

Study of WebSocket Protocol for Real-Time Data Transfer

Mr. Aniket Avinash Naik¹, Mrs. Meghana R Khare²

¹PG Student, Dept. of Electronics, Walchand College of Engineering, Sangli, Maharashtra, India

²Assistant Professor, Dept. of Electronics, Walchand College of Engineering, Sangli, Maharashtra, 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 WebSocket protocol. This paper presents the necessity of real-time data exchange for web applications, comparison of WebSocket with various application layer communication techniques like HTTP polling, Long polling, Server Sent Events (SSE) etc., analysis of WebSocket handshake and data flow. This paper also presents some of WebSocket implementation techniques at server and client side.

Key Words: WebSocket, Full duplex, Persistent, Real-time communication, HTTP, HTML5

1. 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 For a long time, web pages were static and needs to be updated manually when information gets outdated.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].

For any type of web applications, HTTP protocol is a way of achieving communication between client and server. Client first opens a port and makes a request to server for the required data. Server then responds to that request by sending the requested data to the client on that port. HTTP request headers have some client specific information with the help of which, server can respond to the appropriate client. During this process server doesn't

store any state related information about client and hence HTTP is a stateless application layer protocol.

2. NEED 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.

3. WAYS TO ACHIEVE REAL-TIME DATA TRANSFER

HTTP Polling: This is the simplest technique of achieving real-time communication. In this technique, client periodically makes XHR/AJAX requests to the server and server sends a response data for each request. The drawback of this technique is server needs to process large number of requests hence load on server is high. Also, until there are no any changes in resources, client will receive same response again and again.

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 [2].

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].

WebSocket: 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. WebSocket 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.

4. WEBSOCKET

WebSocket 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:// (WebSocket) and wss:// (WebSocket Secure), URI schemes to identify WebSocket connection, the rest of URI components are same as of generic syntax.

4.1 WebSocket Opening Handshake

Client can establish WebSocket connection with server through a process known as the WebSocket handshake. Fig (1) shows the initial handshake for establishment of WebSocket channel between client and server.

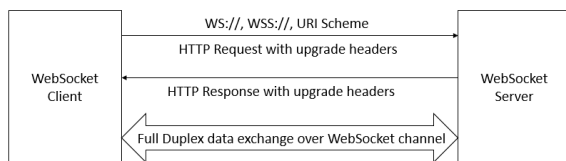


Fig – 1: WebSocket Connection Establishment

Step 1: Client sends HTTP request to the server which includes connection upgrade header.

Step 2: If server supports WebSocket protocol, it responds back to client with connection upgrade response header.

Step 3: Initial HTTP connection gets replaced by WebSocket connection that uses the same underlying TCP/IP connection.

Step 4: Two-way data transfer takes place over newly established WebSocket channel.

After successful establishment of WebSocket 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 WebSocket connection. Fig (2) and (3) shows the details of request and response headers captured during WebSocket network analysis.

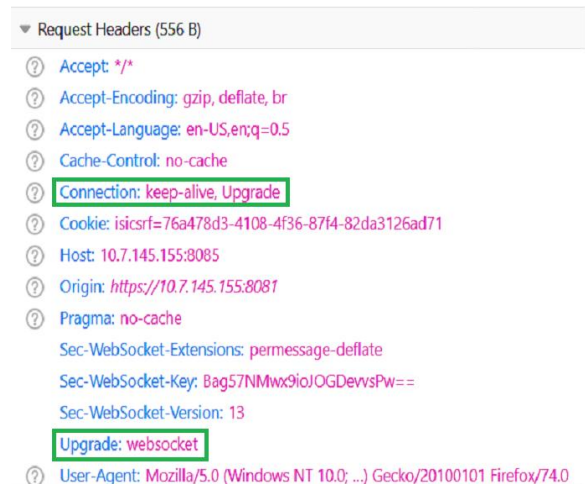


Fig – 2: WebSocket Request Headers

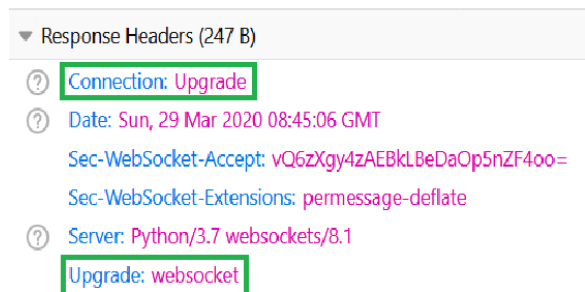


Fig – 3: WebSocket Response Headers

Following are some important parameters included in the headers;

Connection: Upgrade - Generally HTTP request use connection as 'keep-alive'. But for establishing WebSocket connection, 'Upgrade' parameter must be included.

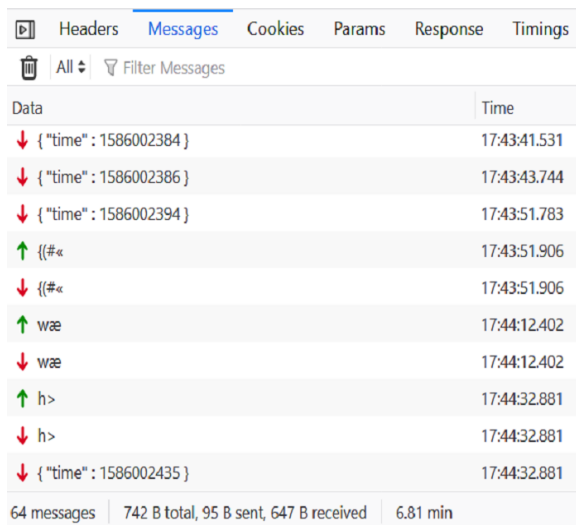
Upgrade: websocket - This parameter is used to ask the server, to switch the protocol to one of the listed protocols which include WebSocket.

Sec-WebSocket-Key - It is 16-byte random value generated by client with base64 encoding.

Sec-WebSocket-Version: 13 - This is the only accepted version of WebSocket protocol i.e. 13. Parameters with different value will result in invalid header.

4.2 WebSocket Data-flow

WebSocket is a framed protocol. The set of data sent over WebSocket 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 [4]. 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 (4) shows the data exchange between client and server, captured during WebSocket network analysis.



Data	Time
↓ {"time": 1586002384}	17:43:41.531
↓ {"time": 1586002386}	17:43:43.744
↓ {"time": 1586002394}	17:43:51.783
↑ {{#α}}	17:43:51.906
↓ {{#α}}	17:43:51.906
↑ wae	17:44:12.402
↓ wae	17:44:12.402
↑ h>	17:44:32.881
↓ h>	17:44:32.881
↓ {"time": 1586002435}	17:44:32.881

64 messages | 742 B total, 95 B sent, 647 B received | 6.81 min

Fig – 4: WebSocket Data-Flow

WebSocket is standardized by IETF as RFC6455, and its official documentation provides more details about the WebSocket frame [5].

4.3 WebSocket Closing Handshake

Both server and client can close the WebSocket 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 opcode and reason for closing. Once closing frame is received by both the parties, underlying TCP connection gets turn off [5].

5. IMPLEMENTATION TECHNIQUES

There are many ways to implement WebSocket 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 WebSocket, Python3 provides a generic WebSocket 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 WebSocket at server side.

Now days, WebSocket technology is supported by all modern browsers. But one cannot directly check the information on browser by simply putting the WebSocket URL. For that purpose, WebSocket compatible client is required. HTML5, JavaScript, Python gives a way to create WebSocket object which will automatically open the connection toWebSocket server.

Ex:

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

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

6. RESULTS & ANALYSIS

Based on the implementation of WebSocket 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.

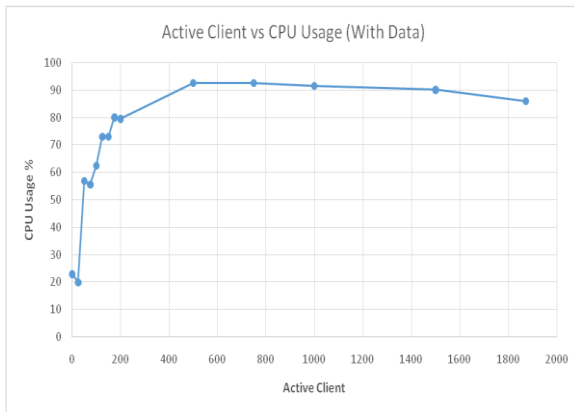


Chart – 1: Active Clients vs. CPU Usage

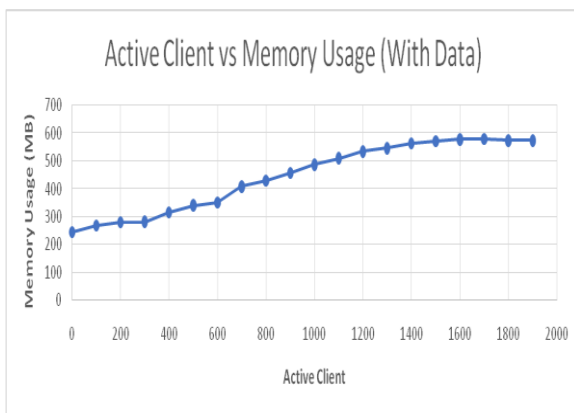


Chart – 2: Active Clients vs. Memory Usage

7. CONCLUSIONS

This paper presents the study of WebSocket protocol for real-time data exchange, its comparison with other technologies, implementation techniques etc. WebSocket 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. WebSocket technology also helps in reducing the network traffic and load on server as compared to other techniques.

REFERENCES

- [1] V. Pimentel and B. G. Nickerson, "Communicating and Displaying Real-Time Data with WebSocket," in IEEE Internet Computing, vol. 16, no. 4, pp. 45-53, July-Aug. 2012, doi: 10.1109/MIC.2012.64.
- [2] Shruti M. Rakhunde, "Real Time Data Communication over Full Duplex Network Using Websocket", IOSR Journal of Computer Science (IOSR-JCE), PP 15-19.
- [3] Server-sent events.
https://en.wikipedia.org/wiki/Server-sent_events
- [4] D. Skvorc, M. Horvat, and S. Srblijic, "Performance Evaluation of Websocket Protocol for Implementation of Full-Duplex Web Streams", MIPRO 2014, 26-30 May 2014.
- [5] <https://tools.ietf.org/html/rfc6455>
- [6] Python websockets documentation.
<https://websockets.readthedocs.io/en/stable/>
- [7] WebSocket in HTML5.
<http://en.wikipedia.org/wiki/WebSocket>.