Instituto Superior de Engenharia de Lisboa

Sistemas Distribuidos

Relatório do trabalho de investigação

Websockets

Autoria:
33724 David Raposo
32632 Pedro Pedroso
33404 Ricardo Mata

Em coordenação com:
Engº Luís Falcão
Engº José Simão

26 de Junho de 2014

Conteúdo

1 Introdução

Com os avanços tecnológicos que têm havido nos últimos anos, é cada vez mais fácil perder a noção do que está a acontecer dentro dos computadores. Isto faz com que se tenha algum desleixo perante os recursos que se usam. No entanto, com os dispositivos móveis em constante crescimento[1], volta a tornar-se importante a otimização dos recursos usados.

A compatibilidade entre diferentes plataformas é garantida pela utilização de protocolos que faz com que cada plataforma saiba comunicar entre si. Um exemplo desses protocolos é o protocolo HTTP[2], que surgiu como necessidade de transferir conteúdo estático (páginas de hipertexto). Desde a sua implementação, o protocolo foi beneficiando de revisões que expandiram o seu uso original. Contudo, o propósito continuava a ser o mesmo: transmissão de conteúdo estático entre dispositivos.

FALAR SOBRE WEB APPS

2 Visão global

Antes da adoção dos *websockets*, os pedidos web eram feitos puramente através do protocolo HTTP. Cada pedido estava sujeito às limitações deste protocolo, como por exemplo, cada pedido tinha que conter o cabeçalho HTTP correspondente, e cada pedido estava afeto a apenas uma conexão.

Nas aplicações simples, isto não é problemático. Caso fizessem poucos pedidos, era criadas poucas ligações, e se pedissem muitos dados de uma só vez o "'*overhead*"' do cabeçalho era negligenciável. Contudo, em aplicações mais complexas que precisem de dados não estáticos, ou gerados em tempo real isto é um problema que pode arrastar a performance do sistema todo.

Os pedidos de informação não estática são geralmente feitos das seguintes formas:

1. *Polling*: Consiste em efetuar periodicamente pedidos a questionar o servidor se existem novos dados a obter. Isto trás um custo elevado, pois ao serem feitos constantemente pedidos é necessário estar a criar novas conexões constantemente, e podem haver bastantes pedidos a que o servidor não tenha informação para enviar. Isto adicionado ao facto de que é necessário incluir os cabeçalhos HTTP causa um constrangimento enorme à rede.

2. *Long-Polling:* Semelhante ao *polling*, mas o servidor prende a ligação até que haja informação a enviar. Assim que haja informação a enviar, o servidor envia-a e, seguindo o protocolo HTTP, fecha a ligação. É mais vantajoso que o *polling*, pois os recursos só são libertados assim que não forem necessários, apesar de ter de ser necessário criar novas ligações para novos pedidos.

3. *Pushing:* É feito um pedido de dados ao servidor. O servidor mantém a ligação aberta e vai enviando dados ao cliente assim que houverem novos

[1] Fonte: `http://www.digitalbuzzblog.com/infographic-2013-mobile-growth-statistics/`

[2] A sigla HTTP vem de *HyperText Transfer Protocol*, que significa Protocolo de Transmissão de Hipertexto.

```
GET /chat HTTP/1.1
Host:  server.example.com
Upgrade:  websocket
Connection:  Upgrade
Sec-WebSocket-Key:  dGhlIHNhbXBsZSBub25jZQ==
Origin:  http://example.com
Sec-WebSocket-Protocol:  chat, superchat
Sec-WebSocket-Version:  13
```

Figura 1: Pedido de um cliente

```
HTTP/1.1 101 Switching Protocols
Upgrade:  websocket
Connection:  Upgrade
Sec-WebSocket-Accept:  s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol:  chat
```

Figura 2: Resposta de um servidor

dados. É mais vantajoso face a *Long-Polling* na medida que o cliente recebe novos dados sem ter de iniciar uma nova ligação, no entanto, pedidos seguintes têm que iniciar uma nova ligação.

Estes métodos de pedidos são desvantajosos para o enorme volume de dados a que as aplicações complexas estão sujeitas, ainda para mais com o constante crescimento de tráfego da internet[3], o que faz com que seja desejável que as aplicações sejam o mais eficiente possível.

O que os *websockets* trazem é uma nova forma de fazer pedidos, reaproveitando a ligação do pedido original. Isto permite fazer vários pedidos sem o custo adicional acrescido dos cabeçalhos HTTP nem da criação de novas ligações.

3 Funcionamento

Para utilizar *websockets* é necessário fazer um pedido inicial através de HTTP. A informação contida no cabeçalho indicará ao servidor que se pretende fazer a comunicação através de *websockets*. O servidor depois responde com sucesso ou insucesso, dependendo se suporta ou não o protocolo.

Nas figuras 1 e 2 podemos ver um exemplo de cabeçalhos *HTTP* usados no *handshake* inicial entre um cliente e um servidor. No fundo, são cabeçalhos HTTP perfeitamente normais, com pares de campos//valor. Podemos ver que o cliente pretende efetuar a comunicação através de um canal *websocket* através do campo *Connection*. O campo *Connection*, quando contem o valor *Upgrade*

[3]Fonte: `http://en.wikipedia.org/wiki/Global_Internet_usage`

indica que a comunicação deve passar a ser feita por *websocket*. O campo *Sec-WebSocket-Key* contem um valor codificado em base 64 que é processado pelo servidor, cujo resultado do processamento é enviado para o cliente no campo *Sec-WebSocket-Accept* da resposta. O servidor ao receber esta string concatena um valor constante, volta a converter para base 64 e é interpretado pelo cliente para saber se o cabeçalho da resposta equivale realmente a um cabeçalho de sucesso para *websocket*. O campo *Sec-WebSocket-Protocol* indica quais os sub protocolos é que o cliente pretende utilizar. Na figura, o cliente pretende utilizar os protocolos *chat* e superchat. O servidor, entre todos os sub protocolos que conhece, escolhe apenas um daqueles que vem no pedido do cliente (desde que o conheça), e coloca o protocolo escolhido no campo *Sec-WebSocket-Protocol* da resposta. Apesar desta verificação, ao receber a resposta do servidor o cliente vai verificar se o protocolo enviado corresponde a algum dos que ele colocou.

Após o *handshake* terminar, a comunicação entre o servidor e o cliente são feitas através de um *websocket*. A partir deste ponto,

4 Vantagens

Como já foi referido, *websockets* permitem que se faça comunicação entre aplicações e servidores web sem se estar preso ás limitações do protocolo HTTP. Isto porque no final do *handshake* os intervenientes da ligação comunicam diretamente sobre o canal TCP aberto, e é nisto em que consistem os *websockets*.

Os *websockets* têm o conceito de mensagens. Nativamente, o protocolo TCP funciona com *streams* de *bytes*. A implementação de *websockets* permite trabalhar num nível acima de *streams*, para abstrair o cliente da necessidade de cuidar da transferência de dados. Fazendo um paralelo com *sockets*: Numa aplicação que utilize *sockets*, uma chamada ao método *recv* iria retornar um conjunto de bytes que podem pertencer a mais que uma mensagem. Com *websockets*, há a garantia que uma chamada a *recv* retorna não só os bytes de apenas uma mensagem, como retorna todos os bytes da mensagem.

Referências

[1] I. Fette, A. Melnikov (December 2011) *The WebSocket Protocol PROPOSED STANDARD*

http://tools.ietf.org/html/rfc6455