e-ISSN:2582-7219

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY

Volume 7, Issue 10, October 2024

© 2024 IJMRSET | Volume 7, Issue 10, October 2024| DOI: 10.15680/IJMRSET.2024.0710049

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Study of Web Protocol for Real-Time Data Transfer

Dr.Susil Kumar Sahoo, K R Sai Deekshitha

Professor & Head, Department of Computer Science & Applications, The Oxford College of Science, Bangalore, India

PG Student, Department of Computer Science & Applications, The Oxford College of Science, Bangalore, India

ABSTRACT: Today, with rapid growth in number of internet users, there is increasing demand for real-time data update. There are lots of business entities who want their customers to get instant product specific information updates on their web portals. HTML5 gives easy ways of implementing client side for real-time, full duplex client server communication over Web Socket protocol. This paper presents the necessity of real-time data exchange for web applications, comparison of Web Socket with various application layer communication techniques like HTTP polling, Long polling, Server Sent Events (SSE) etc., analysis of Web Socket handshake and data flow. This paper also presents some of Web Socket implementation techniques at server and client side.

KEYWORDS: Web Socket, Full duplex, Persistent, Real time communication, HTTP, HTML5, HTTP Streaming; Long-Polling; Latency

I. INTRODUCTION

It's been more than 20 years since large amount of information is available on internet in the form of web pages. For a long time, web pages were static and needs to be updated manually when information gets outdated. The changes in information were not getting reflected to the web page, until that page was not reloaded. Previously there was not that much need for frequent information update on web page. But in this web 4.0 era, trend for real time web page information update is becoming necessary. There are certain areas like share markets, stock price, foreign exchange, browser-based multi-player online gaming etc. where real-time communication is important [1]. store any state related information about client and hence HTTP is a stateless application layer protocol.

II. USE OF REAL-TIME COMMUNICATION

Now days, REST APIs (Representational State Transfer – Application Programming Interface) plays a role of middleware for the data exchange between client and server. Client requests to the server for data resource using unique identifier called URI (Uniform Resource Identifier). The request action is identified using normal HTTP methods viz. GET, PUT, POST and DELETE. The request – response cycle gets completed when server responds to the client for any of the above action. Now suppose resources present on server are updating frequently or based on some internal server event, then this becomes an asynchronous task. And clients can get these resource updates only when they put a fresh request to the server for those resources. For that purpose, client needs to constantly poll the server to get the periodic resource updates. But with this way, lot of unnecessary bandwidth/data gets consumed and client will get same response until there is any update at server resources. This also affects performance of the server. Remedy for this problem is what if server can push resource data to the registered/connected clients, whenever the resources get updated? This also helps to those web applications that are responsible to display real time information on web pages in the form of dashboards.

III. DIFFERENT WAYS OF ACHIEVING REAL-TIME DATA-TRANSFER

3.1 HTTP Polling

An early solution to server-initiated communication, known as HTTP polling, has been around for more than two decades. In HTTP polling, a client desiring near-real-time updates sends frequent HTTP requests to a server, which usually replies with a empty or baseline response. When an update becomes available, the client receives it from the server with a latency of roughly the time between requests. To further optimize this solution, HTTP offers the keep-alive header. This option allows the reuse of a TCP connection for multiple HTTP requests, reducing the overhead of creating a new TCP connection for each HTTP request. The problems with this approach are straightforward and well known. Even with persistent connections, the server cannot push updates to the client directly. While HTTP polling may be sufficient for applications where data arrives at known times, it will lead to many wasted client requests when servers need to update clients at inconsistent intervals. Each of these requests and responses contain HTTP headers, leading to significant amounts of wasted network traffic. In addition, updates from the server can only arrive at the frequency with which it is polled. An optimization on polling, called "long polling", uses the ability of servers to hold client queries open until data becomes available. This reduces the delay in delivering updates to clients, since the server will respond to the request at the moment when the data becomes available. Since requests need to be sent much less frequently (only when the server sends data or when a request times out), this solution offers a significant decrease in bytes on the wire.

3.2 Long Polling:
Unlike HTTP polling, in this technique, server holds the client connection open until new data is available, or threshold timeout occurs for that client. At client side, once the response is received, client needs to initiate a new request to the server to repeat this process. With this technique data loss may be possible if client was not able to receive the response from server. Also, one may need to consider performance, scaling, device support and fallbacks before using it

3.3 Server Sent Events:
Also abbreviated as SSE is one-way communication mechanism where data flows from server to client only. In this technique, server keeps the client connection open and sends the data in the form of streams. Though this technique is based on HTTP, not all browsers supports server sent events. Also, its implementation at server and client side is quite complex [3].

3.4 Web Socket:
It is an application layer protocol which provides a full duplex and persistent way of communication between client and server over a single underlying TCP connection. It's an upgrade over HTTP protocol. Web Socket keeps the connection open and allows the data/messages to be passed back and forth between client and server. This helps achieving a real-time data transfer to and from the server.

IV. WEBSOCKET

Web Socket acts as an upgrade over HTTP protocol which means, it uses same underlying TCP connection and creates channel between client and server for achieving full duplex, persistent way of communication. Like HTTP, web sockets also have their own set of protocols. It uses ws:// (Web Socket) and wss:// (Web Socket Secure), URI schemes to identify Web Socket connection, the rest of URI components are same as of generic syntax.

4.1 Web Socket Opening Handshake
Client can establish Web Socket connection with server through a process known as the Web Socket handshake. Fig (1) shows the initial handshake for establishment of Web Socket channel between client and server**.**

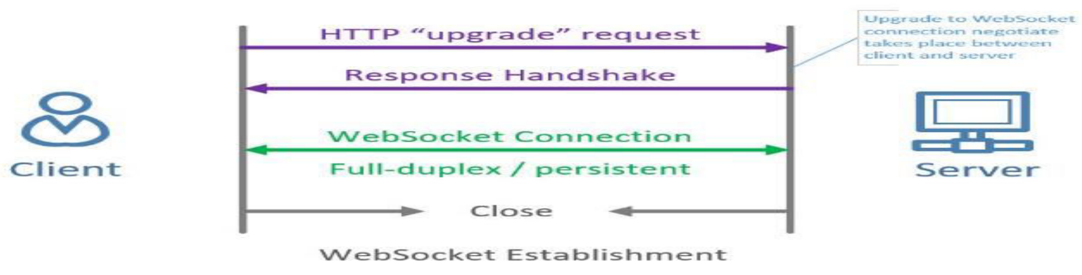
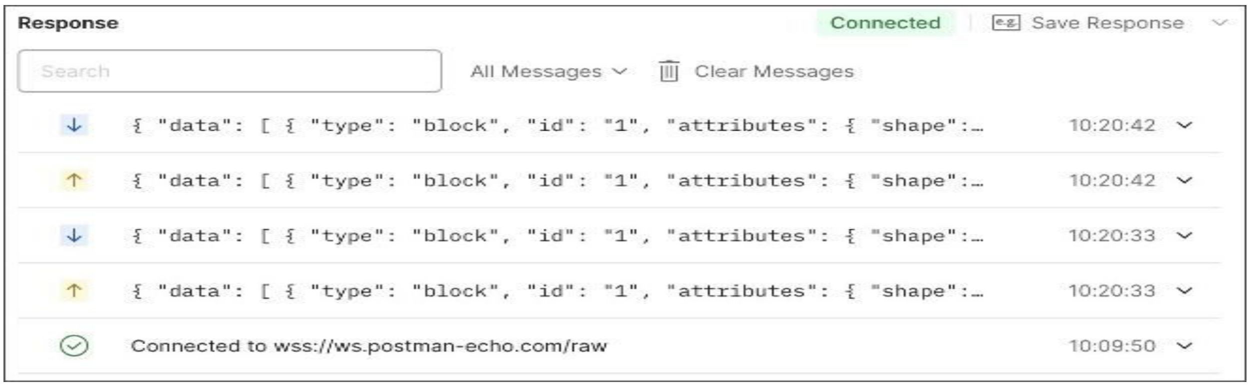


Fig : (1) Establishment of connection b/w client and server

Step 1: Client sends HTTP request to the server which includes connection upgrade header.
Step 2: If server supports Web Socket protocol, it responds back to client with connection upgrade response header.
Step 3: Initial HTTP connection gets replaced by Web Socket connection that uses the same underlying TCP/IP connection.
Step 4: Two-way data transfer takes place over newly established Web Socket channel. After successful establishment of Web Socket channel between server and client, server responds with 101 - HTTP response code. It stands for protocol switching i.e. initial HTTP connection gets upgraded to Web Socket connection.

4.2 Data-flow of Web Sockets

Web Socket is a framed protocol. The set of data sent over Web Socket called message is encoded in a frame. This frame includes various parameters like frame type, payload length, payload, masking key etc. These frames can be classified into 2 types viz. data frame and control frame. Data frame includes actual data to be sent between client and server. Control frame contains the control messages like ping/pong, which are required to maintain the connection between client and server. Normally server takes care of these control messages and transfers them periodically after certain interval. Fig () shows the data exchange between client and server, captured during Web Socket network analysis.



```
↓  { "data": [ { "type": "block", "id": "1", "attributes": { "shape":…   10:20:42
↑  { "data": [ { "type": "block", "id": "1", "attributes": { "shape":…   10:20:42
↓  { "data": [ { "type": "block", "id": "1", "attributes": { "shape":…   10:20:33
↑  { "data": [ { "type": "block", "id": "1", "attributes": { "shape":…   10:20:33
✓  Connected to wss://ws.postman-echo.com/raw                             10:09:50
```

Fig (2) Data Exchange between client and server

Web Socket is standardized by IETF as RFC6455, and its official documentation provides more details about the Web Socket frame .

4.3 Closing Handshake of Web socket

Both server and client can close the Web Socket connection by sending a closing frame. Receiving party must also send the close frame in response and no more data should be exchanged after that on the channel. This frame body may contain closing on code and reason for closing. Once closing frame is received by both the parties, underlying TCP connection gets turn off .

V. IMPLEMENTATION TECHNIQUES

There are many ways to implement Web Socket protocol at server and client side. Depending on requirement and some other development factors, one can chose right technique. Some of the techniques are as described below;
For server-side implementation of Web Socket, Python3 provides a generic Web Socket module. This module is based on AsyncIO programming which is officially supported by Python3.5+. Socket.IO, Tornado are some other frameworks based on python for implementing server side. Apart from that NodeJS, CPP - BOOST also gives some useful ways to implement Web Socket at server side

Now days, Web Socket technology is supported by all modern browsers. But one cannot directly check the information on browser by simply putting the Web Socket URL. For that purpose, Web Socket compatible client is required.

HTML5, JavaScript, Python gives a way to create Web Socket object which will automatically open the connection to Web Socket server.
Ex:

```
var ws = new websocket (url, protocol);
```

In this example, ws is the JavaScript Web Socket object. It takes Web Socket URL as a mandatory parameter. It then provides some methods like on_open(), on_close(), on_error(), on_message() etc. to capture Web Socket specific events. And one can implement business logic using these methods.

VI. RESULT AND ANALYSIS

Based on the implementation of Web Socket server on a system with dual-core processor and 6GB of physical memory, it is seen that server is able to sustain more than 1500 clients with one way data flow from server to client. Data size in this case is limited in a range of few KBs in the form of JSON format. Server side development part is based on python - AsyncIO [6]. For scalability and performance analysis, Chart (1) & (2) shows CPU & physical memory usage with respect to increasing number of active clients within fixed interval. This analysis is carried out using 'Artillery' – an open source load testing tool supported on Node.js 10.16.3 or later.

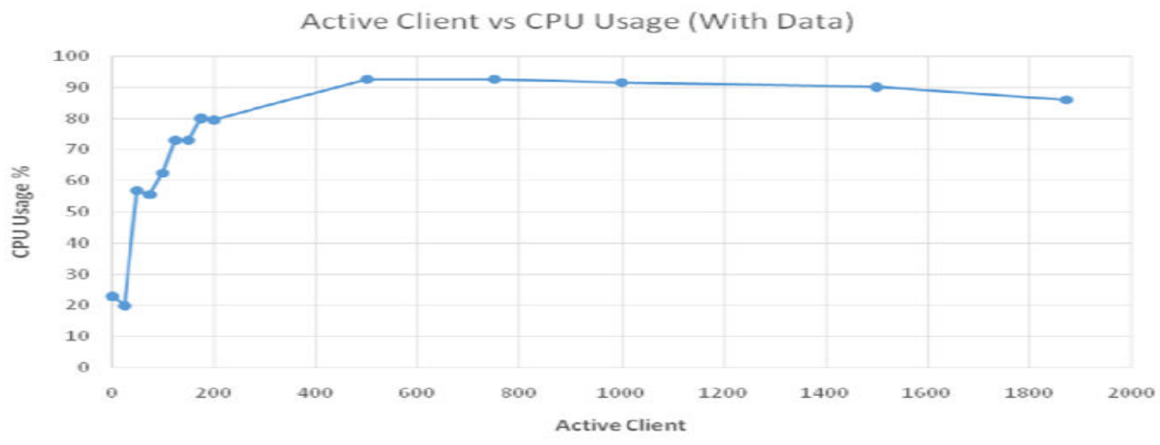


Chart 1: Active Client vs CPU Usage

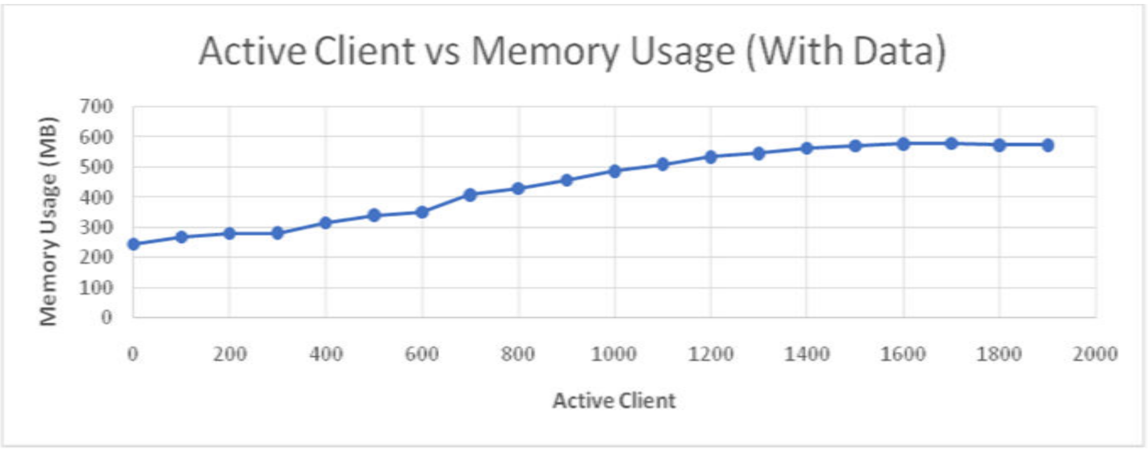


Chart – 2: Active Clients vs. Memory Usage

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VII. CONCLUSION

This paper presents the study of Web Socket protocol for real-time data exchange, its comparison with other technologies, implementation techniques etc. Web Socket cannot replace existing HTTP polling but can be considered as an upgrade for that. This technology can be used in practice, where there is necessity of real-time data update over the web. Web Socket technology also helps in reducing the network traffic and load on server as compared to other techniques.

REFERENCES

[1] D. G. Synodinos, "HTML 5 Web Sockets vs. Comet and Ajax," 2008. SciRes. http://www.infoq.com/news/2008/12/websockets-vs-com et-ajax
[2] Wikipedia, "WebSockets," 2010. http://en.wikipedia.org/wiki/WebSockets
[3] Peter Lubbers, "Pro HTML 5 Programming," Apress, Victoria, 2010.
[4] W3C, "The Web Sockets API," 2009. http://www.w3.org/TR/2009/WD-websockets-20091222
[5] D. Sheiko, "Persistent Full Duplex Client-Server Connec tion via Web Socket," 2010. http://dsheiko.com/weblog/persistent-full-duplex-client-ser ver-connection-via-web-socket
[6] Makoto, "Living on the Edge of the WebSocket Proto col," 2010. http://blog.new-bamboo.co.uk/2010/6/7/living-on-the-edge of-the-websocket-protocol
[7] P. Lubbers and F. Greco, "HTML5 Web Sockets: A Quan- tum Leap in Scalability for the Web," 2010. http://websocket.org/quantum.html, 2010.
[8] 2011. rfc6202: Known Issues and Best Practices for the use of Long Polling and Streaming in Bidirectional HTTP. https:// tools.ietf.org/html/rfc6202. (April 2011).
[9] 2011. rfc6455: The WebSocket Protocol. https://tools.ietf.org/ html/rfc6455. (December 2011).
[10] 2019. Amazon Simple Notification Service. https://aws.amazon. com/sns/. (September 2019).
[11] 2019. Can I Use: Push API. https://caniuse.com/#feat=push-api. (October 2019).
[12] 2019. Can I Use: Server-Sent Events. https://caniuse.com/#feat= server-sent-events. (October 2019).