HTML5 Web Sockets

Aleksandar Kotevski[1], Gjorgi Mikrovski[2] and Ilija Jolevski[3]

Abstract – **The HTML5 Web Sockets specification defines the Web Sockets API that enables web pages to use the Web Socket protocol for full-duplex communication with a remote host. HTML5 Web Sockets defines a channel for full-duplex communication that operates through a single socket over the Web and represents a colossal advance, especially for real-time, event-driven web applications - significantly reduction in unnecessary network traffic and latency compared to legacy polling and long-polling solutions that are often used to push real-time data to clients. Through this paper should be present benefit from using Web socket in HTML5 and practical realization on several examples.**

Keywords- ***HTML5 Web Sockets, communication, traffic network, protocol***

I. INTRODUCTION

In the existing technologies that enable pushing of data from a server to a subscribing client are not using true asynchronous communication. Instead they emulate this using long polling where the client polls the server for data. If no data exists for the client, the server does not immediately respond but rather, he waits for data to be available and then sends it to the client. This technique of delayed responses relies on always having an outstanding client request that the server can respond to. What might look like a server initiated communication actually relies on the client making requests to the server.

II. IMPLEMENTING HTML5 WEB SOCKET

HTTP was originally designed for document transfer, but, this protocol is half-duplex (Figure 1 shows the half-duplex architecture), so it can't be use in real-time communications. It that situation, Ajax (Asynchronous JavaScript and XML) can be use to build interactive Web applications, so content can be changed without refreshing the entire page. But, Real-time often achieved through polling and long-polling, which is reason for lots of complexity.

The latest HTML version, HTML5, introduce elements that integrate web front-end much tighter with server back-end. Most importantly, web sockets are now being introduced and thereby allowing browser applications to receive asynchronous updates from the server side, so called server push. Web sockets define a full-duplex communication channel that operates over a single socket using HTML5 compliant browsers. Web sockets allow for true low latency applications and put less strain on the server.

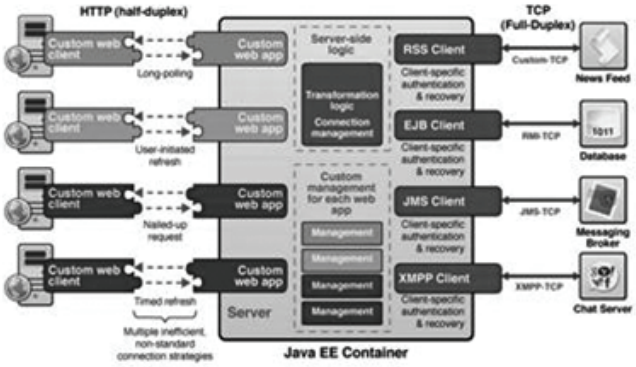

Figure 1: Half – duplex architecture

.

The goal with this paper is to implement the HTML5 web socket protocol for presenting real time data. Data will be pushed to the client without any explicit request from the client allowing for true asynchronous updates from the server. HTML5 Web Sockets is not just another incremental enhancement to conventional HTTP communications; it represents a colossal advance, especially for real-time, event-driven web applications. In simple words the web application client can maintain an always open connection with the server and data can be sent and received at the same time.

It runs via port 80/443, the TCP handshakes are HTTP compatible and it also integrates with cookie based authentication. The message headers have been kept small (only 2 bytes overhead per frame) and latency from establishing new connection every time as in HTTP have been dealt with by using a single persistent connection.

All these result in impressive network throughput. Apart from that Websockets API is much simpler to code than the XmlHttpRequest(). Bottomline, you can make the process of web application developmentmore responsive and interactive with lesser effort.

Today's Web applications demand reliable, real-time communications with near-zero latency, not just broadcast, but bi-directional communication. Examples: financial applications, social networking applications, online games, smart power grid etc.

Implementing WebSocket into the HTML5 is really useful, because he is full-duplex, enables web pages to communicate with a remote host, traverses firewalls, proxies and routers seamlessly. HTML5 WebSocekt API has a following structure:

1 Aleksandar Kotevski is with the Faculty of Law, Partizanska BB, 7000 Bitola, Macedonia
E-mail: aleksandar.kotevski@uklo.edu.mk.
2 Gjorgi Mikarovski is with the Faculty of Technical science, Ivo Lola Ribar BB, 7000 Bitola, Macedonia
E-mail:gjorgi.mikarovski@tfb.uklo.edu.mk.
3 Ilija Jolevski is with the Faculty of Technical science, Ivo Lola Ribar BB, 7000 Bitola, Macedonia
E-mail:ilija.jolevski@uklo.edu.mk.

```
interface WebSocket {
  readonly attribute DOMString URL;
// ready state
const unsigned short CONNECTING = 0;
const unsigned short OPEN = 1;
const unsigned short CLOSED = 2;
readonly attribute unsigned short readyState;
readonly attribute unsigned long bufferedAmount;
// networking
attribute Function onopen;
attribute Function onmessage;
attribute Function onclose;
boolean send(in DOMString data);
void close();
}
```

III. USING HTML5 WEB SOCKET

HTML5 WebSocket should be use in situation where web application has data that must flow bi-directional simultaneously, when web application is used by huge number of users simultaneously, in situation where the web application must extend TCP-based protocols to the browser. Also, HTML5 WebSocket should be use when web application developers need an API that is easy to use or when web application must extend SOA over the Web and in the Cloud.

Connection established by upgrading from the HTTP protocol to the WebSocket protocol using the same TCP connection. Once upgraded, WebSocket data frames can be sent to both client and server in full-duplex mode.One possible WebSocket architecture is shown on figure 2.

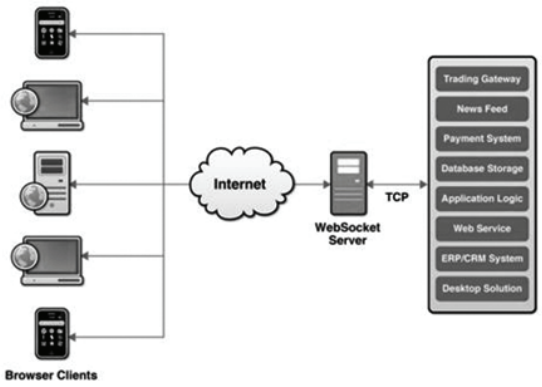

Figure 2: WebSocket architecture

To establish a WebSocket connection, the client sends a WebSocket handshake request, and the server sends a WebSocket handshake response, as shown in the following example**:**

Client

GET /demo HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: example.com
Origin: http://domain.com
Sec-WebSocket-K1: 4 @1 56846xW%3l 1 2
Sec-WebSocket-K2: 12548 5 Y3 1 .P20

^n:df[4R

Server

HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://domain.com
Sec-WebSocket-Location: ws://domain.com/test
Sec-WebSocket-Protocol: sample

8j&S'y:G*To,:xa-

After receiving the response HTTP header, data will be transmitted according to the WebSocket protocol. This means at this point only WebSocket frames will be transferred over the wire. A frame can be sent at each time in each direction. The WebSocket protocol defines two types of frames: a *text frame* and a *binary frame*. The text will be transferred UTF8-encoded between the start and the end byte. A text frame requires only 2 additional bytes for packaging purposes. Figure 3 shows a text frame for the string "Today" and the string "Sat April 25 11:20:005 2011".

Text frame of "Today":
0x22 0x47 0x65 0x73 0x44 0x61 0x41 0x65 0xFF

Text frame of "Sat April 25 11:20:005 2011":
0x00 0x53 0x61 0x74 0x20 0x4D 0x34 0x72 0x20 0x31
0x33 0x98 0x31 0x23 0x3A 0x30 0x30 0x3A 0x34 0x35
0x68 0x95 0x45 0x54 0x30 0xFF

If the handshake was successful, then the data transfer starts.

This is a two-way communication channel where each side can, indeendently from the other, send data.

Data is sent in the form of UTF-8 text. Each frame of data starts with 0x00 byte and ends with a 0xFFbyte, with the UTF- text in between. The protocol is designed to support other frame types in future. Instead of the 0x00 byte, other bytes might in future be defined. Frames denoted by bytes that have the high bit set (0x80 to 0xFF) have a leading length indicator.

The Sec-WebSocket-Key1 and Sec-WebSocket-Key2 fields and the 8 bytes after the fields are random tokens which the server uses to construct a 16-byte token at the end of its handshake to prove that it has read the client's handshake.

The handshake is constructed by concatenating the numbers from the first key, and dividing by the number of spaces. This is then repeated for the second key. The two resulting

numbers are concatenated with each other, and with the last 8 bytes after the fields. The final result is an MD5 sum of the concatenated string.

The handshake looks like HTTP but actually isn't. It allows the server to interpret part of the handshake request as HTTP and then switch to WebSocket.

Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Text frames can be sent full-duplex, in either direction at the same time. Binary frames are not supported yet in the API. WebSocket text frames use a terminator, while binary frames use a length prefix.

The following listing is example of JavaScript using the WebSocket interface:

```
<html>
  <head>
    <script type='text/javascript'>
      var ws = new WebSocket('ws://domain.com/test');
      ws.onmessage = function (message) {
        var messages = document.getElementById('messages');
        messages.innerHTML += "<br>[in] " + message.data;
      };
      sendmsg = function() {
        var message =
document.getElementById('message_to_send').value
        document.getElementById('message_to_send').value =
"
        ws.send(message);
        var messages = document.getElementById('messages');
        messages.innerHTML += "<br>[out] " + message;
      };
    </script>
  </head>
  <body>
    <form>
      <input type="text" id="message_to_send" name="msg"/>
      <input type="button" name="btn" id="sendMsg"
value="Send" onclick="javascript:sendmsg();">
      <div id="messages"></div>
    </form>
  </body>
</html>
```

Next listing present one more example of using the WebSocket:

Step 1:
Create a new WebSocket connection to WebSocket server at test.example.com.

```
var stockTickerWebSocket = new
WebSocket("ws://test.example.com");
```

Step 2: Attach JavaScript Callback Functions
Associate event listeners to handle each phase of the connection life cycle.

```
stockTickerWebSocket.onopen = function(evt) {
alert("Connection open...");
};
stockTickerWebSocket.onmessage = function(evt) {
alert( "Update: " + evt.data);
};
stockTickerWebSocket.onclose = function(evt) {
alert("Connection closed.");
};
```

Step 3: Send and Receive Data
To send a message to the server, simply call the postMessage method on the webocket with the content you wish to send to the server.

```
stockTickerWebSocket.postMessage("MSG:
GOOG,100@200.25″);
```

This will send the MSG message to the server. Any message coming from the server will be delivered to the onmessage callback registered in step #2.

Step 4: Disconnect When Done
When completed, call the disconnect() method to close the WebSocket connection.

```
stockTickerWebSocket.disconnect();
```

As demonstrated in the example above, there are no HTTP requests made to the server from the client side to retrieve data, instead the data was pushed to the client from the server - when it becomes available.

When a new WebSocket connection is established the browser opens an HTTP connection to the server first and then negotiates with the server to upgrade the connection to a dedicated and persistent WebSocket connection. This process automatically sets up a tunnel through to the server - passing through all network agents (proxies, routers, and firewalls) in the middle (very much like HTTPS establishing a secure, endtoend connection), solving numerous issues that the various Comet programming techniques encountered. Once established the WebSocket is a fullduplex channel between the client and the server.

IV. Compatibility

Currently all browsers do not support the Websockets API. List of browsers that support them are:

- Chrome 4.0
- Firefox 4.0 beta
- Opera 11
- Safari 5.0.3
- IE9

For other browsers it’s not so clear but support will come as major browsers are already in line.

Overall things appear exciting as web applications will become realtime and allow better resource utilization allowing developers to build interactive and responsive applications.

V. Conclusion

By using HTML5 WebSockets, writing highly interactive *real-time web* applications becomes very easy and practical. The WebSocket API is very easy to understand andn

also to use. The underlying WebSocket protocol is high efficient: there is a minimal overhead involved in managing a WebSocket. Due the fact that the WebSocket protocol runs on the top of TCP, the WebSocket protocol does not have to deal with "hacks" as do popular Comet protocols like Bayeux or BOSH. Simulating a bidirectional channel over HTTP leads to complex and less efficient protocols. Especially if only a small amount of data will be transferred, such as tiny notification events, the overhead of the classic Comet protocols is very high. This is not true for WebSockets. To establish a new WebSocket connection, the WebSocket protocol makes use of the connection management capabilities of the HTTP protocol.

On the other hand, WebSockets do less for reliability. This has to be done on the application (sub-protocol) level. In contrast to Server-Sent events, the WebSocket protocol does not include reconnect handling or guarantee message delivery. The current WebSocket protocol represents a low-level communication channel only.

REFERENCES

[1] W3C, 2009. Html 5 specification, canvas section. http://dev.w3.org/html5/spec/Overview.html#the-canvas-element

[2] HTML5–A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/

[3] HTML5 Demos and Examples. http://html5demos.com

[4] How HTML5 Will Change the Way You Use the Web. http://lifehacker.com/5416100/how-html5-will-change-the-way-you-use-the-web

[5] Pushing real time data using HTML5 Web Sockets, Nikolai Qveflander

[6] Harnessing the Power of HTML5 WebSocket to Create Scalable Real-time Applications - Brian Albers & Peter Lubbers, Kaazing

[7] Using the HTML5 WebSocket API,Peter Lubbers, Brian Albers and Frank Salim

[8] Using the HTML5 Web Workers API, Peter Lubbers, Brian Albers and Frank Salim

[9] The Extensible HyperText Markup Language. Steven Pemberton et al.W3C, 2000

[10] http://ezinearticles.com/?expert=Eric_D_Rowell

[11] Web sockets now available in google chrome, December 2009. http://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html.

[12] Ian Hickson. Google Inc. The web socket api, June 2010. http://dev.w3.org/html5/websockets/

[13] Ian Hickson. Google Inc. The web socket protocol, May 2010.http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-75

[14] HTML5, W3C Working Draft, 24 June 2010, Ian Hickson, http://www.w3.org/TR/html5/

[15] The Web Sockets API, W3C Working Draft, 22 December 2009, Ian Hickson, http://www.w3.org/TR/websockets/

[16] HTML Device, Editor's Draft, 15 June 2010, Ian Hickson,http://dev.w3.org/html5/html-device/#peer-to-peer-connections