01
6	1	0.00	0.01
7	1	0.00	0.01
8	1	0.00	0.01
9	1	0.00	0.01
10	1	0.00	0.01
11	1	0.00	0.01
12	1	0.00	0.01
13	1	0.00	0.01
14	1	0.00	0.01
15	1	0.00	0.01
16	1	0.00	0.01
17	1	0.00	0.01
18	1	0.00	0.01
19	1	0.00	0.01
20	1	0.00	0.01
21	1	0.00	0.01
22	1	0.00	0.01
23	1	0.00	0.01
24	1	0.00	0.01
25	1	0.00	0.01
26	1	0.00	0.01
27	1	0.00	0.01
28	1	0.00	0.01
29	1	0.00	0.01
30	1	0.01	0.01
31	1	0.01	0.01
32	1	0.02	0.02
33	1	0.03	0.03
34	1	0.05	0.05
35	1	0.07	0.07
36	1	0.11	0.11
37	1	0.18	0.18
38	1	0.28	0.28
39	1	0.45	0.45
40	1	0.73	0.73
41	1	1.17	1.17
42	1	1.89	1.89
43	1	3.06	3.06
44	1	4.94	4.94
45	1	7.99	7.99
46	1	12.92	12.92
47	1	21.00	21.00
48	1	33.85	33.85
49	1	54.99	54.99
50	1	88.67	88.67

7.16. Apêndice P: Uso de CPU e Memória durante os testes

Durante a execução dos testes a memória permaneceu constante em 5.0Gb de um total de 8.0Gb (um tanto alta, porém explicada, por haver alguns programas em execução no computador, como por exemplo o Google Chrome), e o uso de CPU variou, mas ficou em média por volta dos 30%.

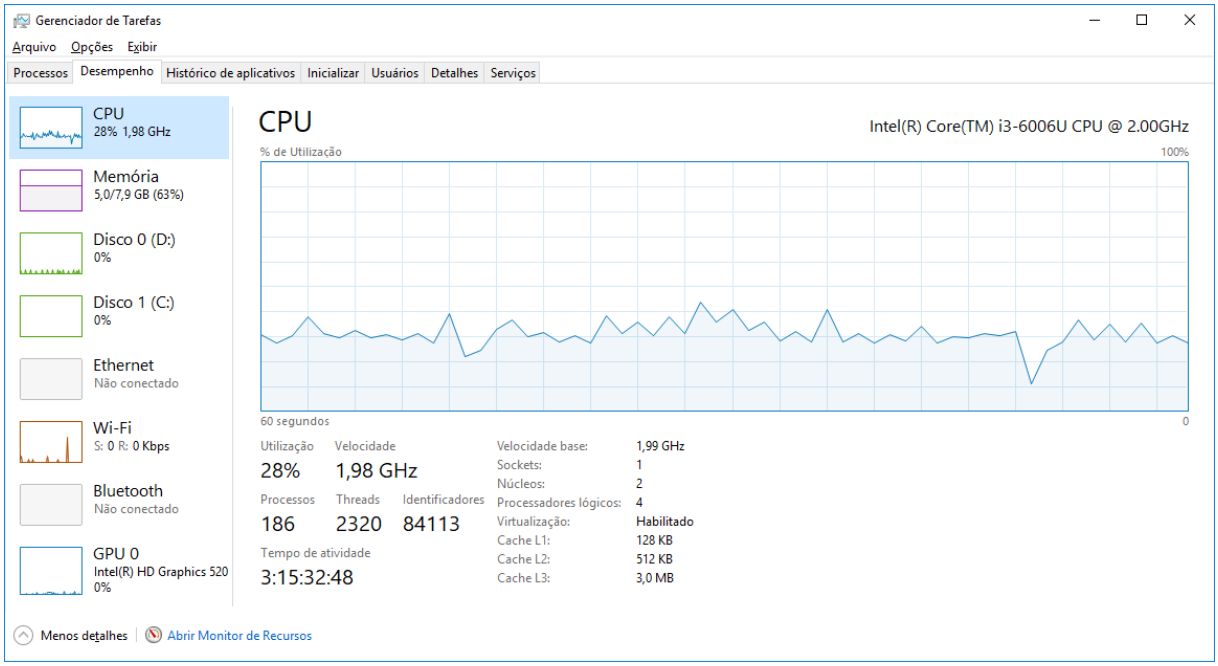


Figura 11: Uso de CPU no servidor com processo Fibonacci em execução.

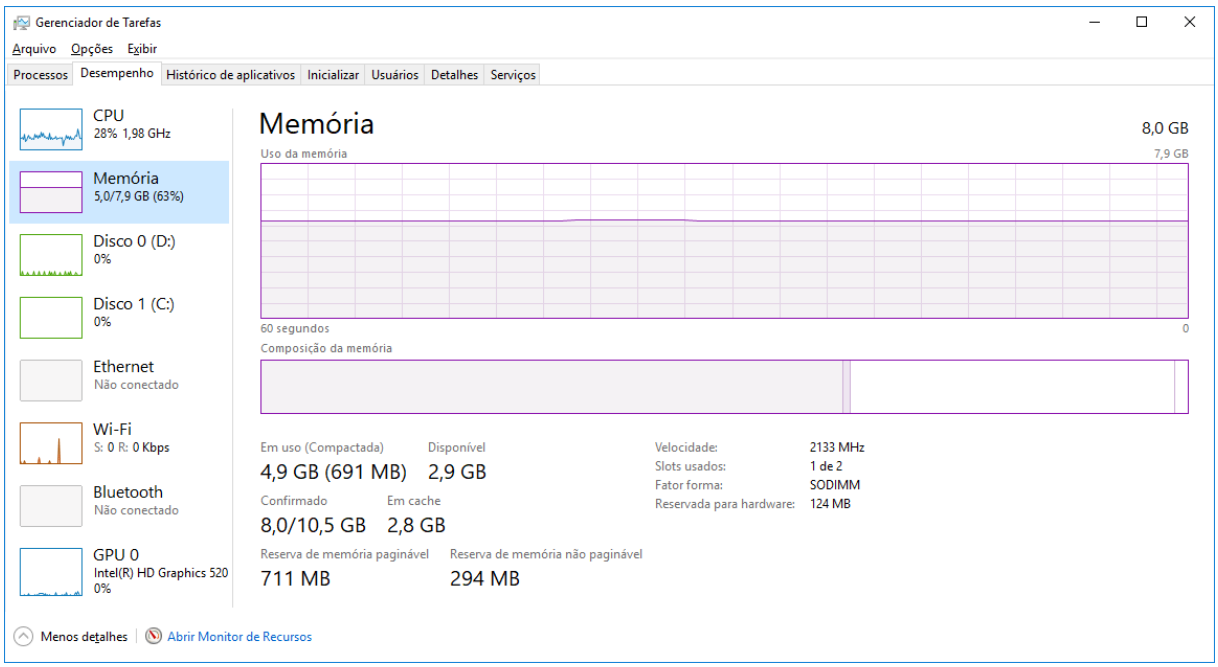


Figura 12: Uso de Memória no servidor com processo Fibonacci em execução.

7.17. Apêndice Q: Intervalo de Confiança

O cálculo do valor médio de um determinado parâmetro por si só não define exatamente a característica do mesmo.

Um intervalo de confiança relativo à média expressa a ideia de que temos um determinado nível de confiança de que a média se encontra naquele intervalo. Além da média, o intervalo de confiança delimita a margem de precisão existente em um determinado conjunto de amostras. Com isso, pode-se ter uma melhor e mais correta ideia do comportamento média de uma medida estatística.

Por exemplo, um intervalo de confiança de 90% indica que a média da população de amostras, obtidas em uma posterior realização daquele mesmo teste, estará entre os limites superior e inferior do intervalo.

O cálculo dos intervalos de confiança utilizados nas amostras dos testes presentes nesse projeto se baseia na seguinte fórmula [10]:

$$Ic = \left(\bar{x} - \left[t_{(1-\alpha/2\,;\,n-1)} \times \frac{s}{\sqrt{n}} \right] \;,\; \bar{x} + \left[t_{(1-\alpha/2\,;\,n-1)} \times \frac{s}{\sqrt{n}} \right] \right)$$

Onde:

$\bar{x}$ – média das amostras;

n – quantidade de amostras;

s – desvio padrão;

α – coeficiente de confiança (se o intervalo de confiança é de 90%, $\alpha = 0{,}1$);

$t_{(1-\alpha/2;n-1)}$ – valor tabelado dependente de α e n.