WEBSOCKET

A solution for real time web applications

Bhupendra Pratap Singh
CDAC, ACTS, Pune

INTRODUCTION

- Web sockets are defined as a two-way communication between the servers and the clients, which mean both the parties communicate and exchange data at the same time.
- Provides **True concurrency** and **optimization of performance**, resulting in more responsive and rich web application.
- The latest specification of Web Socket protocol is defined as **RFC 6455** – a proposed standard.

CONT....

- With this technology, a user can send messages to a server and receive event-driven responses without requiring long-polling, i.e. without having to constantly check the server for a reply.
- Websocket allow both the server and the client to push messages at any time without any relation to a previous request.
- One notable advantage in using websocket is, **almost every browser support websocket**. **RFC 6455** is supported by various browsers like Internet Explorer, Mozilla Firefox, Google Chrome, Safari, and Opera

CONT..

- The Websocket connection uses the same ports as HTTP (80) and HTTPS (443), by default.
- Lower Latency.
- Bi-Directional Communication.
- Persistent connection between client and server.
- Provides a two-way communication (full duplex) over a single TCP connection

WHY WEBSOCKETS

- **(LET US UNDERSTAND BRIEF HISTORY OF REAL TIME WEB APLLICATIONS FIRST)**
- The web was built around the idea that a client's job is to request data from a server, and a server's job is to fulfill those requests.
- This paradigm went unchallenged for a number of years but with the introduction of AJAX around 2005 many people started to explore the possibilities of making connections between a client and server *bidirectional.*

CONT...

- To allow servers to push data to the client. One of the most popular strategies was long-polling on those days (till 2005 still prevail in HTTP).
- This involves keeping an HTTP connection open until the server has some data to push down to the client
- The problem with all of these solutions is that they carry the overhead of HTTP.

CONT....

- Every time you make an HTTP request a bunch of headers and cookie data are transferred to the server. This can add up to a reasonably large amount of data that needs to be transferred, which in turn **increases *latency***.
- Creating a persistent, low latency connection that can support transactions initiated by either the client or server. This is exactly what WebSockets provides.

HOW WEBSOCKET WORKS

- WebSockets provide a persistent connection between a client and server that both parties can use to start sending data at any time.
- The client establishes a WebSocket connection through a process known as the WebSocket handshake.
- This process starts with the client sending a regular HTTP request to the server. An Upgrade header is included in this request that informs the server that the client wishes to establish a WebSocket connection.

SIMPLIFIED EXAMPLE OF THE INITIAL REQUEST HEADERS

GET ws://websocket.example.com/ HTTP/1.1

Origin: http://example.com

Connection: Upgrade

Host: websocket.example.com

Upgrade: websocket

POINT to NOTE: Note: WebSocket URLs use the ws scheme. There is also wss for secure WebSocket connections which is the equivalent of HTTPS.

CONT...

- If the server supports the WebSocket protocol, it agrees to the upgrade and communicates this through an Upgrade header in the response.
- Examples :

HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 30 Oct 2018 10:07:34 PM IST
Connection: Upgrade
Upgrade: WebSocket

HTTP VS WEBSOCKET

- HTTP is the underlying communication protocol of World Wide Web. **HTTP functions as a request–response protocol in the client–server computing model.** HTTP/1.1 is the most common version of HTTP used in modern web browsers and servers.
- Most of the modern improvements to HTTP rely on these two headers.
- Keep-Alive header to set policies for long-lived communications between hosts (timeout period and maximum request count to handle per connection)
- Upgrade header to switch the connection to an enhanced protocol mode such as HTTP/2.0 (h2,h2c) or Websockets (websocket)

HTTP POLLING

- The client polls the server requesting new information by adhering to one of the below mechanism (Mentioned in next slides in detail).
- Polling is used by the vast majority of applications today and most of the times goes with RESTful practices. In practice, HTTP Short Polling is very rarely used and HTTP Long Polling or Periodic Polling is always the choice.

HTTP SHORT POLLING

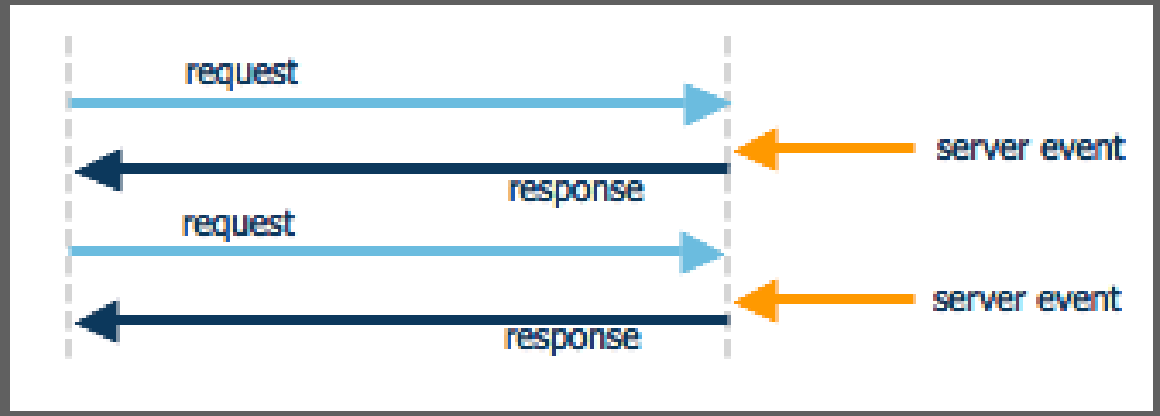
- Simpler approach. A lot of requests are processed as they come to server, creating a lot of traffic (uses resources, but frees them as soon as response is sent back). Since each connection is only open for a short period of time, many connections can be time-multiplexed.

```
00:00:00 C-> Is the cake ready?
00:00:01 S-> No, wait.
00:00:01 C-> Is the cake ready?
00:00:02 S-> No, wait.
00:00:02 C-> Is the cake ready?
00:00:03 S-> Yeah. Have some lad.
00:00:03 C-> Is the other cake ready?
```

HTTP LONG POLLING

- One request goes to server and client is waiting for the response to come. The server holds the request open until new data is available (it's unresolved and resources are blocked). Client is notified with no delay when the server event happens. More complex and more server resources used.

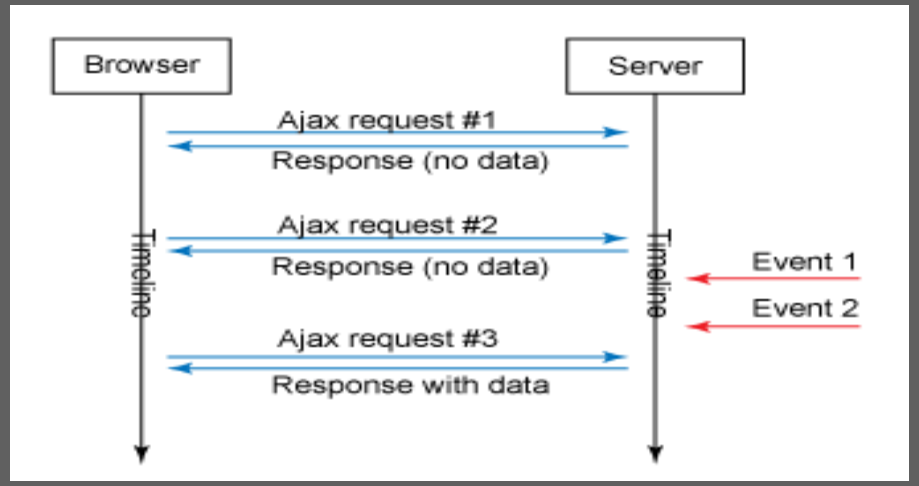
```
12:00 00:00:00 C-> Is the
cake ready?
12:00 00:00:03 S-> Yeah.
Have some lad.
12:00 00:00:03 C-> Is the
other cake ready?
```



HTTP Long Polling—Response is held until server process data (Image from shyamapadabatabyal.wordpress.com)

HTTP PERIODIC POLLING

- There's a predefined time gap between two requests. This is an improved/managed version of polling. You can reduce server consumption by increasing time gap between two requests. But if you need to be notified with no delay when the server event happens, this is not a good option.




```
00:00:00 C-> Is the cake
ready?
00:00:01 S-> No, wait.
00:00:03 C-> Is the cake
ready?
00:00:04 S-> Yeah. Have
some lad.
00:00:06 C-> Is the other
cake ready?
```

WEBSOCKETS

- WebSockets allow both the server and the client to push messages at any time without any relation to a previous request.
- **WebSocket solves a few issues with HTTP.**
- **Bi-directional protocol**—either client/server can send a message to the other party (In HTTP, the request is always initiated by client and the response is processed by server—making HTTP a uni-directional protocol)

CONT...

- **Full-duplex communication**—client and server can talk to each other independently at the same time.
- **Single TCP connection**—After upgrading the HTTP connection in the beginning, client and server communicate over that same TCP connection throughout the lifecycle of Websocket connection.

EXAMPLES

- 00:00:00 CLIENT-> I need cakes
- 00:00:01 SERVER-> Wait for a moment.
- 00:00:01 CLIENT-> Okay, cool.
- 00:00:02 SERVER-> Have cake-1.
- 00:00:02 SERVER-> Wait for cake-2.
- 00:00:03 CLIENT-> What is this flavor?
- 00:00:03 SERVER-> Don't you like it?
- 00:00:04 SERVER-> Have cake-2.
- 00:00:04 CLIENT-> I like it.
- 00:00:05 CLIENT-> But this is enough.

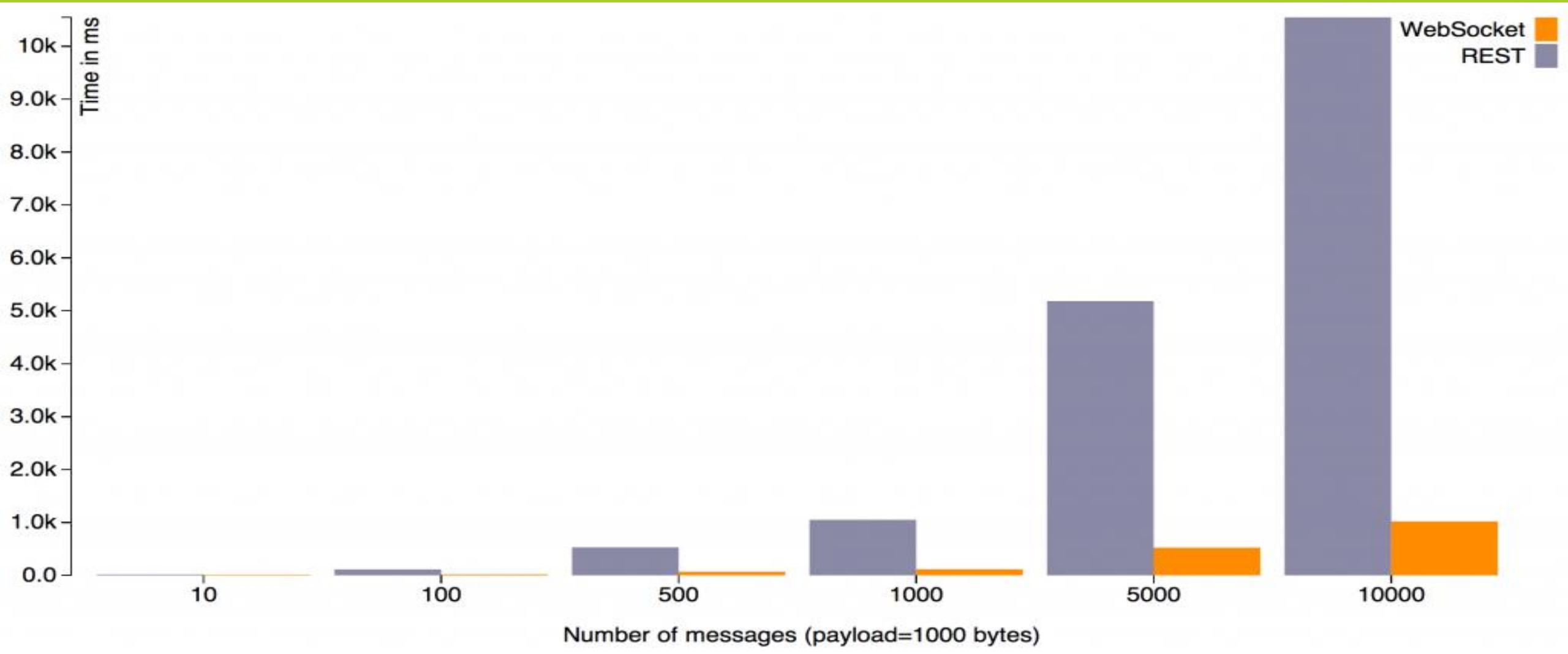
COMPARISON AND BENCHMARKS

(REFERENCE: HTTP://BLOG.ARUNGUPTA.ME/REST-VS-WEBSOCKET-COMPARISON-BENCHMARKS/)

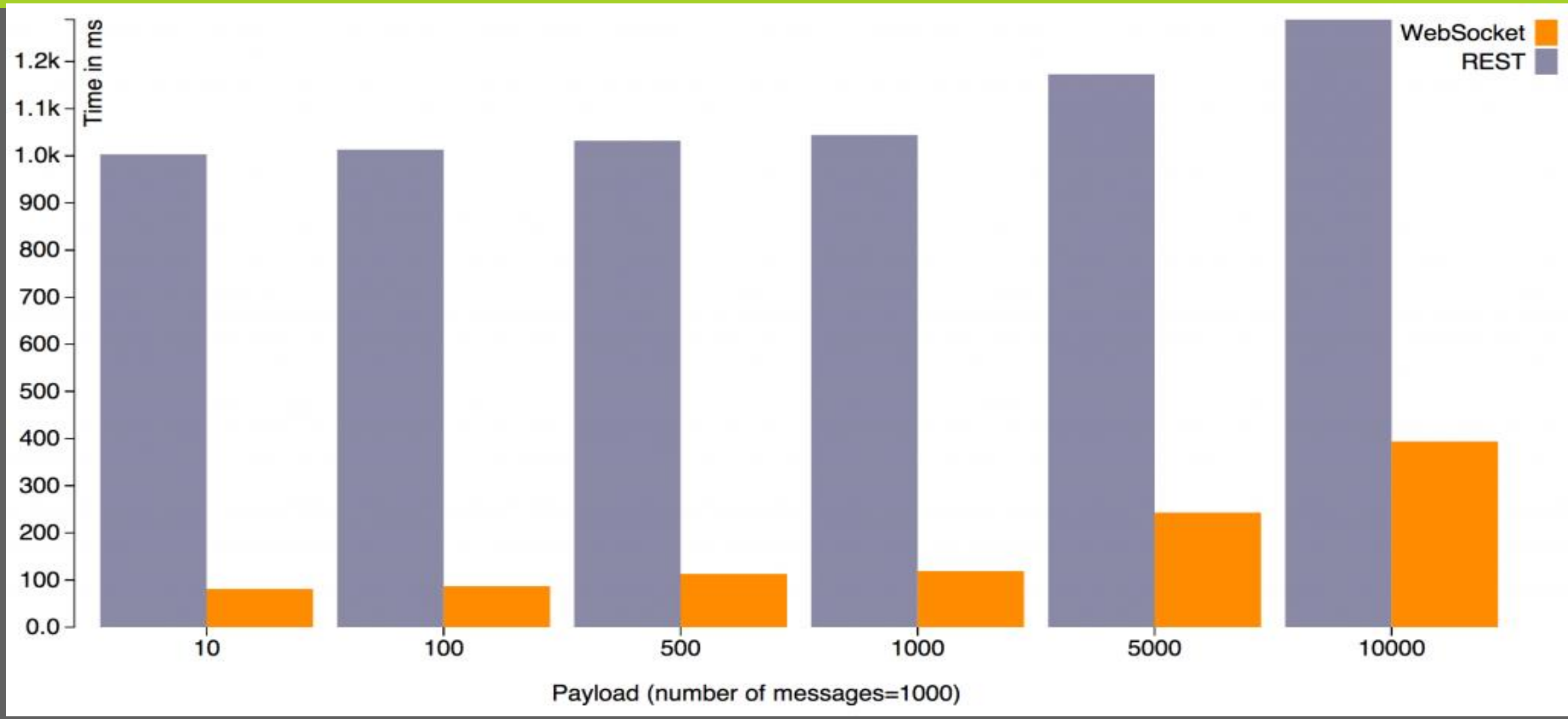
Constant payload, increasing number of messages			
Messages	REST (in ms)	WebSocket (in ms)	x times
10	17	13	1.31
100	112	20	5.60
500	529	68	7.78
1000	1050	115	9.13
5000	5183	522	9.93
10000	10547	1019	10.35

This table shows that the REST overhead increases with the number of messages. This is true because that many TCP connections need to be initiated and terminated and that many HTTP headers need to be sent and received.

GRAPHICAL REPRESENTATION



GRAPHICAL REPRESENTATION



"I never dreamt of success. I worked for it."

– Este Lauder