Proposal to Use of the Websocket Protocol for Web Device Control

Adriano de Oliveira Maia
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
R. Pedro Bezerra de Menezes, n° 387, Manoel Costa Morais.
Jaguaribe, Ceará 63475-000
adhryan.olivirmayer@gmail.com

Danilo Avilar Silva
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
R. Pedro Bezerra de Menezes, n° 387, Manoel Costa Morais.
Jaguaribe, Ceará 63475-000
daniloavilar@gmail.com

ABSTRACT

The Websocket protocol enables a full-duplex communication, besides it simplifies an exchange of data and reduces network overload. This paper proposes the use of the Websocket protocol in control and service devices through web within real-time requirements. Through tests made in a virtual environment and another one in embedded experiment, It is possible to validate an initial proposal of implementation the Websocket protocol. From the analysis of the results obtained, it can be seen the use of the proposal in question provides a considerable reduction in the quantity of requests and transferred data in relation to traditional approach to sending data in HTTP-based communications. Consequently it seems to be a very promising technique for this type of application.

CCS CONCEPTS

• **Networks** → **Network protocols**; **Transport protocols**; • **Computer systems organization** → *Real-time system specification*;

KEYWORDS

Websocket, HTTP, sistema embarcado

ACM Reference format:
Adriano de Oliveira Maia and Danilo Avilar Silva. 2017. Proposal to Use of the Websocket Protocol for Web Device Control. In *Proceedings of WebMedia '17, Gramado, Brasil, 17–20 de outubro de 2017,* 8 pages.
https://doi.org/10.1145/3126858.3126887

WebMedia '17, 17–20 de outubro de 2017, Gramado, Brasil

ACM ISBN 978-1-4503-5096-9/17/10. . . $15.00
https://doi.org/10.1145/3126858.3126887

1 INTRODUÇÃO

Dispositivos conectados a redes e controlados via *Web* como televisores, geladeiras, ou parte integrante de automação residencial e cidades inteligentes, vêm crescendo com a evolução do conceito denominado de Internet das Coisas (*Internet of Things*, IoT) [1]. Este conceito, associado a modelos de hardware e software, serve de suporte para a ascensão do futuro da Internet [14].

Ainda neste contexto, propostas de desenvolvimento de veículos inteligentes junto às novas técnicas de controle remoto de veículos, ganham cada vez mais destaque como exemplo, observa-se a parceria entre as empresas Toyota e Microsoft para o desenvolvimento de um veículo não tripulado inteligente [24]. Já as empresas Hyundai e Cisco propõem um novo conceito de veículo não tripulado com controle via Internet [16]. De acordo com [31], veículos terrestres controlados à distância têm sido explorados para vigilância, desarmamento de bombas, ataques e acesso a lugares que ofereçam riscos para a saúde humana.

Em todas essas aplicações é necessária uma conexão remota que utilize protocolos de uma arquitetura de rede para troca de dados de controle do veículo, exigindo assim, um protocolo que ofereça baixa latência e pouco consumo da rede, pois os comandos acontecem em tempo real. Sobre a classe de sistema de tempo real adotado neste trabalho, considera-se um sistema crítico de tempo real com enfoque no tempo lógico. Trata-se de um grupo de sistemas que trabalha com restrições temporais caracterizadas por considerar que qualquer prazo não atendido implica em falha no sistema, isto é, apresenta restrições críticas. Considera-se ainda que, a partir de relações de precedência entre eventos é possível admitir ordens causais sobre um conjunto de eventos e o tempo físico. Dentre tais aplicações destacam-se: sistemas de supervisão, sistemas embarcados em robôs e automóveis [9]. Esses sistemas trabalham com um tempo de resposta na ordem de milésimos de segundos, admitindo-se atrasos de tempo físico pouco perceptíveis, normalmente, até 300ms [4].

Em [31], foi proposto utilizar o protocolo UDP para controlar remotamente um veículo terrestre não tripulado (VTNT) por ser mais rápido na comunicação entre o computador e o VTNT, em comparativo ao protocolo TCP. Entretanto o UDP não oferece a confiabilidade de entrega de dados do TCP, o que é de suma importância em aplicações de tempo real em uma rede de computadores.

Tendo em vista estas necessidades em aplicações de controle em tempo real, iniciou-se uma pesquisa sobre protocolos que cumpram com tais requisitos, e assim, uma possível solução é a utilização do protocolo Websocket. O Websocket utiliza um pedido HTTP atualizado e envia dados de uma forma baseada em mensagem semelhante ao UDP com toda a confiabilidade do TCP [19].

"Os Websockets surgiram para prover serviço com retardo reduzido, o qual o HTTP não oferece apoio atualmente" [13]. Com essa capacidade, a utilização do Websocket como protocolo de transmissão dos dados de controle do veículo não tripulado proporciona uma resposta rápida aos comandos, assim como a garantia de entrega de dados. Além disso, o protocolo possibilita a utilização via *Web* para controle do veículo.

Diante do exposto, este trabalho tem como objetivo principal validar uma proposta inicial de utilização do protocolo Websocket em aplicações de controle e serviço de dispositivos via *Web*. Adicionalmente, busca-se identificar as principais características do Websocket em comparativo à abordagem tradicional de troca de dados em comunicações baseadas em HTTP, o *Polling*. O restante do artigo está organizado em 4 seções a saber: Na Seção 2 é apresentada uma revisão bibliográfica acerca dos protocolos e linguagens de programação que podem ser utilizados para tal finalidade, atentando-se as particularidades de cada um. A Seção 3 descreve a metodologia aplicada dividida em simulações em ambiente virtual e experimento embarcado. Já na Seção 4 são apresentadas análises dos resultados obtidos. Por fim, na Seção 5, são apresentadas as conclusões alcançadas a partir dos testes realizados e trabalhos futuros.

2 ESTADO DA ARTE

A Internet é uma rede caracterizada por conectar milhares de dispositivos pelo mundo. Antigamente, essa rede se restringia a computadores pessoais de mesa (PCs), servidores de armazenamento e transmissão de informações como e-mail e páginas *Web*. No entanto, com o crescimento e evolução dos dispositivos finais, essa rede é composta por *smartphones*, televisores e câmeras com acesso à rede, sistemas de segurança, automóveis e outros dispositivos que realizam tráfego de dados [17]. É através de enlaces de comunicação e comutadores, tais como *switches* que trabalham com os pacotes, que a conexão dos dispositivos finais é feita. Toda essa rede é controlada e normalizada por protocolos de rede [3].

Um protocolo determina os padrões e diretrizes em que as mensagens são trocadas entre as entidades que se comunicam por meio de uma rede, tais como a Internet e as redes de computadores. Eles conduzem da melhor forma a transmissão, recebimento e outros eventos que podem ocorrer nessa troca de mensagens [17].

Os protocolos são diversificados em relação as diferentes necessidades de comunicação. Existem protocolos de implementação simples para uma comunicação direta, assim como há protocolos de implementação complexa para aplicações mais específicas, o que torna fundamental na área de redes de computadores o estudo de tais, para saber qual protocolo utilizar de acordo com a necessidade da comunicação [3].

2.1 HTTP

O Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*, HTTP), é o gerenciador central da *Web*, e se divide em dois agentes: um servidor e um cliente. Ambos são executados em sistemas finais diferentes comunicando-se entre si através de mensagens HTTP. O HTTP é normatizado pela [RFC 1945] e [RFC 2616] [28].

A estrutura e o modo de trocas de mensagens entre cliente e servidor HTTP são definidas pelo próprio protocolo e acontecem de dois modos: persistente e não persistente. No modo não persistente, o HTTP trabalha com requisição e resposta entre cliente/servidor através de conexões TCP distintas. Já o modo persistente, caracteriza-se pela ocorrência de uma única conexão TCP (modo padrão) [17].

Nas duas há sempre uma latência ao decorrer da troca de requisição/resposta, sendo que na não persistente ela é bem maior, pois para cada conexão, são alocados *buffers* do TCP e conservadas variáveis tanto no cliente como no servidor. Tal ação, causa uma sobrecarga no servidor *Web* quando existem muitas requisições simultâneas. Já na conexão persistente, os arquivos são enviados por uma única conexão TCP persistente, deixando o canal aberto por um tempo. Entretanto, a cada atualização do arquivo terá que ser feita uma nova requisição [17].

Esses modos estão presentes nas duas técnicas normalmente utilizadas para proporcionar serviço em tempo real no HTTP, a saber: *Polling* e *Long Polling*. A primeira consiste no modo não persistente e a segunda, persistente.

Em um trabalho de análise da utilização de Websockets em sistemas computacionais, [29] fazem testes comparativos entre versões do HTTP e o Websocket sobre tráfego de pacotes, tempo de resposta, largura de banda e troca de mensagens síncronas e assíncronas. Em praticamente todos os resultados o HTTP mostra-se desvantajoso, por ter um consumo maior de banda, maior tráfego de pacotes e tempo de resposta muito superior ao do Websocket.

2.2 HTML e HTML5

A Linguagem de Marcação de Hipertexto (*Hypertext Markup Language*, HTML), é uma linguagem para publicação de diversos conteúdos na *Web* de forma fácil, organizada e independente de plataforma [8].

De início o HTML cumpriu com seu papel, que tinha como principal intuito o compartilhamento de documentos na rede. Mas com a evolução da tecnologia *Web*, que hoje tem necessidade de maior interatividade de seus usuários, o HTML vem se renovando cada vez mais para atender a necessidade de desenvolvimento de aplicações em uma gama diversificada de dispositivos, que vão desde laptops, *smartphones*, *tablets*, câmeras até eletrodomésticos e automóveis [30].

Com o intuito de suprir a ineficácia do HTML para a interação de diversos formatos de mídias, [5] propõem a utilização do HTML+TIME (*Timed Interactive Multimedia Extensions*) para sincronização de multimídia. O HTML+TIME objetiva integrar áudio, vídeo e imagem ou texto em uma mesma aplicação, mantendo a qualidade na transmissão. A sincronização é feita a partir do controle de tempo entre as mídias, atendendo ainda parte das necessidades de transmissão multimídia, no entanto, não oferece suporte para transmissões de tempo real.

O HTML5 tem como principal intuito ter seu desenvolvimento de aplicações *Web* de maneira mais natural e lógica, onde uma vez desenvolvido ou modificado, aquela aplicação se adéque em qualquer lugar [6]. Uma das áreas mais complexas em termos de implementação, devido a evolução de riqueza de recursos, é a conectividade, que com o HTML5 trouxe tecnologias como Websocket, *Server-Sent Events*, e *Cross-Document Messaging* [30].

O Websocket surge como uma solução otimizada para conexões em tempo real, suprindo uma deficiência do HTTP que trabalha com conexão *half-duplex*, ou seja, o tráfego parte em uma direção de cada vez, tornando-o ineficiente para conexões de tempo real.

Ainda neste contexto, [22] em seu estudo do HTML5 Websocket para uma comunicação multimídia, mostram que entre as extensões do HTML5 na API de desenvolvimento do *JavaScript*, onde se encontram aplicações ricas de recursos e funcionalidades, o Websocket em testes comparativos, é mais eficaz que o HTTP para aplicações como jogos online em tempo real e transmissão de recursos multimídia na *Web*.

Já [7] utilizam o HTML5 junto ao protocolo Websocket no desenvolvimento do Laboratório Virtual de Aprendizagem Eletrônica (LABVAE) para auxiliar no aprendizado dos estudantes dos cursos de engenharia elétrica, eletrônica e mecatrônica. Essa ferramenta auxilia os estudantes a terem acesso remoto aos equipamentos e ferramentas do laboratório, permitindo ainda analisarem resultados e visualizá-los através de uma webcam, tudo isso pela interface *Web* em tempo real.

2.3 JavaScript e Node.js

A linguagem *JavaScript*, é uma linguagem de scripts interpretada baseada em objetos e capaz de oferecer suporte à construções estruturadas. Originalmente, foi implementada como parte dos navegadores *Web* para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade de passar pelo servidor, realizando comunicação assíncrona [12].

É considerada a principal linguagem para programação cliente-servidor em navegadores *Web*. Como já mencionado, possui características como orientação a objetos baseada em protótipos, tipagem fraca e dinâmica e funções de primeira classe. Possui ainda, suporte à programação funcional e apresenta recursos como fechamentos e funções de alta ordem regularmente indisponíveis em linguagens como Java e C++ [26].

Node.js é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilitar o desenvolvimento de aplicações de rede rápidas e escaláveis. Utiliza um modelo de I/O direcionado a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com intensa troca de dados através de dispositivos distribuídos.

No contexto de IoT, [21] apresentam as etapas necessárias para desenvolver uma aplicação simples utilizando a plataforma Arduino e o framework Node.js. Já [27] demostram a criação de programas de rede de alto desempenho utilizando *JavaScript* e Node.js.

2.4 WebSocket

Websocket é uma nova tecnologia de conexão em tempo real que cria um canal *full-duplex* entre cliente e servidor. Trata-se de uma evolução do HTTP tendo como base o protocolo TCP e sua API vem sendo padronizada pela W3C (*World Wide Web Consortium*) e IETF (*Internet Engineering Task Force*) [10]. O Websocket além de economizar largura de banda, potência de CPU e apresentar latência reduzida, tem implementação simples em comparação às técnicas de conexão em tempo real do HTTP [30].

Com o crescimento de tecnologias que trabalham sobre a base da IoT, a utilização do Websocket se mostra muito atrativa, pois muitas delas necessitam de conexão em tempo real. Em [11] verifica-se a utilização do protocolo Websocket para implementar uma biblioteca em Erlang que permite desenvolver aplicações capazes de se comunicarem diretamente com aplicações em HTML5 sem a necessidade de plugins. Já [6] comprovam a eficácia do Websocket em relação a outras tecnologias de conexão para controle em tempo real de uma placa de Arduino com um tempo de resposta e tráfego de rede menor e baixo custo de banda.

Pode-se observar no trabalho de [2], a proposição de uma aplicação de notificação em tempo real para cidades inteligentes utilizando o Websocket. A proposta inicial de [2] consistia em comparar as tecnologias Comet e Websocket, e assim, identificar qual teria o melhor tráfego de dados gerados tal qual o impacto desse tráfego no consumo de bateria do dispositivo. Devido ao pouco tempo para realização de testes, junto à complexidade de implementação do Comet, apenas o Websocket foi utilizado. Como benefícios, identificam-se a facilidade de conexão, praticidade e melhor adaptabilidade à plataforma Android.

3 METODOLOGIA

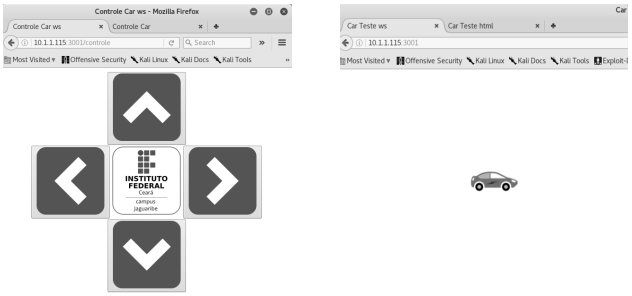
Por se tratar de uma proposta inicial para validação do uso do protocolo Websocket considerando requisitos de sistemas críticos de tempo real com enfoque no tempo lógico, conforme descrito na Seção 1, dois cenários distintos são considerados: o primeiro (simulação) consiste em uma aplicação para o controle de um veículo em um ambiente virtual, mas com toda a interação das conexões de controle na rede *Web* tais como: custo da rede com transferência de informações e latência de resposta dos controles. Já o segundo, consiste em uma implementação embarcada em um microcontrolador

cuja função é controlar parte da interação mecânica presente no circuito eletrônico de movimentação de um veículo, no caso, um dispositivo servo motor, geralmente usado para controle de direção do veículo. Por sua vez, este trabalho apresenta contribuições práticas iniciais sobre a utilização do protocolo Websocket, por isso, experimentos considerando o congestionamento da rede, cenários de comunicação em Redes Veiculares ou ainda a comparação com mais protocolos do cenário de IoT, apresentam-se fora do escopo deste trabalho.

3.1 Simulações em Ambiente Virtual

Para os testes de desempenho e para efeito de comparação, foram criados dois ambientes virtuais: um com o protocolo comumente utilizado pelo HTTP para prover serviço em tempo real, o *Polling*, e outro com o Websocket. A escolha de utilização do *Polling* justifica-se por se tratar da abordagem tradicional utilizada no envio de dados pela *Web* [23] e pela simplicidade de implementação. Vale ressaltar que a mesma base de código foi utilizada em ambos os ambientes virtuais, salvo pequenas alterações apenas para implantação de cada protocolo de comunicação.

A simulação é realizada pelo controle da movimentação de um veículo virtual através de uma página *Web* que contém os comandos básicos de direção (Figura 1a). Por sua vez, a movimentação é ilustrada em outra página *Web*, conforme Figura 1b. Para tal, utilizou-se Node.js juntamente com *JavaScript* e HTML5 para a implementação do servidor e construção das páginas.



(a) Página de controle do veículo **(b) Página de movimentação do veículo**

Figura 1: Cenário para simulação virtual.

A aplicação utilizando o *Polling* faz a comunicação com o servidor simulando uma conexão de tempo real através de várias requisições, onde a página que demonstra o movimento do veículo faz essas requisições HTTP ao servidor até que ele responda com um evento, ou seja, um comando do controle de movimentação (Figura 2). Já na aplicação utilizando o Websocket, ao abrir a página de movimentação do veículo é feito uma requisição HTTP padrão com um pedido de atualização para o protocolo Websocket. O servidor reconhece o protocolo e responde com a diretiva "HTTP 101" no cabeçalho da mensagem de resposta, ou seja, mostrando que o servidor trabalha com o protocolo Websocket. Dessa forma, é aberta uma conexão utilizando a mesma ligação TCP existente entre o veículo e o controle, não sendo mais necessário o HTTP (Figura 3).

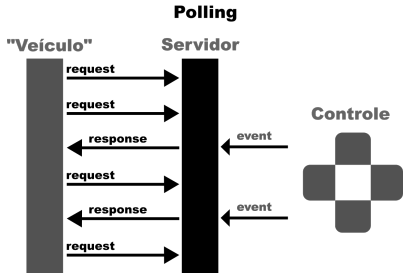


Figura 2: Comunicação utilizando Polling.

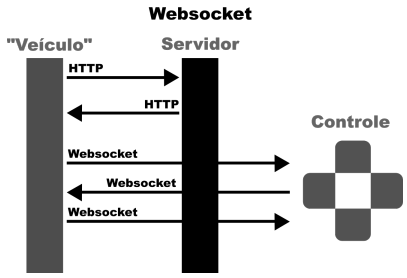


Figura 3: Comunicação utilizado Websocket.

O cenário para execução da simulação em ambiente virtual foi montado a partir da estrutura física e de rede interna do laboratório de redes do IFCE campus Jaguaribe, onde existem conexões ativas cabeadas e sem fio. O servidor Node.js foi hospedado em um computador com processador intel core i3 (Intel Core™i3 CPU M 380 2.53GHz x 4) e 4GB de RAM. Com isso, é possível analisar o controle do veículo através de vários dispositivos distintos como computadores, *smartphones* e *tables*, desde que estejam conectados a rede e possuam um navegador *Web* compatível com o HTML5.

Para a obtenção dos dados da análise, utilizou-se o *Inspector* do Mozilla Firefox'ESR 45.3.0. Essa ferramenta dispõe de opções como monitor de redes e desempenho onde são exibidos dados de comunicação como requisições, tempo de resposta e consumo da rede. Assim, é possível analisar fatores importantes para comparação das aplicações de controle em tempo real, tais como tempo de resposta e sobrecarga da rede.

3.2 Experimento Embarcado

De forma análoga aos procedimentos descritos na Seção 3.1, para a realização dos testes com o experimento embarcado implementou-se um servidor com o HTTP utilizando a técnica *Polling* e outro com Websocket. Os servidores foram embarcados em um kit de desenvolvimento da plataforma Arduino[1]

[1]Arduino Uno - processador ATmega328 - 16 MHz, com 32 kB de memória flash, 2 kB de SRAM e 1 kB de EEPROM.

integrado ao módulo Wi-Fi ESP8266 01-E. O módulo ESP tem por finalidade receber os dados de controle de um cliente via interface sem fio através de um navegador *Web* e assim, o sistema embarcado realiza o controle de um servo motor S3010 - Futaba. A estrutura do experimento é apresentada na Figura 4.

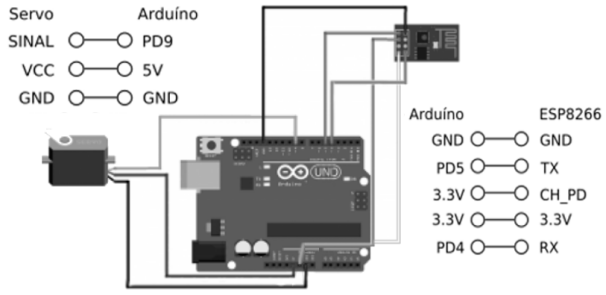


Figura 4: Visão geral do experimento embarcado.

Nesta implementação utilizou-se para o desenvolvimento o mesmo *hardware* descrito na Seção 3.1, acrescentando apenas o Apache2 versão 2.2.9 ao sistema para hospedagem das páginas. O sistema embarcado foi programado através da interface de desenvolvimento Arduíno versão 1.8.2. Utilizou-se ainda as bibliotecas do módulo Wi-Fi ESP8266 versão 2.3.0 para controle de rotinas específicas do módulo, *servo.h* para controle do servo motor, e a *softserial.h* para implementar uma porta serial virtual. Na aplicação utilizando o Websocket importou-se a biblioteca para comunicação desenvolvida por [15]. Assim como na simulação em ambiente virtual, utilizou-se a rede interna do laboratório de redes do IFCE campus Jaguaribe, onde o módulo Wi-Fi foi pré-configurado para se conectar automaticamente a rede.

O cliente (página de controle do servo), foi criada em HTML5 e *JavaScript*, assim como na simulação em ambiente virtual. O *JavaScript* é responsável pela comunicação, isto é, envio e resposta de mensagens. Os arquivos desenvolvidos foram colocados dentro do diretório de arquivos HTML do Apache2, possibilitando assim, o acesso através da rede. A página de controle do servo motor é composta por um campo chamado "Endereço IP ESP8266", onde deve-se inserir manualmente o endereço IP do módulo e um *scrollbar* (barra de rolagem) para controle do servo, como pode ser visto na Figura 5.

Assim como nos testes realizados na Seção 3.1, para a obtenção dos dados de transferência utilizou-se o *Inspector* do Mozilla Firefox'ESR 45.3.0. Vale ressaltar que diferente da simulação no ambiente virtual, o objeto controlado não está em uma página *Web*. Logo para analisar os dados de transmissão especificamente do *Polling* que faz requisições periódicas, foi desenvolvida uma outra página, que através do *Inspector*, são visualizadas as requisições ao servidor para controle do dispositivo servo motor.

Figura 5: Página de controle do servo motor.

4 RESULTADOS

A partir do modelo metodológico apresentado na Seção 3 e suas subseções, foram realizados testes comparativos com ênfase nas seguintes métricas de desempenho a saber: quantidade de requisições, quantidade de dados transferidos e tempo total gasto para estabelecimento das conexões.

4.1 Resultados dos Testes em Ambiente Virtual

Conforme ilustrado na Figura 6, o primeiro teste consiste na aplicação utilizando HTTP *Polling*. Esta página foi configurada para fazer requisições automáticas a cada 1s. Separadamente, cada requisição tem aproximadamente 0,02kB de tamanho. Entretanto, por ser um loop, a página é carregada continuamente, tendendo a uma sobrecarga no servidor.

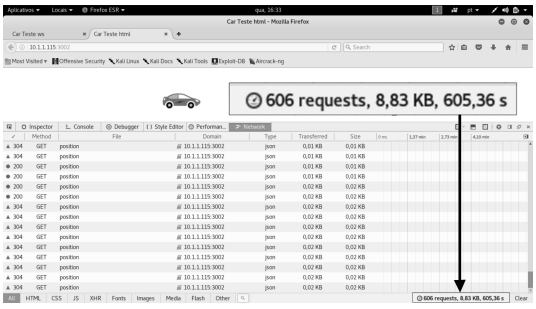


Figura 6: Dados obtidos utilizando HTTP em ambiente virtual.

Ainda na Figura 6, é possível observar que durante um intervalo de tempo de 605,36s a página realiza 606 requisições para um *download* de apenas 8,83kB de dados da página. Ressalta-se que estes dados estão em contínuo crescimento enquanto a aplicação estiver ativa.

Com a configuração atual observa-se uma latência de no mínimo 1s, pois é o tempo de atualização da página a cada requisição, o que representa um tempo muito alto para o modelo de aplicações em tempo real considerado nesse estudo. Com o intuito de obter uma latência menor, os intervalos das requisições foram reduzidos de forma gradativa até a ordem de alguns milissegundos.

Obedecendo ao mesmo tempo ativo de aproximadamente 606 segundos, conforme ilustrado na Figura 7, foram obtidos os seguintes resultados: O primeiro intervalo de requisições analisado é de 1s, já comentado; Em seguida atribui-se um intervalo de 500ms, gerando 1212 requisições e 18,46kB de dados totais transferidos; Com um intervalo de 300ms, obtém-se 1666 requisições e 24,83kB de *download*; e por fim o intervalo mais curto de 200ms, resulta em 1847 requisições e 27,52kB de dados transferidos.

Figura 7: Comparativo entre intervalos de tempo de requisição.

Observa-se uma melhora na resposta dos comandos a cada redução do intervalo de requisição. No entanto, há um aumento considerável na taxa de requisições e no tamanho de dados para manter a página atualizada e funcional. Sabendo que as requisições são constantes havendo ou não controle do veículo, é gerado um custo desnecessário à rede com transferência de dados, além da possibilidade de ocasionar uma sobrecarga do servidor. Considerando a análise da quantidade de dados transferidos *versus* a quantidade de requisições em função do intervalo de solicitações apresentada anteriormente e ainda que valores de latência entre 100ms e 300ms apresentam um pequeno delay perceptível em aplicações com essa natureza [4], define-se então como intervalo de requisições ideal, o valor de 300ms para os demais testes a serem realizados. Desta forma, o tempo de resposta está dentro das métricas aceitáveis para a aplicação em tempo real admitida no estudo além de apresentar um menor custo a rede.

A Figura 8 apresenta os resultados obtidos através da aplicação utilizando o Websocket mediante às mesmas configurações base dos testes anteriores. Após a página ser carregada, a mesma permanece ativa com 7 requisições em 0,30s com um *download* de 71,19kB. Com a conexão do Websocket, é estabelecido um canal *full-duplex* entre a página de simulação de movimentação e a página de controle, não existindo desgastes da rede com requisições periódicas como acontece no HTTP.

O teste final consiste em estabelecer conexões diferentes simultâneas para cada protocolo e obter a média aritmética da quantidade de requisições, tempo total e da quantidade de dados transferidos. Iniciou-se o teste com 2 conexões simultâneas para observação dos dados obtidos. Em seguida, aumentou-se o número de conexões de forma gradativa. Notou-se então que os resultados obtidos aumentam linearmente com o número de conexões. Logo, estabeleceu-se como quantidade padrão, 10 conexões simultâneas. Outro ponto é que devido a utilização da técnica *Polling* no HTTP, o tempo definido para a captura dos dados é de 30 minutos, evitando assim uma sobrecarga na rede. O resultado comparativo é apresentado na Figura 9.

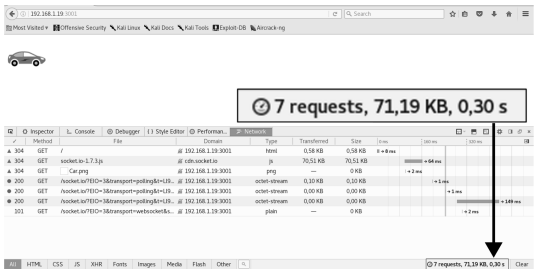
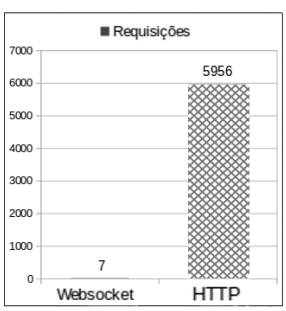
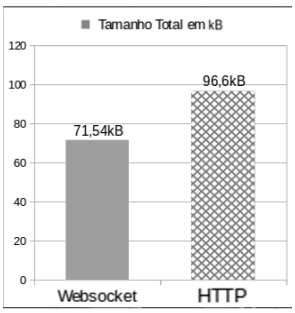


Figura 8: Dados obtidos utilizando Websocket em ambiente virtual.



(a) Média de requisições



(b) Média dos dados transferidos

(c) Média da duração para as conexões

Figura 9: Comparação Websocket *versus* HTTP *Polling* em ambiente virtual.

O Websocket apresenta um melhor desempenho em relação ao HTTP em todas as métricas analisadas. Em 0,772s, o Websocket apresenta uma média de 7 requisições e 71,54kB de

dados transferidos para estabelecer a conexão e ficar ativo, enquanto o HTTP, necessita de 5956 requisições para transferir 96,6kB em um tempo médio de 1792,27s. Isso significa que para controlar o veículo virtual utilizando o Websocket, consegue-se diminuir em 25,95% a quantidade de dados transmitidos pela rede. Vale ainda ressaltar que, com menos 1% da quantidade de requisições necessárias pelo HTTP *Polling* e com uma economia de tempo para estabelecimento das conexões superior a 98%, o Websocket consegue manter a aplicação funcional.

4.2 Resultados dos Testes no Experimento Embarcado

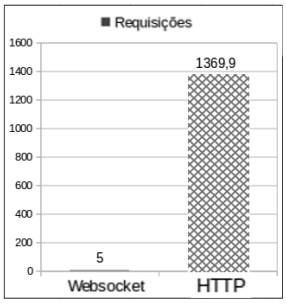
Através da página desenvolvida para visualizar o movimento das requisições no experimento embarcado utilizando HTTP *Polling*, pôde-se observar os dados a serem analisados. Como pode ser visto na Figura 10, na implementação com o HTTP, 51 requisições são realizadas para transferir 12,70kB em aproximadamente 15 segundos.

Figura 10: Dados obtidos utilizando HTTP no experimento embarcado.

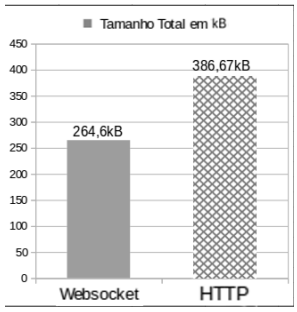
Já na implementação com o Websocket caracterizada por criar uma conexão direta bidirecional, os resultados podem ser obtidos a partir da própria página de controle do servo motor. Na Figura 11 é possível observar que a conexão da página de controle com o sistema embarcado ocorre com apenas 5 requisições, totalizando 264,60kB de transferência de dados em 0,57s.

Figura 11: Página de Controle do servo motor com Websocket.

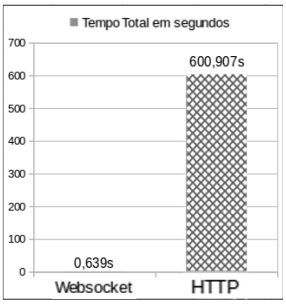
Assim como na Seção 4.1, para concluir os resultados no experimento embarcado, são estabelecidas 10 conexões diferentes simultâneas para cada protocolo e calculada a média aritmética da quantidade de requisições, tempo total e da quantidade de dados transferidos. A Figura 12 ilustra os resultados obtidos.



(a) Média de requisições



(b) Média dos dados transferidos



(c) Média da duração para as conexões

Figura 12: Comparação Websocket *versus* HTTP *Polling* no experimento embarcado.

Novamente é possível notar que o Websocket se apresenta como uma implementação de melhor conexão em termos gerais. Com uma média de 5 requisições para transferência de 264,6kB em 0,639s o cliente estabelece conexão com o sistema embarcado, obtendo total controle do servo motor. Já a implementação com o HTTP ativo por um período de 10 minutos, resulta em uma média 1369,9 requisições, com transferência de 386,67kB em um tempo de 600,907s para prover o mesmo serviço de controle em tempo real. Nota-se que é possível controlar um dispositivo físico através da *Web* com 31,57% de dados a menos transferidos pela rede. O tempo total para o estabelecimento das conexões utilizando o protocolo proposto reduz em mais de 99%, sendo necessárias cerca de 1364 conexões a menos para ter total controle do dispositivo servo motor quando comparada ao HTTP *Polling*.

5 CONCLUSÃO

Neste trabalho é proposto a utilização do protocolo Websocket para o controle de dispositivos remotamente em tempo real

por uma rede de dados. Estudos demonstram que esforços em desenvolvimento nas áreas de IoT via Web utilizando HTML5 junto à API de Websocket, fornecem um grande passo frente a escalabilidade da Web em aplicações de tempo real. Como demonstrado, o Websocket é uma solução viável para serviços de tempo real com conexão via *Web*. A otimização realizada pelo uso do protocolo Websocket em termos de quantidade de requisições, supera a faixa de 99% tanto no ambiente virtual quanto no experimento embarcado. Nota-se ainda que a quantidade de dados transferidos quando se trabalha com o HTTP *Polling*, aumenta em mais de 74% nas simulações e cerca de 68% para o cenário com o dispositivo embarcado. Por fim, o tempo médio de conexão alcançado pelo uso do protocolo Websocket, representa, em ambos os cenários considerados, uma economia superior a 99%.

Como trabalhos futuros pode-se sugerir:

- aplicar essa ideia em um protótipo completo com toda a interação de um veículo;
- implementar o protocolo CoAP (*Constrained Application Protocol*), idealizado para uso em dispositivos de internet com recursos limitados, como os nós de rede de sensores sem fio [25] e comparar os resultados com a implementação do Websocket. O CoAP vem ganhando espaço em trabalhos com IoT, como exemplos pode-se citar [20] e [18];
- considerar experimentos em cenários de rede que apresentam congestionamento.

Demonstra-se aqui a viabilidade de tal implementação, uma vez que a base da conexão da rede de dados é a mesma. Vale salientar que existem muitos pontos a serem analisados como *hardware*, sistemas eletrônicos e a própria estrutura das redes.

REFERÊNCIAS

[1] Hyggo Almeida. 2015. Internet das Coisas: Tudo conectado. *Computação Brasil* 25, 58 (Jul 2015), 7–8.

[2] Sean Carlisto de Alvarenga. 2013. *Tecnologias Push e uma arquitetura de notificação para cidades inteligentes*. Bacharelado em Ciência da Computação. Universidade Estadual de Londrina, Londrina PR.

[3] Douglas E Comer. 2016. *Redes de Computadores e Internet-6*. Bookman Editora.

[4] Cleber Dantas. 2013. Latência, largura de banda e a velocidade da luz. (2013). https://tableless.com.br/latencia-largura-de-banda-e-a-velocidade-da-luz. Acesso em 29 mai. 2017.

[5] Fernando Luiz de Oliveira and Fabiano Fagundes. 2010. Proposta de adequação do padrão HTML + TIME ao modelo de referência de sincronização multimídia. (2010).

[6] Jair Vargas dos Santos, Marco Antônio Silveira de Souza, and Daniel Fernando Anderle. 2015. Controlando Dispositivos em Tempo Real Através do WebSocket. *Tecnologias e Redes de Computadores: estudos aplicados* 88960 (2015), 206.

[7] Marcos Pereira dos Santos and Marco Túlio Chella. 2013. Laboratório de acesso remoto para controle, experimentação, automação e validação de processos em tempo real, "Baseado em Websocket". *Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação* 3, 1 (Nov 2013), 85–92.

[8] Diego Eis and Elcio Ferreira. 2012. *HTML5 e CSS3 com farinha e pimenta*. São Paulo, Brasil: Tableless.com.br.

[9] Jean-Marie Farines, Joni da Silva Fraga, and RS de Oliveira. 2010. *Sistemas de tempo real*. Escola de Computação. 201 pages.

[10] I Fette and A. Melnikov. 2011. RFC 6455 - The WebSocket Protocol. (2011). https://tools.ietf.org/html/rfc6455. Acesso em 16 set. 2016.

[11] Emiliano Carlos de Moraes Firmino. 2011. *Websocket para aplicações Web em Erlang*. Graduação em Engenharia de Computação. Universidade do Estado do Amazonas, Manaus-AM.

[12] David Flanagan. 2006. *JavaScript: the definitive guide*. "O'Reilly Media, Inc.".

[13] Guilherme da Silva Alves Gonçalves, Paulo Henrique Ouverney Bastos, et al. 2014. Um modelo de arquitetura utilizando o protocolo websocket para prover um serviço restful para sistemas em tempo real. (2014).

[14] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[15] Branden Hall. 2014. Arduino-Websocket. (2014). https://github.com/brandenhall/Arduino-Websocket. Acesso em 26 abr. 2017.

[16] Hyunjoo Jin. 2016. Hyundai Motor, Cisco to team up on Internet-connected car technology. (2016). http://www.reuters.com/article/us-hyundai-motor-cisco-systems-idUSKCN0XG04C. Acesso em 19 abr. 2016.

[17] James F Kurose and Keith W Ross. 2013. *Redes de Computadores e a Internet: uma abortagem top-dowm* (6th ed.). São Paulo: Person.

[18] Alan Tomás Lima. 2014. *Aplicação de Internet of Things em casas inteligentes-Serviço Aplicacional*. Ph.D. Dissertation. Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto.

[19] Andrew Lombardi. 2015. *WebSocket: Lightweight Client-Server Communications*. "O'Reilly Media, Inc.".

[20] Ismael Rodrigues Martins and José Luís Zem. 2016. Estudo dos protocolos de comunicação MQTT e COaP para aplicações machine-to-machine e Internet das coisas. *Revista Tecnológica da Fatec Americana* 3, 1 (2016), 24.

[21] Cintia Carvalho Oliveira, Daniele Carvalho Oliveira, João Carlos Gonçalves, and Julio Toshio Kuniwake. 2016. Pratical Introduction to Internet of Things: Practice using Arduino and Node. js. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*. 17–18.

[22] Jin-tae Park, Hyun-seo Hwang, Jun-soo Yun, and Il-young Moon. 2014. Study of HTML5 WebSocket for a Multimedia Communication. *International Journal of Multimedia and Ubiquitous Engineering* 9, 7 (2014).

[23] P.F. Pires, F. Delicato, T. Batista, T. Barros, E. Cavalcante, and M. Pitanga. 2015. *Plataformas para a internet das coisas*. Technical Report. Minicursos SBRC - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

[24] Susanna Ray. 2016. Toyota and Microsoft connect drivers to the future. (2016). https://blogs.microsoft.com/transform/2016/04/04/toyota-and-microsoft-connect-drivers-to-the-future. Acesso em 19 abr 2016.

[25] Z. Shelby, K. Hartke, and C. Bormann. 2014. The Constrained Application Protocol (CoAP). (2014). https://tools.ietf.org/html/rfc7252. Acesso em 16 mai. 2017.

[26] Maurício Samy Silva. 2010. *JavaScript: guia do programador*. São Paulo: Novatec.

[27] Stefan Tilkov and Steve Vinoski. 2010. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing* 14, 6 (2010), 80–83.

[28] Gabriel Torres. 2015. *Redes de computadores*. Novaterra Editora e Distribuidora LTDA.

[29] T Varela and L Stanley. 2012. Implementação e análise da utilização de websockets em sistemas computacionais. (2012).

[30] Vanessa Wang, Frank Salim, and Peter Moskovits. 2013. *The definitive guide to HTML5 WebSocket*. Vol. 1. Berkeley, Calif, USA: Apress.

[31] Alexandre Stürmer Wolf, Eduardo Augusto Lieberknech, Eric Augusto Ruebenich de Quadros, Estevan Luiz Junges, and Luan Araujo dos Santos. 2016. VEÍCULO TERRESTRE NÃO TRIPULADO CONTROLADO VIA REDE WI-FI. *Destaques Acadêmicos* 7, 4 (2016).