Desarrollo e implementación de un prototipo para una plataforma tecnológica para la transmisión de texto y video (streaming) en tiempo real empleando tecnología websocket

Development and implementation of a prototype for a technological platform for the transmission of text and video (streaming) in real time using technology websocket

Alejandro Rodas Vásquez
Universidad Tecnológica de Pereira
alejorodasvasquez@utp.edu.co
Colombia

Alexander Valencia Carrasquilla
Universidad Tecnológica de Pereira alexavalencia@utp.edu.co
Colombia

Tipo de Artículo: Investigación científica y tecnológica**. Recibido:** 24/10/2017**. Aprobado:** 30/04/2018

Resumen. El presente documento describe el proceso de construcción de una plataforma tecnológica enfocada a prestar un servicio de comunicación en tiempo real utilizando Websocket. Primero se realiza la descripción de las tecnologías Polling, Long Polling y Streaming con el fin de contextualizar los enfoques que anteriormente se empleaban para lograr simular una comunicación full-duplex. Así mismo, se hace una reseña de las funcionalidades de los protocolos RTP (Real-time Transport Protocol), RTCP (Real-Time Control Protocol) y RTSP (Real-Time Streaming Protocol) con el fin de realizar un contraste con el empleo de la tecnología WebSockect. Posteriormente, se hace una definición de lo que es la tecnología Websocket y su integración con HTML5. Por último, se muestra el desarrollo de la plataforma y las herramientas software que se emplearon.

***Abstract**. This document describes the process of construction of a technological platform focused on providing a real-time communication service using Websocket. First, the description of Polling, Long Polling and Streaming technologies is made in order to contextualize the approaches previously used to simulate a full-duplex communication. Likewise, a review is made of the RTP (Real-time Transport Protocol), RTCP (Real-Time Control Protocol) and RTSP (Real-Time Streaming Protocol) protocols in order to make a contrast with the use of WebSockect technology. Subsequently, a definition of what is Websocket technology and its integration with HTML5 is made. Finally, it shows the development of the platform and the software tools that were used.*

Palabras clave. Comunicaciones; HTML5; Streaming; Tiempo real; TCP; Web; WebSockets.

***Keywords**. Communications; HTML5, Long Polling; Streaming; Polling; Real time; TCP; Web; WebSockets*

***DOI:** 10.21500/20275846.3277*

1. Introducción

Con la evolución de Internet, el significativo incremento del ancho de banda y el aumento de la capacidad del procesamiento de los ordenadores, la demanda y el consumo de aplicaciones que permitan al usuario acceder a la información en el momento (tiempo real) está forzando el desarrollo de tecnologías que soporten estas demandas. Anteriormente, visitar una página web significaba obtener un contenido en texto y algunas imágenes. En la actualidad, la construcción de una página o una aplicación web comprende la integración de varios lenguajes: tales como: en el *back-end* (Ruby, PHP, Python) y el *front-end* (HMTL, CSS, JavaScript), donde el surgimiento de AJAX brinda funcionalidades dinámicas y se logra empezar a realizar peticiones HTTP de manera asíncrona, permitiendo el surgimiento de tecnologías como *Polling, Long Polling* y *Streaming (Bajo Demanda* y *Video Streaming)* para la construcción de aplicaciones para el consumo de recursos en Tiempo Real. Un ejemplo de ello son las transmisiones de video en vivo y los chats. Por lo tanto, el desarrollo de esta plataforma de transmisión nace con la concepción que integre un módulo que haga parte de una aplicación web orientada a facilitar las labores de gestión realizadas por el programa denominado UTEpitos, ejecutado por la Universidad Tecnológica de Pereira.

UTEpitos [1] busca evitar la deserción escolar de las madres cabeza de hogar que ingresan a la Universidad Tecnológica de Pereira (UTP) a estudiar, mediante la implementación de un jardín infantil para dejar sus hijos al cuidado de un grupo experto de profesionales (docentes profesionales en pedagogía infantil, psicólogos y nutricionistas, entre otros). La plataforma

propuesta busca facilitar la labor de estos expertos, proporcionando una tecnología para interactuar con los usuarios.

El proyecto cuenta con una fase de recopilación de información, enfocada en la búsqueda tecnológica y de otros trabajos realizados en el campo de la transmisión en *Streaming.* Además, con la identificación de cómo son aplicados en la solución de problemas específicos. Posteriormente, se diseña una plataforma Web utilizando un servidor *Streaming* con el fin de publicar material educativo y radial de la propia Universidad Cooperativa de Colombia (UCC) sede Barrancabermeja [2] . Luego, en la Universidad Sergio Arboleda se implementa una arquitectura que permita brindar el servicio radio *Streaming* en la red de la Universidad integrando la movilidad, calidad de servicio (QoS), Calidad De Experiencia (QoE) y la adopción de IP Multimedia Subsystem (IMS) [3]. Seguidamente, se crear una red social [4] donde con acceso exclusivo a las funcionalidades básicas, tales como: compartir como texto, imágenes, videos, audio, enlaces a otros websites, e integrar servicios *Live Streaming* para mejorar la interacción entre los usuarios de la red. Finalmente, se plantea una propuesta para mejorar la interacción comunicativa, poniendo a prueba la plataforma mediante un caso de uso en una empresa privada [5].

Como se puede evidenciar durante este análisis bibliográfico, los proyectos propuestos poseen cierto grado de complejidad técnica relacionados con la tecnología de transmisión *Live Streaming.* Por ello, en el presente proyecto se decide buscar otro enfoque aprovechando HTML5 y WebSocket. Con la llegada de HTML5 se implementan mejoras que facilitan el desarrollo de aplicaciones web. La integración de HTML5 WebSocket permite [6] definir un canal de comunicación full-duplex que opera por un único socket sobre la web. El Websocket es una mejora incremental a las comunicaciones HTTP convencionales y representa un avance para las aplicaciones web en tiempo real. Lo anterior, debido a diferencia de las tecnologías enfocadas en la transmisión por *Streaming*, al emplear WebSocket [7] no se necesitan software de terceros (*browser plug-in*) para recibir y procesar los datos transmitidos desde el servidor, facilitando el desarrollo de la aplicación web y el uso por parte del usuario final. Además, permitiendo crear una plataforma funcional independiente del sistema operativo que se tenga instalado en el ordenador, reduciendo complicaciones a controlar.

Por lo anterior expuesto, esta es la tecnología seleccionada para el desarrollo de la presente plataforma. Para su construcción se emplean herramientas de uso *open source* y gratuitas (a excepción de *Dreamweaver* que pueden ser reemplazado por opciones como Notepad++[1], Brackets[2] o Kompozer[3]).

2. Tecnologías tradicionales para la comunicación web en tiempo real

Como es conocido, el protocolo HTTP es usado por la World-Wide Web desde 1990, y se caracteriza por ser un protocolo a nivel de aplicación. Su funcionamiento radica en la interacción entre un cliente y servidor a través de peticiones y respuestas. Normalmente [6], cuando un navegador visita una página web, se envía una solicitud HTTP al servidor que aloja esa página. El servidor web reconoce esta solicitud y envía la respuesta.

Este procedimiento permite establecer una conexión entre estos dos componentes. Sin embargo, dicha conexión se interrumpe una vez que el servidor a retornado el recurso solicitado, quedando a la espera de una nueva solicitud.

Este comportamiento es poco óptimo cuando se pretende construir aplicaciones que presenten información en tiempo real, debido a que para lograr obtener una información actualizada se requiere refrescar la página manualmente. Para abordar este problema fueron desarrolladas soluciones de comunicación web en tiempo real, tales como “Polling”, “Long Polling” y “Streaming”

2.1 Polling

Con [8] este tipo de conexión el navegador envía peticiones HTTP a intervalos regulares e inmediatamente recibe una respuesta.

Dichas peticiones se realizan de manera asíncrona (Figura. 1), de tal manera que, cuando se produce un cambio en la información del servidor, el *script* del lado del cliente procesa y despliega la información retornada.

```
setInterval(function(){
    $.ajax({ url: '/my/page', success: function(data){
        // do something with the data
    } });
}, 5000);
```

Figura 1. Ejemplo de un script Polling. Elaboración propia

Sin embargo, [9] existen deficiencias en esta solución, debido a que es difícil conocer la frecuencia de actualización de datos, por lo que el navegador no puede obtener los últimos datos en el tiempo. Por otro lado, en el caso en que no exista información para actualizar en los intervalos de tiempo en que el cliente realiza la petición generará tráfico de red innecesario.

[1] https://notepad-plus-plus.org

[2] http://brackets.io/

[3] http://kompozer.net/

2.2 *Long Polling o Comet*

En este tipo de conexión el servidor y el cliente mantienen una conexión abierta en un lapso de tiempo predefinido. Cuando [8] se recibe una notificación dentro de ese período, se envía al cliente una respuesta que contiene el mensaje. Cuando no se recibe una notificación dentro del período de tiempo establecido, el servidor envía una respuesta para finalizar la solicitud abierta. No obstante, como se menciona en [9] cuando ocurre un gran número de concurrencia (peticiones), la memoria del servidor y la capacidad de computación se consumirán en gran medida al mantener esas conexiones HTTP activas. En la Figura 2 se puede observar la implementación de *Long Polling* a través de AJAX.

```
function load(){
    $.ajax({ url: '/my/page', success: function(){
        // do something with the data
    }, complete: load, timeout: 20000 });
}
```

Figura 2. Ejemplo de un script Long Polling. Elaboración propia

3. Transmisión web en tiempo real *streaming* y *websockets*

Hasta el momento se ha hecho referencia a los conceptos de *Polling* y *Long Polling* y cómo estas tecnologías lograron en cierta medida simular una comunicación en tiempo real sobre el protocolo HTTP. No obstante, para poder definir que es *Streaming* y *WebSocket* es importante aclarar el concepto que encierra la Comunicación en Tiempo Real.

El [10] término Tiempo Real (*real-time*) se refiere a la relación entre la ocurrencia de un evento y la capacidad de reaccionar a este (…) En última instancia, el evento debe ser entregado en un tiempo suficientemente corto para que este siga siendo relevante; para que todavía tenga significado dentro del contexto que aplica.

De esta manera, en [11] 1992 aparece la tecnología *HTTP Streaming* como una creación de *Netscape* bajo el nombre 'documento dinámico' (*dynamic document*). *HTTP Streaming* [12] es similar al *long-polling* excepto que la conexión no se cierra con la existencia de nuevos datos disponibles o en un intervalo determinado. En cambio, estos datos se envían a la conexión existente que permanece abierta, esto significa que el servidor podrá seguir enviando fragmentos de datos al cliente.

Así mismo, [13] es importante que la conexión permanezca abierta, ya que, el protocolo HTTP solo permite realizar peticiones desde el lado del cliente. Por lo tanto, cuando se produce un cambio de estado en la información (dato), no hay forma de que el servidor pueda abrir las conexiones necesarias para los clientes interesados. Ahora, cuando se hace referencia a tipos o métodos de transmisión en *Streaming* se tienen dos categorías: Streaming *Bajo Demanda* (On-Demand) y Video Streaming (Live Streaming).

3.1 *Streaming* Bajo Demanda(*On-Demand*)

Este se tipo de *Streaming* [14] se caracteriza porque el usuario descarga el archivo de video completo y luego reproduce. Sin embargo, la transferencia completa de archivos en el modo de descarga generalmente sufre un tiempo de transferencia largo y quizás inaceptable.

3.2 Video *Streaming* (*Live Streaming*)

En contraste, en *Live Streaming* [15] no es necesario la descarga del contenido de video en su totalidad. Por el contrario, este se va reproduciendo mientras se recibe y decodifican las partes del contenido. Una analogía que se emplea para comprender esta categoría de *Streaming* es el servicio por televisión, donde en ella se puede consumir el recurso mientras se transmite y al finalizar el programa televisivo este ya no se encontrará disponible para el usuario.

Ahora, [16] es importante tener en cuenta que tanto para el *Streaming* Bajo Demanda como para *Video Streaming,* los datos deben ser codificados en un formato especial que permita reducir su tamaño permitiendo a su vez que estos sean manejables. De manera que el servidor al entregar los datos al cliente, este logre representarlos y mostrar la información solicitada, ya sea visual o audio.

Algunas de las tecnologías de compresión de videos (también llamadas *codecs*) son: MPEG-4, VP6 y VP8, *QuickTime, Windows Media Video* (WMV) y MPEG-1/2. Del mismo modo, según [17] el término “codec” define al software, o programa, utilizado para codificar y decodificar la señal de video o flujos. Su función es la de comprimir y descomprimir los datos para almacenarlos y transmitirlos en archivos más pequeños. Así mismo, señala que en este grupo se encuentran diversas opciones en el mercado pudiendo volver su elección un verdadero dolor de cabeza para el usuario, lo que significa que no existe un estándar definido. De igual forma, por el lado de audio se tiene los formatos de compresión AAC (*Advanced Audio Coding*), *Vorbis*, *Windows Media Audio* (WMA), y MP3. Por otro lado, *Live Streaming* (Streaming, 2010) también requiere de ciertos componentes que definen su arquitectura, los cuales son: el componente de servidor (*server component*), componente de distribución (*distribution component*), y el software del cliente (*client software*).

- **Componente de servidor***:* es responsable de tomar las transmisiones (*streams*) de entrada de datos (media) y codificarlos digitalmente, encapsulándolos en un formato adecuado para la entrega y distribución.
- **Componente de distribución***:* consiste en servidores web estándar. Son responsables de aceptar las solicitudes de los clientes y entregar los datos (*media*) preparados y los recursos asociados al cliente.
- **Software del cliente***:* es el responsable de determinar los medios apropiados para solicitar y descargar los recursos, posteriormente debe estructurarlos de nuevo para que los datos (*media*)

se logren presentar al usuario en un flujo continuo. Esto significa que el cliente debe tener instalados los *plugins* necesarios para decodificar el audio y video recibidos, como se mencionó anteriormente.

Por otro lado, desde la perspectiva de desarrollo de software, el framework de QuickTime [4] se puede emplear para la creación de aplicaciones de escritorio que permiten reproducir HTTP *Live Streams*.

Con relación a la transmisión en *Live Streaming* es necesario la integración de protocolos, tales como RTP (*Real-time Transport Protocol*), RTCP (*Real-Time Control Protocol*) y RTSP(*Real-Time Streaming Protocol*). Aunque la definición de cada uno de estos protocolos es extensa, en [17] se presenta una conceptualización compacta de ellos, tal como se presenta a continuación.

Estos protocolos pertenecen a la capa de aplicación del modelo TCP/IP, al igual que el protocolo HTTP, donde el RTP es utilizado para el envío de audio y video sobre redes IP diseñado para transferencia de *streaming* multimedia en tiempo real y de extremo a extremo.

De igual manera, RTCP y RTSP pueden emplearse de manera conjunta, en el cual RTP se encarga de los flujos multimedia (audio y video), RTCP se utiliza para sincronizar los cuando tenemos flujos simultáneos, monitorear las estadísticas de la transmisión y calidad de servicio (QoS).

Así mismo, RTSP (*Real Time Streaming Protocol*) permite controlar el flujo en los servidores multimedia. Se usa para el establecimiento y control de las sesiones multimedia entre ambos extremos, utiliza RTP como protocolo base para la entrega de datos.

Por otro lado, en el trabajo presentado en [18] se encuentra que revelan conclusiones interesantes sobre la compatibilidad de RTSP. Allí señalan que uno de los inconvenientes que presenta RTSP consiste en el empleo del puerto 554, el cual es bloqueado frecuentemente por los muros de fuego (*firewalls*). Sin embargo, algunos servidores RTSP ofrecen la funcionalidad de crear un túnel que utilice el mismo puerto que HTTP (puerto 80).

Por consiguiente, esto significa que, aunque exista una arquitectura definida dentro de la tecnología *Live Streaming*, esta requiere de especificaciones precisas al momento de crear una aplicación orientada al usuario. Por un lado, cuando se hace referencia al componente *Software del cliente*, el usuario de la aplicación debe tener instalado dentro de su ordenador el software especializado (*plugins o codecs*) para la decodificación de los datos transmitidos (tal como se mencionó anteriormente), presentando esto un problema al momento de hacer uso de la aplicación ya que se debe tener en cuenta que no todos los usuarios poseen conocimientos en informática y no podrán resolver este problema por sí mismos.

Del mismo modo, dentro de la perspectiva de desarrollado se deben considerar varios elementos y circunstancias relacionados con los protocolos de transmisión tal y como se acaba de hacer referencia. Puesto que al no tener un protocolo estandarizado (tal como lo tiene HTTP) al desarrollar aplicaciones web se puede incurrir en problemas de compatibilidad.

Por tal motivo, la tecnología *WebSocket* se presenta como una alternativa en la construcción de Aplicaciones Web En Tiempo Real.

3.3 WebSocket

Una de las principales ventajas que tiene *WebSocket* es el soporte en escabilidad, ya que [10] utiliza una única conexión TCP para la comunicación entre el servidor y el cliente en lugar de múltiples conexiones. Así mismo, el protocolo *WebSocket* al hacer parte de HTML5 (como se hablará más adelante en el texto) puede ser utilizado por los navegadores web (Chrome, Firefox, Opera, entre otros) lo cual es una ventaja al desarrollar aplicaciones web.

Así mismo, como hace se referencia en [19] al emplear *WebSocket* este permite construir aplicaciones web que posean un canal orientado a la comunicación del tipo *full-duplex* con un comportamiento asíncrono. Permitiendo obtener actualizaciones en tiempo real (*real-time*) en la transmisión, lo cual en el pasado se lograría a través de *long polling* y otras técnicas que ya se han tratado en el presente artículo. También es importante señalar que una las razones por las que *WebSockets* [10] es exitoso es que el establecimiento de esta comunicación bidireccinal entre el cliente y el servidor se hace a través de una única conexión.

Igualmente, a diferencia de la tecnología *Live Streaming*, *WebSocket* no necesita de un software o aplicación especial del lado del cliente ya que su desarrollo e implementación es realiza en *JavaScript* y posee su propio API, tal como se muestra en el apartado 5 del presente artículo.

4. HTML5 y WEBSOCKET

Generalmente cuando se hace referencia a HTML5, su característica más notoria es propiamente los elementos que componen el código HTML (como son las nuevas etiquetas o marcadores de esta versión). No obstante, las aplicaciones Web actuales emplean un conjunto de tecnologías interconectadas que es imposible crear contenidos dejando a un lado alguna de ellas. Por ejemplo, si se desea crear una simple página HTML se requiere CSS para dar una mejor apariencia. Por otro lado, si lo que se desea es crear una aplicación con la cual el usuario pueda interactuar y recibir feedback[4] es necesario integrar HTML, CSS y

4 Retroalimentación

JavaScript, pudiendo aprovechar de esta forma las nuevas funcionalidades de HTML5.

Por consiguiente, HTML5 [20] es un término general que describe un conjunto de tecnologías relacionadas. Por otro lado, dentro de las nuevas tecnologías que enmarcan a HTML5 se encuentra el componente de Conectividad [21] que posee: *WebMessaging*, *WebSocket* y *WebWorkers* (Figura 4).

Así pues, como se define en [22] *WebSocket* es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP (Figura 3). Está diseñada para ser implementada en navegadores y servidores web, pero puede ser usado por cualquier aplicación cliente/servidor. La API de *WebSocket* está siendo normalizada por el W3C, mientras que el protocolo *WebSocket* ya fue normalizado por la IETF como el RFC 6455. Debido a que las conexiones TCP comunes sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporcionaría una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios *WebSocket* sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo).

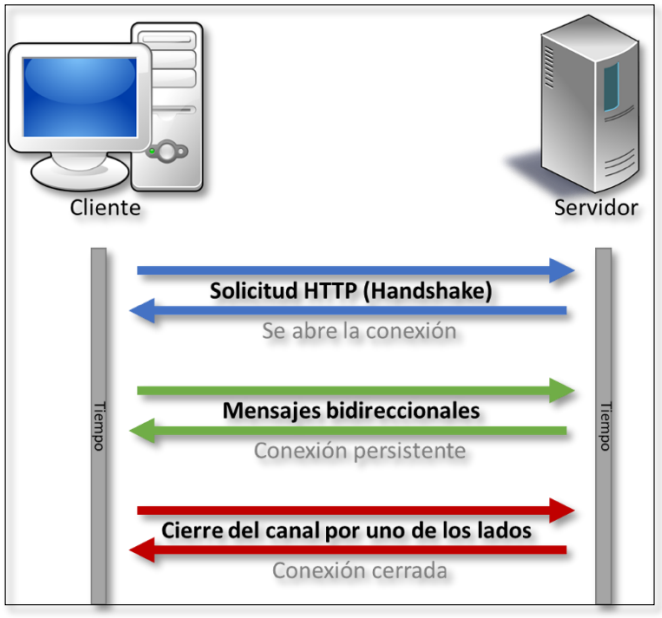

Figura 3. Comunicación *WebSocket.* Elaboración propia)

WebSockets es una tecnología basada en el protocolo ws, este hace posible establecer una conexión continua full-duplex, entre un cliente y servidor. Como se especifica en [23], un cliente *websocket* podría ser el navegador del usuario, pero el protocolo es una plataforma independiente.

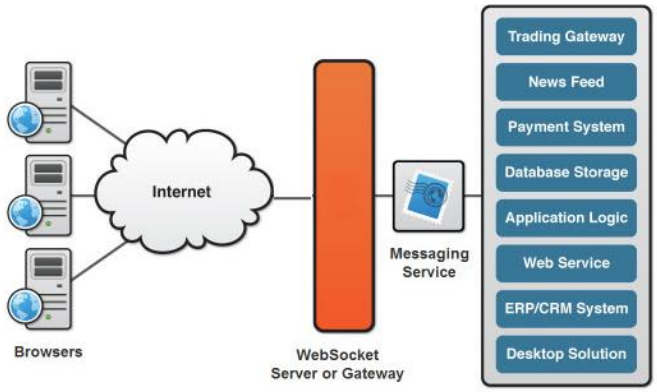

Figura 4. Servicios implementados en Websocket. Tomado de websocket.org

Una de las características que proveen los *WebSockets* [7] , es su habilidad para atravesar firewalls y *proxies*, un área problemática para muchas aplicaciones. Donde estas suelen emplear consultas de los estados de red de larga duración como una línea de defensa rudimentaria contra *firewalls* y *proxies*. Esta técnica es eficaz, pero no es adecuada para aplicaciones que tienen una latencia de 500 milisegundos o un alto rendimiento. Así mismo, las tecnologías basadas en complementos, como Adobe Flash, también proporcionan cierto nivel de soporte de socket, pero han sido durante mucho tiempo agobiados por los problemas de proxy y de cortafuegos.

Por otro lado, el *WebSocket* detecta la presencia de un servidor proxy y configura automáticamente un túnel para pasar a través de este. Dicho túnel se establece mediante la emisión de una instrucción HTTP CONNECT al servidor proxy, que solicita que el servidor proxy abra una conexión TCP / IP a un host y puerto específicos. Una vez que el túnel está configurado, la comunicación puede fluir sin obstáculos a través del proxy. Dado que HTTP / S funciona de manera similar, los *WebSockets* seguros a través de SSL pueden aprovechar la misma técnica HTTP CONNECT. Tenga en cuenta que los *WebSockets* son compatible con navegadores modernos, un ejemplo de ello es Chrome, el cual ahora posee *WebSockets* de forma nativa.

5. Construcción de la plataforma empleando tecnología *websocket*

Para la construcción de la plataforma fue necesario el uso de las siguientes herramientas:

- **Dreamweaver:** Usado para la edición y desarrollo en HTML, JavaScript, PHP y CSS.
- **Jquery 3.1.1:** Librería en Java, principalmente para algunas tareas que se ejecutan en el lado cliente.
- **Navegador Mozilla Firefox:** Se toma este navegador para las pruebas ya que ofrece menos restricciones al momento de usar características como la cámara del usuario en conexiones que no sean HTTPS.
- **Amazon AWS:** Es necesaria una máquina virtual que permita fácilmente la administración de puertos a las

necesidades del desarrollador, se llegó a la conclusión que AWS se ajusta perfectamente al desarrollo del proyecto.

- **Webmin** [24]**:** Software que permite administrar fácilmente y de manera gráfica, características y propiedades del servidor.
- **php-websockets**: Script encontrado en [25] que permite rápidamente la implementación de un servidor WebSocket en PHP.

5.1 Implementación del servidor

Para la implementación del servidor virtual se optó por el uso de Amazon AWS [26]. No obstante, este fácilmente se puede sustituir por cualquier máquina virtual o física en Linux, no se recomienda el uso de servicios hosting ya que este tipo de sistemas por lo general poseen numerosas restricciones en cuanto al uso de puertos no estándar.

La máquina a usar en AWS es una instancia básica (t2.micro) la cual se carga y configura el paquete XAMPP que contiene todo lo necesario el servidor web (PHP, Apache, etc). Para evitar el uso de IP se crea el siguiente subdominio *aws.tumundoenlared.com*, el cual apunta a una máquina virtual.

Seguido a esto, se procede con la instalación de Webmin, el cual permite administrar el servidor desde un navegador web, así como también cargar y actualizar archivos en el sistema.

Usando la consola de administración de AWS abrimos los puertos 80-443 para permitir el tráfico web, el puerto 10000 para Webmin, los puertos 9000-9001 para usarlos en el servicio de websockets y el puerto 22 para realizar conexiones SSH al servidor en caso de ser necesario.

Con base en la documentación encontrada en [27], se procede a usar el script **Server.php** (Figura 5) que se encargará de correr el servicio y de crear los sockets necesarios para la comunicación. Este *Script* posee dos variables parametrizables, **$host** y **$port**:

```php
<?php
$host = 'localhost'; //host
$port = '9000'; //port
$null = NULL; //null var

//Create TCP/IP sream socket
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
//reuseable port
socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR, 1);

//bind socket to specified host
socket_bind($socket, 0, $port);

//listen to port
socket_listen($socket);

//create & add listning socket to the list
$clients = array($socket);
```

Figura 5. Parametrización del servidor en Server.php. Elaboración propia

En la Figura 5 se observa la parametrización del servicio, el cual estará disponible en **localhost** y será accesible por el puerto **9000**.

5.2 Chat con websocket

Una vez puesto en marcha el servidor se debe crear el módulo HTML que se ejecutará del lado cliente, primero se debe crear un objeto tipo WebSocket con JavaScript (Figura 6)

```javascript
$(document).ready(function(){
    //Se crea un objeto WebSocket.
    var wsUri = "ws://aws.tumundoenlared.com:9000/chat/server.php";
    websocket = new WebSocket(wsUri);

    websocket.onopen = function(ev) { // Funcion llamada al conectarse al servidor
        $('#message_box').append("<div class=\"system_msg\">Connected!</div>");
    }
```

Figura 6. Creación del objeto WebSocket y método onopen. Elaboración propia

Al momento de crear el objeto WebSocket se debe enviar como parámetro la URL donde se encuentra ubicado el servidor. El método **onopen** es ejecutado cuando la conexión se realiza correctamente, en este caso se debe indicar al usuario simplemente que se encuentra conectado.

Usando el método **click** (Figura 7) de JQuery se captura la acción cuando el usuario haga clic en el botón enviar (SEND), posteriormente se realizan validaciones a los campos y se agrupan los datos a enviar en formato JSON, esta información es transferida al servidor mediante el método **send** de la clase WebSocket.

De igual forma, el método **onmessage** (Figura 8) se ejecuta inmediatamente llegan datos del servidor; la variable **ev.data** contiene en formato JSON los datos recibidos.

Por otro lado, los métodos **onerror** y **onclose** se disparan en caso de algún error o cierre de la conexión respectivamente. El resto de etiquetas HTML se emplean para dar formato y apariencia al chat. Así mismo, en el archivo **server.php** (que se encuentra en el servidor) se capturan los datos enviados desde el cliente usando el método **socket_recv**, luego se aplica el formato JSON para posteriormente replicarlo a todos los clientes conectados.

Una vez listos los archivos se procede a iniciar el servidor ejecutando **server.php** usando SSH:

```
~$ php /opt/lampp/htdocs/chat/server.php
```

A este archivo se accede mediante una URL, como se observa en la Figura 10. Del mismo modo, varios clientes se pueden conectar de forma simultanea por medio de navegadores web ingresando a la misma URL. Por lo tanto, si uno de ellos envía un mensaje de texto desde el formulario se podrá apreciar de inmediato que éste es replicado a todos los usuarios que se encuentran conectados (Figura 11). Este comportamiento es implementado por medio del script que se observa en la Figura 9.

Es importante recalcar que al tratarse de HTML5 y JavaScript, el sistema puede ser accedido a través dispositivos móviles.

```
$('#send-btn').click(function(){ // Función para enviar
                                 //  mensaje al presionar boton
    var mymessage = $('#message').val();
    var myname = $('#name').val();

    if(myname == ""){
        alert("Enter your Name please!");
        return;
    }
    if(mymessage == ""){
        alert("Enter Some message Please!");
        return;
    }
    document.getElementById("name").style.visibility = "hidden";

    var objDiv = document.getElementById("message_box");
    objDiv.scrollTop = objDiv.scrollHeight;
    //prepare json data
    var msg = {
    message: mymessage,
    name: myname,
    color : '<?php echo $colours[$user_colour]; ?>'
    };
    //convert and send data to server
    websocket.send(JSON.stringify(msg));
});
```

Figura 7. Script en JS para capturar, preparar y enviar los datos al servidor. Elaboración propia

```
    //#### Al recibir un mensaje del servidor
    websocket.onmessage = function(ev) {
        var msg = JSON.parse(ev.data);
        var type = msg.type; //message type
        var umsg = msg.message; //message text
        var uname = msg.name; //user name
        var ucolor = msg.color; //color

        if(type == 'usermsg')
        {
            $('#message_box').append("<div><span class=\"user_name\"
            style=\"color:#"+ucolor+"\">"+uname+"</span> :
            <span class=\"user_message\">"+umsg+"</span></div>");
        }
        if(type == 'system')
        {
            $('#message_box').append("<div class=\"system_msg\">
            "+umsg+"</div>");
        }

        $('#message').val(''); //reset text

        var objDiv = document.getElementById("message_box");
        objDiv.scrollTop = objDiv.scrollHeight;
    };

    websocket.onerror    = function(ev){$('#message_box').append("<div
    class=\"system_error\">Error Occurred - "+ev.data+"</div>");};

    websocket.onclose    = function(ev){$('#message_box').append("<div
    class=\"system_msg\">Connection Closed</div>");};
});
```

Figura 8. Script en JS para recibir y mostrar la información recibida desde el servidor. Elaboración propia

```
//loop through all connected sockets
foreach ($changed as $changed_socket) {

    //check for any incomming data
    while(socket_recv($changed_socket, $buf, 1024, 0) >= 1)
    {
        $received_text = unmask($buf); //unmask data
        $tst_msg = json_decode($received_text); //json decode
        $user_name = $tst_msg->name; //sender name
        $user_message = $tst_msg->message; //message text
        $user_color = $tst_msg->color; //color

        //prepare data to be sent to client
        $response_text = mask(json_encode(array(
        'type'=>'usermsg',
        'name'=>$user_name,
        'message'=>$user_message,
        'color'=>$user_color)));
        send_message($response_text); //send data
        break 2; //exist this loop
    }

    $buf = @socket_read($changed_socket, 1024, PHP_NORMAL_READ);
    if ($buf === false) { // check disconnected client
        // remove client for $clients array
        $found_socket = array_search($changed_socket, $clients);
        socket_getpeername($changed_socket, $ip);
        unset($clients[$found_socket]);

        //notify all users about disconnected connection
        $response = mask(json_encode(array(
        'type'=>'system',
        'message'=>$ip.' disconnected')));
        send_message($response);
    }
}
```

Figura 9. Script PHP para recibir y replicar los datos desde y hacia los clientes. Elaboración propia

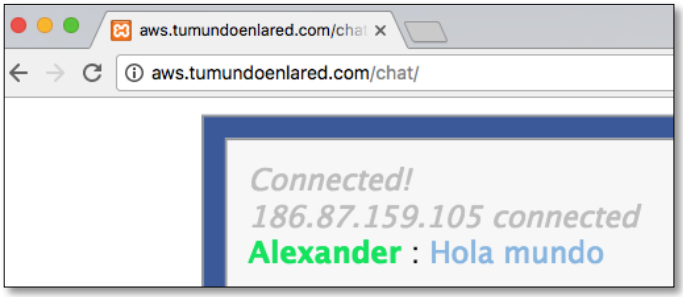

Figura 10. Acceso de clientes desde navegador web. Elaboración propia

Figura 11. Cada cliente puede visualizar los mensajes enviados al servidor. Elaboración propia

5.3 Trasmisión de video usando *websocket*

Ahora, una vez que el servidor y el *streaming* de datos se encuentran funciona, se transmite video sin necesidad de usar *frameworks* o librerías adicionales. Existen herramientas que permiten realizar esta tarea por medio de *Websockets*, entre las más comunes se encuentran node.js, express.js y socket.io, pero para este proyecto se empleará únicamente JavaScript y HTML5 en el lado cliente.

Donde, para lograr este cometido se realizarán algunos cambios en los archivos **index.php** y **camara.php**.

El archivo **index.php** es renombrado como **camara.php** y se agrega el método **mediaDevices** de la clase **navigator,** éste se encargará de proveer acceso a la cámara desde JavaScript mediante el navegador web.

Tomando como referencia a [28] se describe el funcionamiento básico de la integración del código JavaScript con HTML (Figura 13):

- La cámara entrega el *stream* de video dentro de las etiquetas <video>…<\video> en el DOM de HTML.
- Se define la función denominada transmitiendo () que se encargará de crear un *frame* de video en Canvas mediante el método *drawImage*.
- Se obtiene la información del *frame* o fotograma en base64 mediante el método toDataURL.
- Ya que el resultado en base64 es la representación del fotograma mediante una cadena de texto, esta se almacena en la variable *datosfoto* que luego es enviada al servidor con el método *send* de la clase *websocket*.
- Con el método *setInterval* se ejecuta la función transmitiendo() cada 500 ms, por cada iteración del método se procesará y enviará la información de un fotograma al servidor.

A partir de este instante la información está siendo enviada al servidor cada 500ms. Posteriormente, se realiza algunos cambios en el archivo **server.php** para enviar los datos en base64 a cada uno de los clientes conectados (se guardan dichos cambios como **serverCam.php)**.

De esta forma, se crea el módulo encargado de recibir y mostrar los *frames* en el navegador del cliente (este módulo se guarda como **visorCamara.php**). Ahora, usando nuevamente el archivo **index.php** como base, se reestructura el script para recibir los fotogramas del servidor con el método **onmessage**, los datos se almacenan en la variable ev.data que son desplegados después como una imagen en un contenedor en el DOM.

Finalmente se logra enviar la secuencia de los fotogramas a los clientes, quienes verán en vivo las capturas realizadas con la cámara del equipo que ejecuta el módulo **camara.php**.

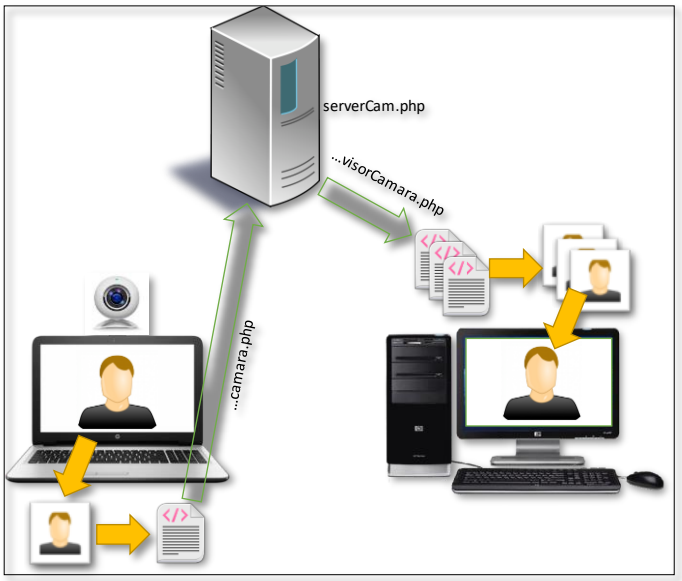

Figura 13. Funcionamiento general del sistema de video. Elaboración propia

6. Conclusiones

En el proceso de construcción de la plataforma se decidió utilizar *Javascript* puro, con el fin de seguir una línea "purista" de implementación. Sin embargo, en el proceso que implica el estado del arte se encontraron *frameworks* orientados al desarrollo de aplicaciones web tales como Ruby on Rails, Django y Lavarel. No obstante, por motivos de compatibilidad se realizó el presente desarrollo en PHP puro.

Como se puede observar en el proceso de implementación no fue necesario la instalación de software de terceros (*plugins*) lo cual es una de las ventajas al utilizar *Websockets*, puesto que elimina una de las barreras más notorias referente al tipo del sistema operativo que utilice el usuario.

Del mismo modo, otro de los hallazgos importantes fue comprobar que al desarrollar una aplicación web que emplee las tecnologías HTML5 y *WebSockets* puede ser accedida desde dispositivos móviles (como se mencionó en el apartado 5), lo que lleva a la conclusión que el requerimiento más importante es que debe contar con un navegador compatible con las mismas.

Para finalizar, en la fase de pruebas se pudo constatar perfectamente la bidireccionalidad en la transmisión que propone el empleo de *WebSocket* tanto en texto como en video.

Por último, a continuación, se muestran los pasos para recrear el funcionamiento de este proyecto:

Chat

- Active el servidor ingresando a la siguiente URL: http://quipito.pluginsoft.net/chat/server.php Incluya clientes accediendo desde diferentes equipos a la siguiente URL: http://quipito.pluginsoft.net/chat/index.php

Video

- Use siempre Mozilla Firefox para esta actividad.
- Active el servidor ingresando a la siguiente URL: http://quipito.pluginsoft.net/chat/serverCam.php
- Ingrese a la siguiente URL desde un equipo con webcam: http://quipito.pluginsoft.net /chat/camara.php
- Incluya clientes accediendo desde diferentes equipos a la siguiente URL: http:// quipito.pluginsoft.net/chat/visorCamara.php

Este proyecto puede ser descargado en el siguiente enlace: https://pluginsoft.net/DEMOS/Servidor_Socket_PHP_Alexander_Valencia.zip

Referencias

[1] R. I. y E. CRIE, "Apertura de la Casa Utepitos," 2016. [Online]. Available: https://www.utp.edu.co/pdi/noticias/apertura-de-la-casa-utepitos.html. [Accessed: 26-Jul-2018].

[2] E. J. Acevedo Clavijo, S. Hernández Chacón, and E. Cardoza Vásquez, "Tecnología streaming para radio digital universitaria," *Rev. ESAICA*, vol. 1, no. 1, p. 9, 2015.

[3] D. Carolina *et al.*, "Implementación De Un Servidor Radio Streaming Con Openims En La Universidad Sergio Arboleda," 2010.

[4] P. Por *et al.*, "Prototipo de Red Social usando tecnología Live Streaming para transmisión en tiempo real con aceptación de los usuarios . Plan de proyecto de Trabajo de Graduación," 2014.

[5] I. Anibal Ruben Mantilla Guerra and I. Jaime Fabian Naranjo Anda, "Implantación de un Sistema de Video Conferencia Multipunto a Través de Internet Aplicando Tecnología 'Streaming,'" 2009.

[6] A. Development, *HTML5 Programming*, vol. 2011. 2011.

[7] S. Panagiotakis, K. Kapetanakis, and a G. Malamos, "Architecture for Real Time Communications over the Web," *Int. J. Web Eng. 2013*, vol. 2, no. 1, pp. 1–8, 2013.

[8] Kaazing, "About HTML5 WebSocket." [Online]. Available: https://www.websocket.org/aboutwebsocket.html. [Accessed: 26-Jul-2018].

[9] Q. Liu and X. Sun, "Research of Web Real-Time Communication Based on Web Socket," *Int. J. Commun. Netw. …*, vol. 2012, no. December, pp. 797–801, 2012.

[10] J. Lengstorf and P. Leggetter, *Realtime Web Apps: With HTML5 WebSocket, PHP, and jQuery*. 2013.

[11] E. Bozdag, A. Mesbah, and A. Van Deursen, "A Comparison of Push and Pull Techniques for A JAX Web-based Real-time Event Notifi-," *Symp. A Q. J. Mod. Foreign Lit.*, vol. 3, pp. 1–8, 2007.

[12] U. K. Algorithm, "Real Time Web Applications Comparing: Frameworks and transport mechanisms," pp. 1–4, 2014.

[13] E. Bozdag, "Push solutions for AJAX technology," 2007.

[14] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M.~Peha, "Streaming Video over the Internet: Approaches adn Directions," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 282–300, 2001.

[15] F. Kozamernik, "Media streaming over the internet," *EBU Tech. Rev.*, no. October, pp. 1–15, 2002.

[16] M. MACIEJEWSKI, N. I. Caroline FISCHER, and Y. ROGINSKA, "Streaming and online access to content and services," 2014.

[17] E. I. Acuña, "Análisis De Tecnologías De Streaming: Evaluación De Protocolos Y Diseño De Un Caso De Estudio," Universidad Politécnica de Madrid, 2016.

[18] A. Fecheyr-Lippens, "A Review of HTTP Live Streaming," *Issuu.com*, no. January 2010, 2010.

[19] A. Lombardi, *WebSocket Lightweight Client-Server Communications*. O'Reilly, 2015.

[20] V. Wang, F. Salim, and P. Moskovits, *The Definitive Guide to HTML5 WebSocket*. 2013.

[21] V. Pterness, *Getting Started with HTML5 Web Socket Programming*. 2013.

[22] I. Fette, "The WebSocket Protocol," 2011.

[23] Mozilla, "Mozilla Developer Network." [Online]. Available: https://developer.mozilla.org/es/. [Accessed: 26-Jul-2018].

[24] J. Cameron, "Webmin," 2015. [Online]. Available: http://www.webmin.com/. [Accessed: 26-Jul-2018].

[25] A. Kikabidze, "GitHub - php websockets chat," 2014. [Online]. Available: https://github.com/akalongman/php-websockets-chat.

[26] Amazon, "AWS | Cloud Computing - Servicios de informática en la nube." [Online]. Available: https://aws.amazon.com/es/. [Accessed: 26-Jul-2018].

[27] PHP, "PHP: Hypertext Preprocessor." [Online]. Available: http://php.net/. [Accessed: 26-Jul-2018].

[28] J. L. Monteagudo, "Emitir video con tu dispositivo móvil con Node.js, Express.js y Socket.IO," 2012. [Online]. Available: http://www.jlmonteagudo.com/2012/10/emitir-video-con-tu-dispositivo-movil-con-node-js-express-js-y-socket-io/. [Accessed: 26-Jul-2018].