UNIVERSIDAD
DE GRANADA

Fundamentos de Redes

Comunicación bidireccional en arquitecturas cliente/servidor

*El protocolo **WebSocket***

Juan Francisco Díaz Moreno
Javier Sáez de la Coba

Curso 2018-2019

22 de noviembre de 2018

Índice

1. Introducción

El modelo Cliente/Servidor es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios o cualquier recurso de un servidor.

El cliente es el que inicia un requerimiento de servicio. Administran la interfaz de usuario, procesan la lógica de la aplicación, recibe los resultados y les da formato.

El servidor es la entidad que provee un servicio y devuelve resultados; ejecuta el procesamiento de datos, aplicaciones y manejo de la información o recursos. Es reactivo, pues realiza una función posterior a una petición o a la ejecución de una transacción requerida por el cliente u otro servidor.

¿Que ocurriría si necesitáramos que el servidor dejara de ser un elemento reactivo y tomara un papel activo en la comunicación y pudiera enviar datos cuando quiera al cliente sin tener que esperarlo?

2. Evolución histórica

Vamos a comenzar resumiendo las técnicas de comunicación bidireccional que existían antes de la llegada de los WebSockets.

Internet no comenzó preparado para ser tan dinámico, si no para ser una colección de documentos HTML conectados unos con otros mediante enlaces formando una red de información. Los clientes únicamente descargaban de los servidores Web las páginas HTML tal y como estaban guardadas en éstos. Los servicios tecnológicos evolucionaron permitiendo páginas dinámicas, es decir, páginas cuyo contenido se generaba para cada cliente conectado.

Aunque las páginas web fueran dinámicas usando scripting de lado del cliente con lenguajes como *VBScript* o *JavaScript* las páginas no podían recibir información del servidor más allá de la obtenida en la carga inicial. Si se querían obtener nuevos datos había que recargar la página completa.

Una evolución de estas páginas dinámicas fue el desarrollo de sitios web que imitaban la utilidad de aplicaciones de escritorio. Para evitar recargar la página completa, se dividía en marcos (*iframes*) que contenían elementos de la aplicación. Estos marcos se recargaban de forma independiente y conseguían mostrar información actualizada, pero con el inconveniente de que esto solo ocurría cuando el cliente hacía una petición al servidor.

Estas técnicas arriba descritas solo nos muestran casos en los que el cliente pide datos nuevos al servidor, pero si el servidor tiene información nueva que dar al cliente no podía mandársela hasta que no hiciera una petición.

La primera aproximación a la solución de que el servidor mandara información al cliente era la técnica del *HTTP Polling*. En ella el navegador automáticamente hacía peticiones al servidor de forma periódica en busca de nuevos datos. Esta técnica era muy ineficiente y bastante ineficaz. Esta técnica no es una comunicación real entre el servidor y el cliente, esto puede llevar a sobrecargas en la red y en el servidor debido a peticiones innecesarias por parte del cliente para intentar obtener datos nuevos cuando realmente no era necesario.

Una solución a esta técnica pasó por usar plugins externos. Uno de ellos era LiveConnect. Con LiveConnect un applet escrito en Java se comunicaba con el servidor y pasaba la información a los scripts JavaScript que se ejecutaban en la página. Un problema grave de LiveConnect era que dependía de la versión exacta de la máquina virtual de Java integrada en el navegador, lo que llevaba a problemas de compatibilidad entre distintas plataformas. Debido a esto se popularizó la técnica del Forever Frame. Esta forma de comunicación servidor ->cliente aprovecha la extensión HTTP 1.1 y su opción de chunked encoding, donde los datos se mandan de forma troceada. Esto permitía abrir un iframe de fondo que pidiera al servidor información. El servidor lo mandaba los datos de forma troceada. A medida que iba llegando se iba cargando en el marco y un script en la página iba comprobando los datos contenidos por ese marco.

Con la llegada de AJAX (Asynchronous JavaScript And XML) las peticiones al servidor no tienen por qué ser iniciadas por el usuario. Desde el mismo código JavaScript de la página se inician peticiones al servidor. Esto posibilita las técnicas de Long-Polling, donde el código AJAX abre una conexión al servidor que se mantiene abierta. Cuando el servidor manda información la conexión se cierra y el cliente vuelve a abrirla a la espera de nuevos datos.

Con la llegada de HTML5 se introdujeron nuevos métodos de comunicación cliente-servidor. Uno de ellos son los SSE (Server-Sent Events). Mediante estos el servidor puede mandar mensajes de forma asíncrona al cliente. Sin embargo, el cliente tiene que hacer una petición POST normal para responder al servidor, haciendo así dos conexiones distintas. Con la llegada del protocolo WebSockets cliente y servidor mantienen una conexión única activa entre ellos por los que pueden mandar mensajes de forma bidireccional. Estos mensajes pueden ser tanto texto como datos binarios, permitiendo mandar no solo información textual (contenido texto de la página o scripts javascript) sino elementos multimedia como imágenes o vídeos entre otros.

3. Presentación del protocolo WebSockets

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse en cualquier aplicación cliente/servidor.

Con esta API, puede enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar al servidor para una respuesta.

Los WebSockets necesitan ser implementados tanto del lado del cliente como del servidor. En el primero, la API es una parte de HTML 5. En el lado del servidor necesitamos una librería que implemente WebSockets, aunque en la actualidad casi todos los servidores las soportan.

En el lado del cliente, WebSocket está ya implementado en Mozilla Firefox 8, Google Chrome 4 y Safari 5, así como la versión móvil de Safari en el iOS 4.2 y en el Internet Explorer 10.

Ejemplos de usos de WebSockets:

- Whatsapp Web.
- Facebook
- Netflix.
- Twitch.
- Bet365.
- ShareLatex
- GitHub

4. Historia de WebSockets

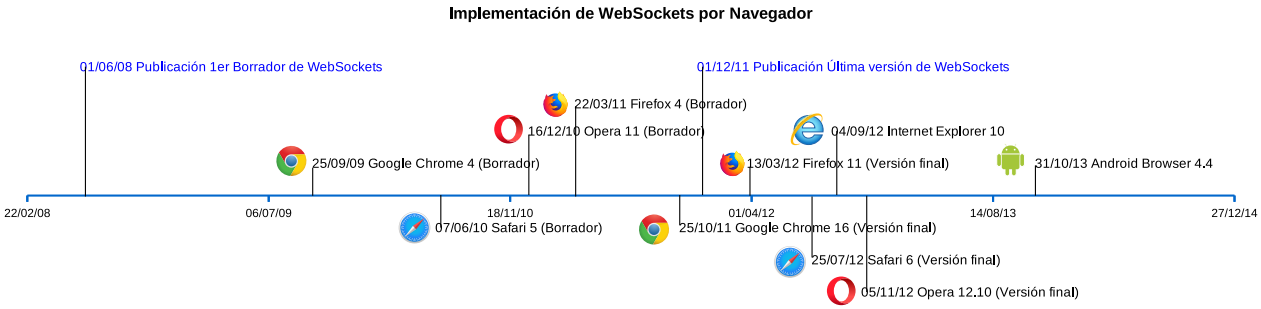
WebSocket apareció por primera vez en la especificación de HTML5 referenciado como TCPConnection. Fue en junio de 2008 cuando una serie de debates llevados a cabo por Michael Carter resultaron en la primera versión del protocolo que se conocería como WebSocket.

El nombre fue dado por Ian Hickson y Michael Carter, siendo el primero quien autorizó su inclusión en HTML5 y el segundo quien lo anunció en su blog.

En Diciembre de 2009, Google Chrome 4 fue el primer navegador que lo implementó por completo.

Después de que el protocolo fuese implementado y permitido por diversos navegadores, el RFC fue publicado en diciembre de 2011, dando por finalizada la implementación de WebSocket.

Aquí mostramos una linea de tiempo de la implementación de WebSockets en los distintos navegadores.



5. Utilidad de WebSockets

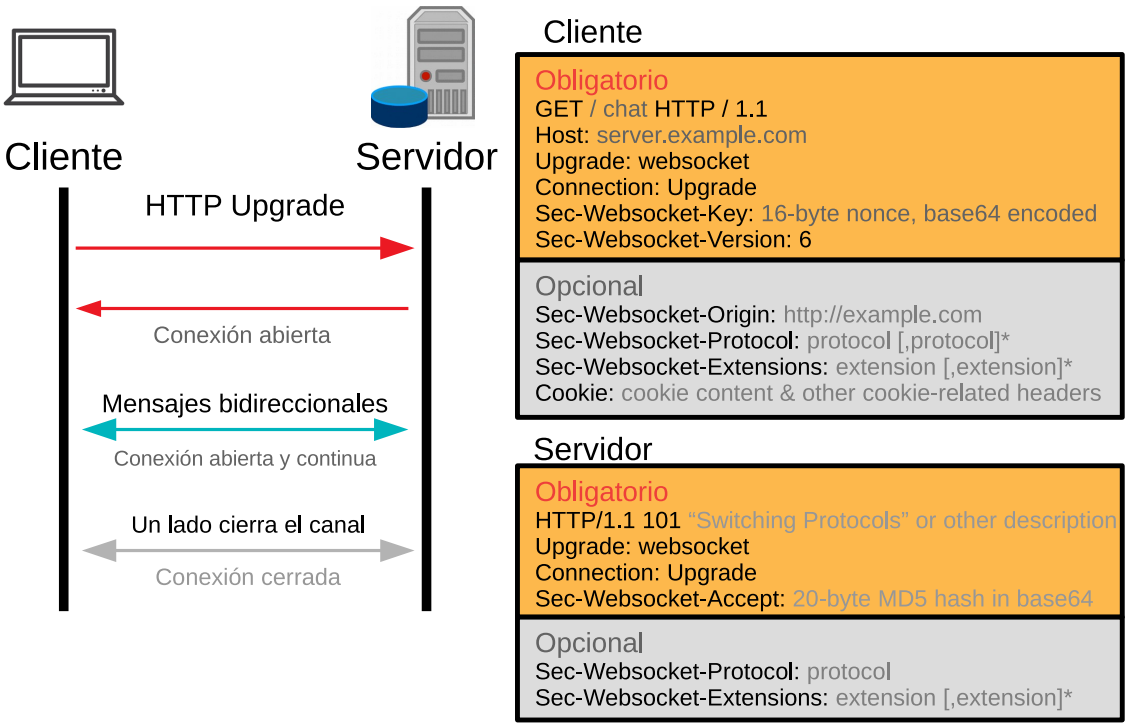
WebSockets tiene una gran cantidad de ventajas entre las que destacan:

- Comunicación full-duplex entre cliente y servidor.
- Mejora la eficiencia de la comunicación cliente-servidor, estableciendo una única conexión.
- Estandariza las comunicaciones evitando soluciones propietarias.
- API bien documentada y fácil de implementar e integrar en nuestras aplicaciones.
- Permite crear aplicaciones modernas con requerimientos de comunicación en tiempo real.

WebSockets se implementa de lado de cliente en JavaScript pero necesita soporte de lado de servidor. Una gran cantidad de lenguajes de programación permiten el uso de WebSockets, permitiendo crear servidores de aplicaciones en el lenguaje que queramos. Un ejemplo de estos lenguajes que soportan WebSockets son: C, JavaScript, Python, Ruby, Haskell, Java, PHP, Objetive-C, Swift...

6. Detalle técnico de una conversación servidor-cliente mediante WebSockets

Ahora presentamos el esquema de comunicación usado para iniciar una conexión mediante WebSockets:



En primer lugar, el cliente solicita conectarse a través de una llamada HTTP cuya cabecera sería la que vemos en la parte superior derecha. En ella distinguimos las partes más importantes:

- Host es el nombre del servidor al que estamos llamando.
- Upgrade indica que es una llamada de actualización, en este caso WebSocket.
- Conneccion define que es una llamada de actualización.
- Sec-Websocket-Key es una clave generada aleatoriamente que es posteriormente usada para autentificar la respuesta.
- Sec-Websocket-Origin muestra dónde se ha originado la petición. En el lado del servidor es usado para comprobar la autenticación de la petición.

Una vez que el servidor comprueba la autentificación, envía una respuesta que se ve como en la parte inferior derecha.

- Sec-Websocket-Accept tiene una clave que es decodificada y comparada con la key enviada para ver que la respuesta se está enviando al cliente correcto.

Una vez que la conexión está abierta, el cliente y el servidor pueden enviar los datos de uno a otro. Estos datos son enviados en forma de pequeños paquetes usando el protocolo TCP, por lo que no son visibles directamente dentro de la herramienta de desarrolladores.

Finalmente, tanto el cliente como el servidor pueden cerrar el canal de comunicaciones que comparten.

7. Implementación de una app usando WebSockets

Vamos a implementar una app de prueba muy simple tipo chat para poder demostrar el funcionamiento de los WebSockets. Para ello vamos a utilizar SocketIO junto con Flask. SocketIO es una librería multilenguaje que se utiliza para establecer conexiones entre cliente y servidor de forma bidireccional. Entre las múltiples formas de conexión que tiene se encuentran WebSockets. SocketIO está implementado en gran variedad de lenguajes de programación: desde JavaScript hasta Scala. Nosotros vamos a usar la implementación en Python para el framework de desarrollo web Flask.

Empezamos instalando los paquetes necesarios:

```
pip install flask-socketio eventlet
```

Ahora creamos las dos partes necesarias de nuestra aplicación: el código que se ejecuta en el cliente y el código que se ejecuta en el servidor. Para la parte del servidor usamos la facilidad de Flask para definir las rutas de nuestra aplicación.

```
#!/usr/bin/env python

from flask import Flask, render_template, session, request
from flask_socketio import SocketIO, emit, join_room, leave_room,
    \
    close_room, rooms, disconnect

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app, async_mode=async_mode)
thread = None
thread_lock = Lock()



@app.route('/')
def index():
    return render_template('index.html', async_mode=socketio.
        async_mode)

@socketio.on('connect', namespace='/test')
def test_connect():
```

```
    print("Cliente conectado")
    session['usuario'] = 'Sin nombre'
    emit('my_response', {'data': 'Conectado'})

@socketio.on('my_event', namespace='/test')
def test_message(message):
    emit('my_response',
         {'data': message['data']})

@socketio.on('my_broadcast_event', namespace='/test')
def test_broadcast_message(message):
    emit('my_response',
         {'data': session['usuario'] + ': ' + message['data']},
         broadcast=True)
```

Vemos que se reduce a inicializar Flask y SocketIO y definir rutas y eventos a los que responder. con el decorador siguiente definimos la función que se encarga de procesar los mensajes que lleguen por el WebSocket del tipo `my-event`.

```
@socketio.on('my_event', namespace='/test')
```

De forma muy similar implementamos en el lado del cliente (index.html) las conexiones pertinenentes de SocketIO esta vez en JavaScript.

```
<!DOCTYPE HTML>
<html>
<head>
    <title>WebSockets Demo</title>
    <script type="text/javascript" src="//code.jquery.com/jquery
       -1.4.2.min.js"></script>
    <script type="text/javascript" src="//cdnjs.cloudflare.com/
       ajax/libs/socket.io/1.3.5/socket.io.min.js"></script>
    <script type="text/javascript" charset="utf-8">
        $(document).ready(function() {

            namespace = '/test';

            var socket = io.connect(location.protocol + '//' +
               document.domain + ':' + location.port + namespace)
               ;

            socket.on('connect', function() {
                socket.emit('my_user', {data: 'Sin usuario'});
            });
```

```
            socket.on('disconnect', function() {
                socket.close();
            });

            socket.on('my_response', function(msg) {
                $('#log').append('<br>' + $('<div/>').text('
                    Recibido #' + msg.count + ': ' + msg.data).
                    html());
            });

            //Formulario
            $('form#connect').submit(function(event) {
                socket.emit('my_user', {data: $('#connect_data').
                    val()});
                return false;
            });

            $('form#emit').submit(function(event) {
                socket.emit('my_event', {data: $('#emit_data').
                    val()});
                return false;
            });
            $('form#broadcast').submit(function(event) {
                socket.emit('my_broadcast_event', {data: $('#
                    broadcast_data').val()});
                $('#broadcast_data').attr('value','')
                return false;
            });
            $('form#disconnect').submit(function(event) {
                socket.emit('disconnect_request');
                event.preventDefault();
                $('#conectar_button').attr('disabled',true)
                return false;
            });
        });
    </script>
</head>
<body>
    <h1>WebSockets con Flask y SocketIO</h1>
    <p>Conectado mediante: <b>{{ async_mode }}</b></p>
    <h2>Usuario: </h2>
    <form id="connect" method="POST" action='#'>
```

```
        <input type="text" name="connect_data" id="connect_data"
            placeholder="Tu nombre">
        <input type="submit" id="conectar_button" value="Enviar
            usuario">
    </form>
    <h2>Enviar:</h2>
    <form id="emit" method="POST" action='#'>
        <input type="text" name="emit_data" id="emit_data"
            placeholder="Mensaje">
        <input type="submit" value="Prueba Echo">
    </form>
    <form id="broadcast" method="POST" action='#'>
        <input type="text" name="broadcast_data" id="
            broadcast_data" placeholder="Mensaje">
        <input type="submit" value="Enviar a todos">
    </form>
    <form id="disconnect" method="POST" action="#">
        <input type="submit" value="Desconectar">
    </form>
    <h2>Recibidos:</h2>
    <div id="log"></div>
</body>
</html>
```

Aquí hacemos una cosa similar. Después de importar el código de SocketIO abrimos el socket que usará WebSockets contra el servidor y definimos las funciones que manejarán los mensajes recibidos a través del socket. Aquí se puede observar que la sintáxis de uso del WebSocket es muy similar en ambos lenguajes. Esto se debe a que la librería SocketIO estandariza bastante las funciones a usar.

El código de la aplicación anterior da lugar a la siguiente app:

Todo el código así como este documento se encuentran en https://github.com/jscoba/fr-websockets

Referencias

[1] Jesús Conde. *¿Qué son los WebSockets?* Mayo de 2015. URL: https://www.youtube.com/watch?v=SCe5A0lQzlM.

[2] Wikipedia contributors. *NPAPI — Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/wiki/NPAPI#LiveConnect.

[3] Miguel Grinberg. *Easy WebSockets with Flask and Gevent*. Feb. de 2014. URL: https://blog.miguelgrinberg.com/post/easy-websockets-with-flask-and-gevent.

[4] IETF. *The WebSocket Protocol*. Dic. de 2011. URL: https://tools.ietf.org/html/rfc6455.

[5] Kaazing.com. *websocket.org*. URL: https://www.websocket.org/.

[6] Phil Leggetter. *What came before WebSockets?* Ago. de 2011. URL: https://blog.pusher.com/what-came-before-websockets/.

[7] Stack Overflow. *WebSockets vs. Server-Sent events/EventSource*. URL: https://stackoverflow.com/questions/5195452/websockets-vs-server-sent-events-eventsource.

[8] Dylan Schiemann. *The forever-frame technique*. Nov. de 2007. URL: http://cometdaily.com/2007/11/05/the-forever-frame-technique/.

[9] Tanya. *5 Benefits Of WebSockets*. Abr. de 2015. URL: http://headerlabs.com/blog/5-benefits-of-websockets/.

[10] *WebSockets*. Dic. de 2016. URL: https://developer.mozilla.org/es/docs/WebSockets-840092-dup.

[11] Wikipedia. *WebSocket — Wikipedia, La enciclopedia libre*. 2018. URL: https://es.wikipedia.org/wiki/WebSocket.