INF01151 – Sistemas Operacionais II

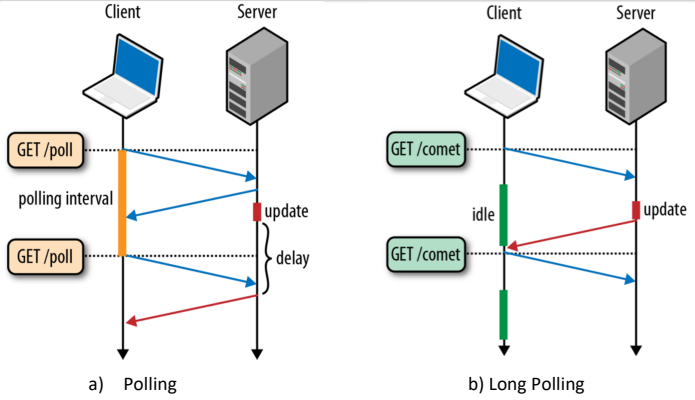
WebSockets

Prof. Alberto Egon Schaeffer Filho

Introdução

- **Problema:** Criar aplicações WEB (*client-server*) que necessitavam de comunicação bidirecional requeria que o servidor ficasse sendo consultado de forma contínua em busca de atualizações. (*polling*)

Introdução



Introdução

Essa abordagem acarreta alguns problemas:

- **Servidor com inúmeras conexões TCP para cada cliente:** tanto para enviar informações e para receber as requisições
- **Consumo de Largura de banda:** requisições sendo respondidas mesmo quando nenhum dado novo está disponível
- **Aplicações-cliente:** Mapeamento das várias conexões feitas às conexões de resposta.

Introdução

- **A solução:** Utilizar uma única conexão TCP para o tráfego de forma bidirecional, sem utilizar *polling* HTTP. Assim, têm-se o protocolo **WebSocket,** bem como a sua API (WSAPI).

WebSockets: a definição

"O protocolo WebSocket permite a comunicação em duas vias entre um cliente, executando código não-confiável em um ambiente controlado, a um host remoto que aceitou estabelecer comunicação a esse código." [RFC 6442]

WebSockets: a definição

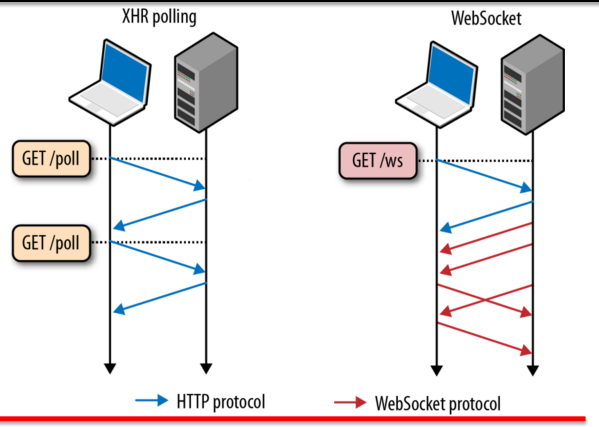
O protocolo WebSocket é um protocolo independente baseado em TCP, que utiliza HTTP para processar o handshake entre as entidades.

Utiliza a porta 80 para conexões regulares e 443 para conexões com TLS (Transport Layer Security), por padrão.

Objetivos de WebSockets

Resolver os empecilhos encontrados nas soluções HTTP existentes que implementavam comunicação bidirecional: Prover uma maneira para as aplicações estabelecerem comunicação em duas vias com servidores, sem depender da abertura de múltiplas conexões HTTP.
(ex: *XMLHttpRequest, long polling, ...*)

Objetivos de WebSockets

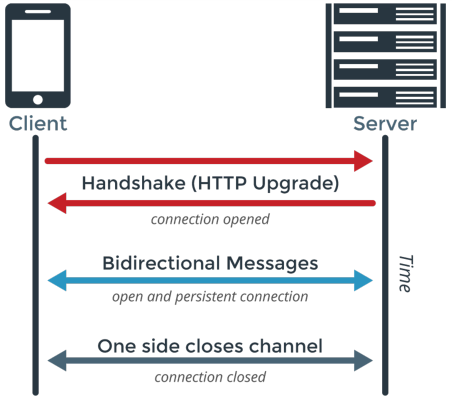


WebSockets: estrutura

O protocolo possui duas partes essenciais: o *handshake* (para estabelecimento da conexão entre ambas as partes) e o envio de dados.

Após o envio do *handshake* por ambas as partes (e se foi bem sucedido), ambas as entidades estão aptas a enviar dados através do canal estabelecido.

WebSockets: estrutura



WebSockets: estrutura

Opening Handshake

- O *handshake* cliente é uma requisição de Upgrade HTTP
- A ordem dos campos no *header* não importa

```
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
```

WebSockets: estrutura

Opening Handshake (client)

"Request URI": identificar o endpoint da conexão WebSocket. Para permitir vários domínios servidos por um único endereço IP e múltiplos endpoints servidos por um único servidor.

```
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
```


WebSockets: estrutura

Opening Handshake (client)

Cliente inclui o *hostname* na requisição para que ambos (cliente e servidor) possam concordar qual host estará em uso.

```
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
```


WebSockets: estrutura

Opening Handshake (client)

HTTP Upgrade: define o protocolo que será utilizado após a concordância no handshake.

```
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
```


WebSockets: estrutura

Opening Handshake (client)

Subprotocol Selector: campo adicional para indicar quais subprotocolos (a nível de aplicação) são aceitos pelo cliente. O servidor seleciona qual usará e o mencionará no seu handshake para indicar que o selecionou.

```
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
```

WebSockets: estrutura

Opening Handshake (client)

Informa o servidor sobre a origem da requisição. Se o servidor não deseja estabelecer conexão com essa origem, responde com um código de erro HTTP para rejeitá-la.

```
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
```


WebSockets: estrutura

Opening Handshake (client)

Para provar que o handshake foi recebido, o servidor pega o valor desse campo, concatena-o com seu identificador (GUID), aplica hash SHA-1, codifica com Base64 e o retorna em seu próprio *handshake*.

```
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
```


WebSockets: estrutura

Opening Handshake (server)

HTTP status line: retorna o código HTTP 101 para sinalizar ao cliente qual protocolo foi aceito. Qualquer código diferente de 101 indica que o handshake WebSocket não foi concluído e que a semântica do HTTP ainda é aplicável.

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


WebSockets: estrutura

Opening Handshake (server)

Indica o aceite da parte do servidor em manter conexão. Contém a concatenação da hash do cliente, presente em **Sec-WebSocket-Key.** Esse campo será verificado pelo cliente e se não for o valor esperado, a conexão não será estabelecida.

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

WebSockets: estrutura

Opening Handshake: requisitos para o cliente

- Prover campos válidos para a requisição
- Se um cliente já possui conexão para um host identificado por <host><port> ele deve esperar ela ser estabelecida ou falhar para realizar outra como mesmo esquema. Não deve existir mais de uma conexão no estado CONNECTING.

WebSockets: estrutura

Opening Handshake: requisitos para o cliente

Não há limites para o número de conexões WebSocket estabelecidas que um cliente possa ter com um mesmo servidor, porém servidores podem recusar estabelecer conexão com hosts com número excessivo de conexões já estabelecidas, quando sobrecarregados.

WebSockets: estrutura

Opening Handshake: requisitos para o servidor

- Processar parte do *opening handshake* do cliente para extrair informações necessárias.
- Verificar violações de formato nos dados recebidos
- Retornar códigos de erro caso encontre violações;
- Se aceitar a conexão do cliente, retornar a ele uma resposta HTTP válida.

WebSockets: estrutura

WebSockets URIs

- Define dois esquemas para URIs: `ws` e `wss`

```
ws-URI = “ws:” “//” host[“:” port] path [“?” query]
wss-URI = “wss:” “//” host[“:” port] path [“?” query]
```

ex: “ws://example.com/home”

WebSockets: *Framing*

- Dados são enviados em sequências de quadros (*frames*)
- O cliente deve mascarar os *frames* enviados
- O servidor não deve mascarar os *frames* enviados ao cliente.

WebSockets: *frames*

- *Control frames*
  - Ping
  - Pong
  - *Close frame*
- *Data frames*
  - *Text*
  - *Binary*

WebSockets: Envio de dados

Para o envio de dados sobre uma conexão WebSocket:

- Assegurar que a conexão está aberta, ou abortar o envio.
- Encapsular os dados em um quadro WebSocket. Se o tamanho for muito grande, encapsular em uma série de *frames*
- Cliente deve mascarar os dados.

WebSockets: Receber dados

- "Escutar" na conexão estabelecida
- Checar o tipo dos dados recebidos (*data frames*)
- Processar *control frames*
- Observância dos FIN bits
- Remover o mascaramento dos dados feito pelo cliente

WebSockets: *closing handshake*

- Deve-se utilizar um método que feche a conexão de forma segura.
- Realizar o envio de um *close frame* identificando a razão da finalização da conexão.
- Após o envio e recebimento do *close frame*, o endpoint deve fechar a conexão WebSocket.

WebSockets e HTTP

- WebSocket utiliza HTTP/1.1 para o processamento do Handshake.
- HTTP/2 não menciona WebSockets em sua definição
  - Porém possui melhorias para HTTP/1.1
  - Algumas de suas *features* podem ser confundidas como uma substituição do protocolo WebSocket.

WebSocket VS. HTTP/2.0

	HTTP/2	WebSocket
Headers	Compressed (HPACK)	None
Binary	Yes	Binary or Textual
Multiplexing	Yes	Yes
Prioritization	Yes	No
Compression	Yes	Yes
Direction	Client/Server + Server Push	Bidirectional
Full-duplex	Yes	Yes

Permite múltiplos fluxos de dados sobre uma única conexão TCP

WebSocket VS. HTTP/2.0

	HTTP/2	WebSocket
Headers	Compressed (HPACK)	None
Binary	Yes	Binary or Textual
Multiplexing	Yes	Yes
Prioritization	Yes	No
Compression	Yes	Yes
Direction	Client/Server + Server Push	Bidirectional
Full-duplex	Yes	Yes

Server Push: Servidor encaminha à cache do cliente dados disponíveis, sem necessidade de uma requisição. Porém, cabe à aplicação a monitoração dos eventos originados pelo servidor.

Leituras adicionais

- [RFC 6455] - The WebSocket Protocol. <https://tools.ietf.org/html/rfc6455>

- [RFC 6202] - Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. <https://tools.ietf.org/html/rfc6202>