PUSH HTTP COMMUNICATION ON HTML5 WEBSOCKET:
IMPLEMENTATION IN CURRENT HTML4 ENVIRONMENT

Hendy Wijaya1)  Pujianto Yugopuspito2)
Universitas Pelita Harapan, Lippo Village, Karawaci
hendy_wijaya@uph.edu1)  yugopuspito@uph.edu2)

ABSTRAK

Kata Kunci: Web Socket, Real-Time Communication, Server-Sent Events

1. Introduction
Since its wide emergence in mid 1990's, web technology has brought many changes not only in computing and information technology field, but also in other fields such as social, life sciences, and even to today's economic, entertainment, and people's daily life activities. Pushed by ever-flowing demands for improvements has made development in web technology to grow rapidly. One of the biggest pursuits in web technology enhancement is to achieve real-time communication between the client browser and the web server.

The need for real-time and pushed communication in the web environment has been increasing in many applications such as browser based pushed email, live stock option data, bidding sites, chat, and many more[3]. The biggest challenge in providing such communication is the limitation of HTTP, a protocol used in current web communication, which capable only in delivering a half-duplex communication.

Various methods has been developed to answer this challenge. The earliest method is to use polling technique where the client browser is configured to send a continuous request to the web server in a given set of intervals, in order to receive and trigger a response from the server. This makes an almost real-time communication with interval delays. The major disadvantage of polling is its exhausting process of continuous request sending, resulting (n-1) unnecessary requests out of n requests, and since data occurrence or update in real-time environment is unpredictable, this technique which uses a set of pre-determined intervals fails to meet the high expectancy of real-time web communication.

Instead of working around the client-side technology using exhausting client requests, current methods lead to server-side push technology allowing server to initiate message delivery without having the client to send request in the first time. The World Wide Web Consortium (W3C) has decided to establish standardization of this technology by including Web socket and server-sent events in the upcoming HTML5 specification, which will be available in web browsers by the year of 2020.

The general objective of this research is to explore and observe the implementation of HTML5 Web socket for real-time and pushed HTTP communication in current HTML4 browsers environment. This objective can be broken-down as the following:

1. To construct a proof-of-concept of Web socket capability in serving real-time HTTP communication.
2. To understand basic functionality and Web socket workflow.
3. To observe and analyze sample applications using Web socket.

This research ponders on three problem definitions as the following:

1. Establishment of a working environment for Web socket implementation based on its requirements
2. Web socket implementation in serving a proven and appropriate real-time and pushed communication
3. Feasibility study in current HTML4 browser environment as a client of real-time and pushed communication provided by HTML5 Web socket.
2. Fundamental Theories

2.1 HTML and HTML5

HTML, which stands for Hypertext Markup Language, is the predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists etc as well as for links, quotes, and other items.

The current version of HTML is version 4, while HTML5 was first published in draft in January 2008, many browser vendors has been focusing on certain newly introduced features since then. The communication section of HTML5 specification contains Web sockets and server-sent events, which making it possible for browser to support full-duplex direct TCP communication.

In 2007 the specification of full-duplex communication of TCPConnection API and protocol within HTML5 were added. With its development since then up until now, the HTML5 Communication section can be considered stable and nearly complete with no major changes or additions in plan. Therefore browser vendors has been putting their interest in having a native support for the Communication section, despite of the draft status in the full HTML5 proposal. There are three major areas in which HTML5 specification affects real-time and pushed HTTP communication:

2.1.1 Cross-Document Messaging

For security reasons of preventing cross-site scripting (XSS) attacks, browser programs implement same-origin policy preventing communication between pages from different origins to affect one another. On the other side, this security feature also disable communication of non-malicious documents between different origins. Cross-Document Messaging defines a method which can allow documents and scripts to communicate across different origins with a set of security rules and protections.

2.1.2 Server-Sent Events

Standardization and formalization of sending a continuous and server triggered data stream from web server to a client browser is stated in server-sent events of HTML5. The design of server-sent events emphasizes in enhancing native and cross-browser streaming. A new HTML DOM element called `eventsource` is included in server-sent events specification. Eventsource connects to a server URL to listen and receive a server-triggered-sent event stream.

The communication handshake is managed through the HTTP headers traded between server and client. Each event is given a unique id which included in every event sent by the server, and the client holds and sends last-event-id header upon communication reconnection happens to make sure event stream can resume without anything missed or unnecessary repeated during message delivery. Server-sent events also include other headers to handle TCP connection break-down, delay, or network error, giving more exception control in the server and client side.

2.1.3 Web Sockets

Web sockets interface specified in HTML5 specification states a full-duplex communication channel that runs over a single socket.

Web socket connection is established by upgrading HTTP protocol to Web socket protocol during the initial handshake between web server and client browser. Upon connection establishment, Web socket data frames can be transmitted between web server and client browser in a full-duplex communication mode.

Apart from its advanced feature in providing full-duplex communication, Web socket still supports basic HTTP protocol features such as cookie-based authentication, HTTP load balancers, and there is no changes needed in infrastructure firewalls or routers. Although Web socket protocol supports many major client browsers, Web socket cannot deliver raw binary data to Javascript client, since Javascript does not support byte type for binary data.

2.2 Kaazing Gateway Server

Kaazing Gateway is a Web socket server technology that enables full-duplex communication from the browser to any TCP-based back-end service such as JMS, JMX, IMAP, and Jabber. Kaazing Gateway eliminates the need for complex server- and client-side architectures to bridge various protocols to the browser over HTTP.

This simplified architecture allows developers to code directly against back-end services, using JavaScript, which eliminates the need for complex server- and client-side architectures to bridge various protocols to the browser over HTTP. Developers simply configure Kaazing Gateway to communicate directly with the back-end APIs such as Stomp or XMPP, and develop the client application.

Kaazing Gateway consists of 2 (two) components, the java-based gateway server components and the client library which is a collection of Javascript library.
3. **Research Methodology**

The methodology used in this research is defined as follows:

1. An architectural design of Web socket implementation is constructed and given more details. This design and workflow overview helps in understanding the functionality of Web socket, helping in the implementation of Web socket and its connection and integration with other web components.

2. Implementation of the required components to prepare a demonstration of fully working sample application which runs by using Web socket technology. The sample application will elaborate more in server-sent events to achieve a real-time communication which initiated both the web server by sending continuous event stream and the browser clients by starting, the first request transaction, and communicating interactively with the web server.

3. Analyze the sample Web socket application in order to determine its objective to conduct a real-time and pushed HTTP communication between the web server and browser client.

4. **Web Socket Communication Design**

Web Socket enables a full-duplex communications channel that operates over a single socket\(^1\). Addressing the security issues, Web socket connection can be established both on plain text HTTP (using ws protocol) or encrypted HTTPS (using wss protocol) without the need to change any behavior or feature.

Web socket is initiated by the browser client to setup the socket, while on the server-side, a Web socket server supports direct TCP connectivity between the server and client. In current HTML4 browser environment, enhancement must be made in both client and server side in order to establish Web socket. Kaazing gateway architecture enhances both client browser with a set of Javascript libraries, and web server side with a java-based gateway server. Overview of this architecture can be described in the following figure:

![Figure 4.1 Architectural Overview of Web Socket Communication](image)

The initial connection handshake by request sending from the client browser does not have much difference with traditional HTTP connection, except during reconnection an additional last-event-id is added to the request header. The following figure describes the interaction flow between web server and client browser on Web socket connection.

![Figure 4.2 Interaction Diagram Between Components in Web Socket Driven Communication](image)
As described in the interaction diagram, the client requests for a specific URL to the web server, then web server sends back a response of HTML page with Web socket Javascript client library for the browser to load. The Javascript client library then initiates a Web socket connection to the gateway server by sending the desired web socket URI. This process can be authenticated by browser cookie to prevent unauthorized request sent to the gateway server. Based on the given Web socket URI, gateway server determines what kind of request is being made. The router then requests to the respective back-end server whether it is an application, e-mail, or database server. The gateway server then sends back a server-sent response event to the client browser. Up to this point the Web socket connection and a full-duplex communication between the server and the client has been established. The back-end server can now trigger the gateway server to broadcast and push particular message to all the connected browser clients. In each server-sent event from the server, the client will then send back a request type communication to the server containing acknowledgement message.

5. Implementation Using Push Communication Application

To assess the previously described design, an implementation using application sample was made. The primary objective is to provide a proof-of-concept when a real-time communication between client and server is achieved. The implementation uses a stock option feeds application as a push communication application that pushes the stock option changes to the gateway server broadcasting the message to all connected clients. This implementation involves the following components:

2. Apache ActiveMQ Web Server.
3. Java-based sample application which reads sample stock option data from text file, and pushes the information for 30 minutes time range with random tick interval in seconds.
4. Web browser; used in this research are Firefox 3.0, Internet Explorer 6.1, Opera 9.0.
5. HTTP Proxy to monitor connection in between client and server by trapping the request and response transactions along with their headers.

The following figure describes how the implementation of the sample application is being accessed through a web browser.

![Figure 5.1 Web Application Interface Using Web Socket for Real-Time and Pushed Communication](image)

The interface above shows several columns containing company names of the stock issuer, ticker names, prices, price changes and the percentages of the price changes that are continuously updated in a random interval generated by the sample application server. The application was run for 30 minutes while the proxy server is recording the request and response send back-forth the client and the server. The following header samples were collected during web application execution.

<table>
<thead>
<tr>
<th>Event</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Initial URL Request</td>
<td>GET /demo/stock.html HTTP/1.0 Accept: <em>/</em> Accept-Language: en-us UA-CPU: AMD64</td>
<td>HTML Content</td>
</tr>
</tbody>
</table>
| **(2) Initial web socket connection** | POST `/activemq?encoding=base64`  
HTTP/1.0  
Accept: */*  
Accept-Language: en-us  
x-origin: http://lkpp.uph.ac.id:8000  
Referer: http://lkpp.uph.ac.id:8001/?kr=x!lkpp.uph.ac.id!http%3A%2F%2Flkpp.uph.ac.id%3A8001!2321203218606191!  
UA-CPU: AMD64  
Pragma: no-cache  
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; Win64; x64; SV1; .NET CLR 2.0.50727)  
Host: lkpp.uph.ac.id:8001  
Content-Length: 0  
Cookie: KSESSIONID=y8yUpteW2KiPtrAfXOj2TK4p/dN3T7+0  
HTTP/1.1 201 Created  
Server: Kaazing Gateway  
Content-Type: text/plain  
Connection: close  
Access-Control-Allow-Origin: http://lkpp.uph.ac.id:8000  
Access-Control-Allow-Credentials: true  
Content-Length: 70  
http://lkpp.uph.ac.id:8001/activemq?.kz=mBSKaVavwpgo643ILkSdGKpxjAhwgoxv  |
| **(3) Server-sent** | GET `/activemq?.kz=mBSKaVavwpgo643ILkSdGKpxjAhwgoxv&.kb=512&.kp=256&.kc=text/plain;charset=UTF-8&.ko=http%3A//lkpp.uph.ac.id%3A8000 HTTP/1.0  
Accept: */*  
Accept-Language: en-us  
Referer: http://lkpp.uph.ac.id:8001/?kr=x!lkpp.uph.ac.id!http%3A%2F%2Flkpp.uph.ac.id%3A8000  
UA-CPU: AMD64  
Pragma: no-cache  
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; Win64; x64; SV1; .NET CLR 2.0.50727)  
Host: lkpp.uph.ac.id:8001  
Cookie: KSESSIONID=y8yUpteW2KiPtrAfXOj2TK4p/dN3T7+0  
HTTP/1.1 200 OK  
Transfer-Encoding: chunked  
Location: ws://lkpp.uph.ac.id:8001/activemq?.kz=mBSKaVavwpgo643ILkSdGKpxjAhwgoxv  
Content-Type: text/plain;charset=UTF-8  
Connection: close  
Cache-Control: no-cache  
Access-Control-Allow-Origin: http://lkpp.uph.ac.id:8000  
Access-Control-Allow-Credentials: true  
id:72  
data:Q09OTkVDVEVECnNlc3Npb246SUQ6dXBoLTEzM  
D1MTI0ODY0MjQ3Nzc2NS0yOjE6MToxMzUwMjA  
ZGVzdGluYXRpb246L3RvcGljL3N0b2RvbWVzdG  
PtcDoxMjQNjU4NTY4MDQ2CmV4cGlyZXM6MApwcm  
lcml0eTo0CgplIXZdsZXR0LVhY2thcmQgQ28uOkhQU  
T0yjwAo==  |
| **(4) Client Acknowledge** | POST `/activemq?.kz=mBSKaVavwpgo643ILkSdGKpxjAhwgoxv`  
HTTP/1.0  
Accept: */*  
Accept-Language: en-us  
x-origin: http://lkpp.uph.ac.id:8000  
Referer: http://lkpp.uph.ac.id:8001/?kr=x!lkpp.uph.ac.id!http%3A%2F%2Flkpp.uph.ac.id%3A8001  
UA-CPU: AMD64  
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; Win64; x64; SV1; .NET CLR 2.0.50727)  
HTTP/1.1 200 OK  
Location: ws://lkpp.uph.ac.id:8001/activemq?.kz=mBSKaVavwpgo643ILkSdGKpxjAhwgoxv  
Last-Event-ID: 1  
Connection: close  
Access-Control-Allow-Origin: http://lkpp.uph.ac.id:8000  
Access-Control-Allow-Credentials: true  
Content-Length: 0 |
From the samples shown above, it is indicated that after the Web socket connection has been successfully established, the server then began to push message to the client. For each successful message received by the client, the client sends back a request header containing acknowledgement message for each server-sent event id. The real-time and pushed communication can be observed in sample number 3, where client sends a complete request to the server through its Web socket Javascript client library, then in the response header, responses with chunked transfer encoding, allowing message to be pushed back to the client by the server through the Web socket interface.

6. Conclusion
HTML5 specification specially in Communication section which includes the feature of real-time communication over the HTTP protocol detailed with the specification of Web socket and server-sent events has been made possible in current HTML4 environment using additional Javascript client library and gateway server. Implementation on the sample stock option feed application, and the header data collected from the request and response transactions between client and server is a proof-of-concept of delivering real-time communication and pushed server message over Web socket interface. This can significantly increase the development pace of having real-time and push communication that complies future HTML5 Web socket specification in today’s HTML4 browser environment.

7. Research Advancement
There are two additional aspects that can be further developed excluded in this research. The first one is to explore more about the security issue and specification of using Web socket, not just relying on SSL availability of Web socket, but to explore more details in security issue when implementing Web socket supported application in both client and server side. The second interest field to explore more is the application framework development using real-time communication. Since such advancement will create more possibilities of user interaction within and across applications.

Bibliography