### Integrantes:

### Luis Gustavo Saavedra veliz

### Jonathan Ramírez García

WEB SOCKETS

Taller de Programación

2016

Resumen

HTML5 como estándar incorpora Websockets entre sus nuevas características, para comprender esta tecnología en primer lugar vamos a exponer que intenta resolver Websockets y como se implementa la comunicación entre cliente y servidor hoy en día.

Explicaremos el concepto y las diferencias entre técnicas de polling y Websockets pero no las desarrollaremos específicamente de manera teórica o práctica.

COMUNICACIÓN CLIENTE - SERVIDOR

La problemática que intenta resolver websockets, generalizando, es comunicar los procesos del servidor con los procesos del cliente (navegador) en una plataforma web, mejorando las técnicas actuales. Para los que no conocen la plataforma de desarrollo web básica esta se compone generalmente de un lado cliente y un lado servidor, que interactúan mediante peticiones.

El servidor generalmente recibe las peticiones de los clientes, las procesa y envía posteriormente los resultados a través de la conexión que existe entre ellos.

En sus comienzos el modelo Cliente-Servidor servía las páginas de una manera sincrónica, lo que significa que por cada petición del cliente a través de un request HTTP, el cliente quedaba bloqueo hasta que el servidor daba una respuesta. En la actualidad este mecanismo de comunicación sigue siendo válido, pero no contribuye al dinamismo que los sistemas web requieren en la actualidad, por lo que desde hace varios años hizo su aparición una nueva técnica denominada AJAX, la cual mediante un nuevo tipo de request soportado por los navegadores permite llamadas asincrónicas desde el cliente al servidor.

Muchas veces es requerido que el servidor informe al cliente de eventos acontecidos sin que el cliente haya realizado un request para pedir información, para resolver este problema, otra tecnología fue incorporada: COMET.

Si bien el hecho de que el cliente no pide la información no es tan así, debido a que en realidad el cliente realiza un primer request y luego queda esperando en una conexión abierta que el servidor valla respondiendo en la medida que tiene información.

Este tipo de tecnologías permitieron comunicar los procesos del servidor de manera que las aplicaciones que corren en el servidor se comuniquen con el cliente y este último muestre la información procesada y actualizada en cierta parte del sitio web sin tener que recargarlo o sin que el cliente lo solicite explícitamente. Esto permitió dotar a la web de “aplicaciones web” más interactivas, es decir, sitios con tanta interacción asincrónica como quisiéramos.

Los Antecedentes

En Diciembre de 2009, apenas 8 meses después del primer borrador del W3C, Google Chrome se convertía en el [primer navegador en ofrecer soporte para WebSockets.](http://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html) No tardaría en seguirle Safari, mientras que la gente de Firefox, Opera e Internet Explorer prefirió mantener una postura más conservadora al respecto, esperando a que la especificación fuera más firme y clara. Sin ir más lejos, un año después de que Chrome ya implementara Firefox, la gente de Mozilla se planteaba introducirlo en Firefox, pero concluían que esperarían a tener una [versión más madura del futuro estándar](http://hacks.mozilla.org/2010/04/websockets-in-firefox/), puesto que aún se cernían sobre él números interrogantes.

Sin embargo, para Diciembre de este año tanto Firefox como Opera habían, finalmente, implementado WebSockets (desde la versión 10.70 en Opera y desde la Beta 7 de Firefox 4), desdiciéndose de algún modo de sus palabras previas. Sólo la gente de Internet Explorer mantenía su idea original: esperar a una versión más estable de la especificación.

Desde hace bastantes años aha sido la comunicación entre cliente y servidor utilizando el protocolo HTTP. Sin embargo, esta comunicación si bien es full dúplex y bidireccional, no permite que el servidor le envie contenido por cuenta propia al cliente. Esto ha sido “solucionado” en multiples oportunidades a través de apliaciones que simulan (desde el punto de vista del usuario y/o programador) una conexión única en la que el servidor puede enviar contenido de manera asincrónica al cliente.

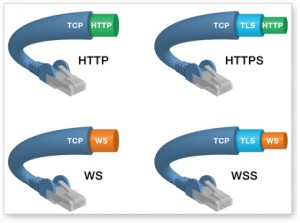
Fundamento Teórico

La especificación WebSockets define un API que establece conexiones “stocket” entre un navegador web y un servidor. Dicho con otras palabras: existe una conexión persistente entre el cliente y el servidor, y ambas partes pueden empezar a enviar datos en cualquier momento.

WebSockets es una nueva tecnología que, [en palabras del W3C](http://dev.w3.org/html5/websockets/), "permite a las aplicaciones web mantener una comunicación bidireccional con procesos en el lado del servidor". La primera aparición de un borrador de trabajo sobre esta tecnología data de Abril de 2009 y, aunque más de año y medio después sigue en el mismo estado, ha sido una tecnología que ha levantado grandes expectativas, hasta el punto de ser calificada como "el TCP de la Web". Esta tecnología venía a sustituir en buena a medida a XMLHttpRequest como forma de comunicación con el servidor, prometiendo simplificar el actual modelo.

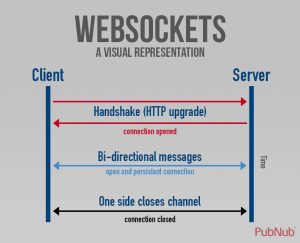
Entras las características que hacen interesante a WebSockets se encuentra la promesa de ofrecer una API simple para comunicar con el servidor así como la posibilidad de que estas comunicaciones sean bidireccionales, es decir, que puedan iniciarse en el lado del servidor; todo ello con una mejora en la latencia que actualmente soportan esta clase de comunicaciones.

El protocolo WebSocket permite la comunicación bidireccional entre un usuario agente que se ejecuta código no confiable que funciona en ambiente controlado a una host remoto que ha optado a las comunicaciones de ese código. El modelo de seguridad utilizado para ello es el modelo de seguridad basado en el origen comúnmente utilizada por los navegadores web. El protocolo consiste en un inicial apretón de manos seguido de encuadre mensaje básico, en capas a través de TCP. Es un protocolo nuevo para la web bajo TCP, por el cual, a diferencia de la conexión que venimos usando bajo HTTP, este es bi-direccional, que significa esto, hoy por hoy venís usando una conexión en una sola dirección, pedís al servidor y esperas la respuesta (o viceversa con Server Events). Pero con web sockets el servidor te habla también, te puede llamar y mandar un mensaje en cualquier momento.



Análisis de la Tecnología Websocket

La tecnología WebSocket consiste en una API que permite a las páginas web utilizar el protocolo web socket para establecer una vía de comunicación full dúplex (ida y vuelta de forma simultanea) con el servidor utilizando un mismo socket. Anteriormente a la aparición de esta tecnología, los programadores utilizaban diversos métodos para emular una conexión full dúplex con el servidor, sin embargo estos métodos provocaban un alto tráfico en la red, consumo de recursos del servidor y latencia.



VENTAJAS:

* Reduce el uso de la red, ya que evita la necesidad de utilizar los paquetes HTTP que contienen grandes cantidades de datos de cabecera, lo que recae, además, en un mayor trabajo de procesamiento.
* Se reduce la latencia en las conexiones ya que ponen menos carga en los servidores, y esto permite que estos equipos atiendan más conexiones simultaneas
* Permite atravesar firewalls y servidores proxy. En este caso si un WebSocket detecta la presencia de un servidor proxy, solicita una conexión TCP/IP usando una instrucción Connect HTTP, luego de lo cual se puede pasar por el proxy sin problemas
* Facilita una mayor escalabilidad en la Web debido a su eficiencia al momento de mantener conexiones persistentes con los servidores
* Es más rápido que el protocolo HTTP

DESVENTAJAS:

* Es una tecnología que continua en desarrollo, por lo que es común encontrar problemas de conectividad con algunos navegadores. Actualmente los principales navegadores que lo implementan son los siguientes: Chrome, Safari, Firefox, entre otros.
* En el 2010 se encontraron serias vulnerabilidades en el protocolo de WebSocket, lo que obligó a muchos navegadores a deshabilitar su implementación.

Caso Demostrativo

Esta semana hemos conocido que Adam Barth y otros colegas han publicado un documento demostrando que el [protocolo sobre el que funciona WebSockets es vulnerable.](http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html) Esta vulnerabilidad afecta a todas las implementaciones de WebSockets, incluidas las que hicieron Java y Flash de esta nueva API, y que algunos desarrolladores han venido utilizando como sustitutivo en los navegadores que aún no la soportaban.

Esta vulnerabilidad es lo suficientemente grave como para haber llevado al [equipo de Firefox](http://hacks.mozilla.org/2010/12/websockets-disabled-in-firefox-4/) y [Opera](http://annevankesteren.nl/2010/12/websocket-protocol-vulnerability) a deshabilitar su implementación sobre WebSockets. La gente de Chrome y Safari aún no se habían pronunciado al respecto al momento de redactarse este artículo, aunque dada la gravedad del asunto no se esperaba otra reacción diferente que la que han tomado otros fabricantes.

Utiliza WebSocket siempre que necesites una conexión casi a tiempo real y de latencia baja entre el cliente y el servidor. Ten en cuenta que esto podría significar tener que replantearte cómo has desarrollado tus aplicaciones de servidor, adoptando un nuevo enfoque en tecnologías como las colas de eventos. Estos son algunos ejemplos de casos prácticos:

* Juegos online multijugadores
* Aplicaciones de chat
* Rotativos de información deportiva
* Actualizaciones en tiempo real de las actividades de tus amigos

El Problema

El problema no es sólo el hecho de que WebSockets se haya demostrado gravemente vulnerable. La gran dificultad aquí se la van a encontrar los muchos desarrolladores que confiaron en que WebSockets era un estándar maduro, una tecnología sobre la que podían empezar a construir sus nuevas aplicaciones.

Estos desarrolladores se encontrarán, a partir de esta semana, con que casi ningún navegador va a seguir ofreciendo soporte a WebSockets, con lo que deberán reprogramar sus aplicaciones web. Aquellos que hayan utilizado la detección de características y se hayan apoyado en Java y Flash en los casos en que el navegador no implementara la tecnología tampoco habrán resuelto el problema puesto que estarán confiando en una implementación igual de insegura que la que sufren los navegadores, y que no sería de extrañar que también termine siendo desactivada en breve.

Algunos han decidido tomárselo con humor, haciendo circular por la Red un [vídeo de humor](http://www.xtranormal.com/watch/7991991/) al respecto de lo que está pasando con WebSockets. Otros, probablemente, no se lo han tomado con tanta filosofía.

La Discusión

La situación que se ha creado en torno a WebSockets ha merecido una reflexión en algunos casos y [una crítica en otros](http://blogs.msdn.com/b/giorgio/archive/2010/12/08/is-chrome-4-the-next-ie6.aspx), dada su gravedad. Las primeras críticas no son actuales, se remontan al punto en el tiempo en que Chrome y Safari ya implementaban WebSockets, y Firefox y Opera estaban en proceso de hacerlo. Estas críticas afectaron a IE9 y su equipo de desarrollo y venían a señalar que el navegador de Microsoft era el menos respetuoso con los estándares. Todo lo ocurrido en el pasado con IE6 ha impedido a muchos ver que, con el desarrollo de IE9, Microsoft está siendo tanto o más respetuosa con los estándares que sus rivales, hasta el punto de haber sacado la [mejor nota en los recientes test sobre HTML5](http://test.w3.org/html/tests/reporting/report.htm) que la W3C publicó hace unas semanas.

Quizás para poder comprender mejor porqué se ha llegado a este punto, deberíamos re-examinar el concepto de estándar. En un artículo anterior desgranamos los [pasos que se tienen que seguir para que una especificación se convierta en Recomendación](http://www.desarrolloweb.com/articulos/proceso-estandarizacion-w3c.html)por parte del W3C, es decir, el estado de estándar. Obviamente los fabricantes de navegadores no pueden esperar a este estadio último para empezar a implementar la especificación en su software, pues de lo contrario tampoco se alcanzaría nunca el nivel de Recomendación, dado que es necesaria la existencia previa de dos implementaciones interoperables entre sí.

Sin embargo, muchos fabricantes de navegadores se han lanzado últimamente a una carrera por ser el que "más soporta los estándares" que les ha llevado a implementar especificaciones que, como WebSockets, no era más que un borrador de trabajo susceptible de todo tipo de modificaciones. La propia implementación que de WebSockets ha hecho Chrome ya [había sufrido un cambio muy importante meses atrás](http://blog.chromium.org/2010/06/websocket-protocol-updated.html), consecuencia de una de las revisiones que de WebSockets estaba haciendo su equipo de trabajo.

Y es precisamente balancear entre la prontitud a la hora de implementar una especificación susceptible de convertirse en estándar y la estabilidad suficiente para no hacer la vida más difícil a los desarrolladores de aplicaciones web, lo que debemos exigir a los fabricantes de navegadores. Nosotros no debemos ser, de nuevo, las víctimas de un nuevo escenario de "guerra de navegadores" en la que la amenaza de "no ser compatible" se convierta en arma arrojadiza entre Apple, Google, Microsoft, Mozilla y Opera.

La Conclusión

* Conclusiones Generales. Esta especificación tiene muchos beneficios ya que, al momento de utilizar una conexión “socket” puede haber más interactividad en tiempo real con un cliente y el servidor, así mismo crear un WebSocket es fácil comparado con otras maneras de crear conexiones.
* Contratiempos presentados al realizar la actividad. No se encontraron ejercicios entendibles para realizarlos y probarlos.
* Opinión acerca de la dificultad de la actividad y el tiempo y formato de entrega. Se podría explicar ejemplos ya hechos para hacer el tema más entendible.
* Tiempo aproximado de realización (en horas). Aproximadamente se realizó en 24 horas.

En el caso concreto de WebSockets, es demasiado tarde para una solución. Pero podemos aprender para el futuro que, antes de empezar a utilizar una nueva API basada en un futuro estándar, no debemos dejarnos llevar por la parafernalia marketinera. En su lugar, debemos investigar directamente del W3C o el organismo estandarizado que corresponda, en qué estado de madurez se encuentra la especificación y evaluar si la ventaja que supone su uso merece la pena el riesgo de cambios sin compatibilidad hacia atrás o que, como en un caso grave como el de WebSockets, lleven a la desactivación total de la funcionalidad en los navegadores.

Otra posible solución es el ejercicio de su responsabilidad por parte de los fabricantes de navegadores, como generador de un soporte de plataforma como son, a la hora de implementar nuevas características. En concreto algunas voces piden que, cuando un fabricante decida "arriesgarse" e implementar una especificación que no está cerca de ser estable, lo haga [bajo su propio etiquetado](http://blogs.msdn.com/b/ie/archive/2010/11/15/ie9-vendor-prefixes-and-developers.aspx). Es decir, que las APIs correspondientes lleven un prefijo que las identifiquen unívocamente como propias del vendedor, al no estar respaldadas por ningún estándar. En Microsoft, por ejemplo, han seguido esta aproximación a la hora de implementar la API que sirve para medir los [tiempos de carga de una aplicación web](http://blogs.msdn.com/b/ie/archive/2010/06/28/measuring-web-page-performance.aspx).

## Bibliografía:

* http://www.arkaitzgarro.com/html5/capitulo-13.html
* https://www.websocket.org/
* http://es.wikipedia.org/wiki/WebSocket
* <http://blogs.msdn.com/b/giorgio/archive/2010/12/08/is-chrome-4-the-next-ie6.aspx>
* <http://hacks.mozilla.org/category/websocket/>
* <http://dev.w3.org/html5/websockets/>
* <http://blog.chromium.org/search/label/websockets>
* <http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html>
* <http://annevankesteren.nl/2010/10/websocket-protocol>
* <http://annevankesteren.nl/2010/12/websocket-protocol-vulnerability>
* <http://test.w3.org/html/tests/reporting/report.htm>