How Websockets are implemented?

Ask Question

13

11

- How Websockets are implemented?
- What is the algorithm behind this new tech (in comparison to Long-Polling)?
- How can they be better than Long-Polling in term of performance?

I am asking these questions because here we have a sample code of Jetty websocket implementation (server-side).

> If we wait long enough, a timeout will occur, resulting in the following message on the client.

And that is definately the problem I'm facing when using Long-polling. It stops the process to prevent server overload, doesn't it ?

javascript websocket

edited Apr 15 '18 at 14:37

asked Jan 11 '16 at 20:19

David
2,349 3 22 43

Please explain how exactly your question is on topic? You have not given any actual problem - if you knew how websockets were implemented, then what would this solve for you? And asking `better ... in terms of performance` makes the question too broad as well. – JK. Jan 12 '16 at 0:32

Please note I just updated my question. – David Jan 13 '16 at 9:03

2 Answers

23

> How Websockets are implemented?

webSockets are implemented as follows:

1. Client makes HTTP request to server with "upgrade" header on the request
2. If server agrees to the upgrade, then client and server exchange some security credentials and the protocol on the existing TCP socket is switched from HTTP to webSocket.
3. There is now a lasting open TCP socket connecting client and server.
4. Either side can send data on this open socket at any time.
5. All data must be sent in a very specific webSocket packet format.

Because the socket is kept open as long as both sides agree, this gives the server a channel to "push" information to the client whenever there is something new to send. This is generally much more efficient than using

client-driven Ajax calls where the client has to regularly poll for new information. And, if the client needs to send lots of messages to the server (perhaps something like a mnulti-player game), then using an already open socket to send a quick message to the server is also more efficient than an Ajax call.

Because of the way webSockets are initiated (starting with an HTTP request and then repurposing that socket), they are 100% compatible with existing web infrastructure and can even run on the same port as your existing web requests (e.g. port 80 or 443). This makes cross-origin security simpler and keeps anyone on either client or server side infrastructure from having to modify any infrastructure to support webSocket connections.

> What is the algorithm behind this new tech (in comparison to Long-Polling)?

There's a very good summary of how the webSocket connection algorithm and webSocket data format works here in this article: Writing WebSocket Servers.

> How can they be better than Long-Polling in term of performance?

By its very nature, long-polling is a bit of a hack. It was invented because there was no better alternative for server-initiated data sent to the client. Here are the steps:

1. The client makes an http request for new data from the client.
2. If the server has some new data, it returns that data immediately and then the client makes another http request asking for more data. If the server doesn't have new data, then it just hangs onto the connection for awhile without providing a response, leaving the

request pending (the socket is open, the client is waiting for a response).

3. If, at any time while the request is still pending, the server gets some data, then it forms that data into a response and returns a response for the pending request.
4. If no data comes in for awhile, then eventually the request will timeout. At that point, the client will realize that no new data was returned and it will start a new request.
5. Rinse, lather, repeat. Each piece of data returned or each timeout of a pending request is then followed by another ajax request from the client.

So, while a webSocket uses one long-lived socket over which either client or server can send data to the other, the long-polling consists of the client asking the server "do you have any more data for me?" over and over and over, each with a new http request.

Long polling works when done right, it's just not as efficient on the server infrastructure, bandwidth usage, mobile battery life, etc...

> What I want is explanation about this: the fact Websockets keep an open connection between C/S isn't quite the same to Long Polling wait process? In other words, why Websockets don't overload the server?

Maintaining an open webSocket connection between client and server is a very inexpensive thing for the server to do (it's just a TCP socket). An inactive, but open TCP socket takes no server CPU and only a very small amount of memory to keep track of the socket. Properly configured servers can hold hundreds of thousands of open sockets at a time.

On the other hand a client doing long-polling, even one for which there is no new information to be sent to it, will have to regularly re-establish its connection. Each time it re-establishes a new connection, there's a TCP socket teardown and new connection and then an incoming HTTP request to handle.

Here are some useful references on the topic of scaling:

- 600k concurrent websocket connections on AWS using Node.js
- Node.js w/1M concurrent connections!
- HTML5 WebSocket: A Quantum Leap in Scalability for the Web
- Do HTML WebSockets maintain an open connection for each client? Does this scale?

edited Apr 28 at 4:03

Lucia
8,657 5 28 41

answered Jan 12 '16 at 0:16

jfriend00
446k 56 588 633

Thank you. The last part of your answer is what I as looking for. But can you detail with external sources please ? – David Jan 13 '16 at 9:08

@David - added some references to my answer. – jfriend00 Jan 13 '16 at 16:21

So, if I understand your answer, there is no effect on client performances, like for example, on mobile battery ? – Emrys Myrooin Feb 11 '16 at 16:17

3

Very good explanation about web sockets, long polling and other approaches:

[In what situations would AJAX long/short polling be preferred over HTML5 WebSockets?]

Long poll - *request* → *wait* → *response*. Creates connection to server like AJAX does, but keep-alive connection open for some time (not long though), during connection open client can receive data from server. Client have to reconnect periodically after connection is closed due to timeouts or data eof. On server side it is still treated like HTTP request same as AJAX, except the answer on request will happen now or some time in the future defined by application logic. Supported in all major browsers.

WebSockets - *client* ↔ *server*. Create TCP connection to server, and keep it as long as needed. Server or client can easily close it. Client goes through HTTP compatible handshake process, if it succeeds, then server and client can exchange data both directions at any time. It is very efficient if application requires frequent data exchange in both ways. WebSockets do have data framing that includes masking for each message sent from client to server so data is simply encrypted. support chart (very good)

Overall, sockets have much better performance than long polling and you should use them instead of long polling.

edited May 23 '17 at 11:33

Community ♦

1 1

answered Jan 11 '16 at 20:25

Mihail Petkov
1,342 8 10

Yep, I've already seen this post before. But it doesn't really answer my question. What I want is explanation about this: the fact Websockets keep an open connection between C/S isn't quite the same to Long Polling `wait` process? In other words, why Websockets don't overload the server? – David Jan 11 '16 at 20:29

@David - WebSocket servers are built with large numbers of persistent connections in mind, while (traditional) HTTP servers are geared towards handling requests quickly. – gzost Jan 13 '16 at 13:50