Understanding Websockets In depth

vishal rana

WebSockets are a powerful communication protocol that enables bidirectional, real-time communication between a client and a server over a single TCP connection. In this blog, we'll cover everything that you need to know about web sockets.

Web sockets allow us to create "real-time" applications which are faster and require less overhead then TRADITIONAL API PROTOCOL

Web socket uses a `DUPLEX` protocol for communication between client and server.

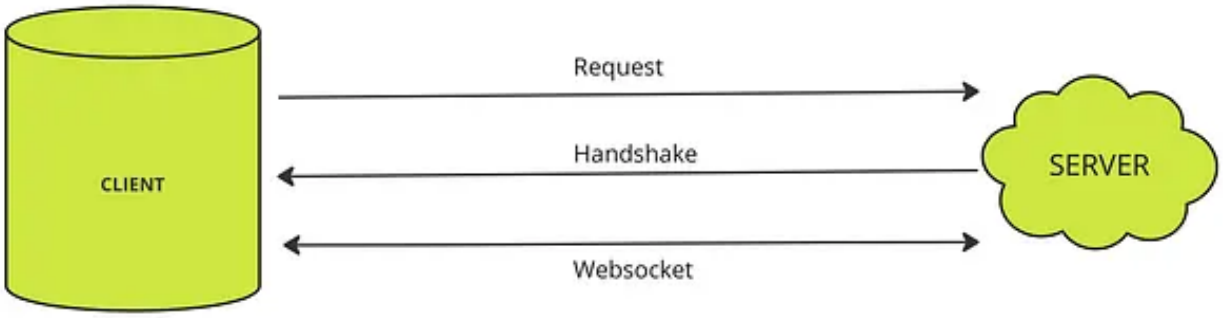
Here are some of the key points of Websocket:

- Web-sockets are bi-directional in nature.
- Connection developed using the web socket lasts as long as any of the participant(client or server) closes the connection.
- Web-socket makes use of HTTP to initiate connection.

WHY WEB-SOCKET IS BEING USED?

Websockets are reliable and easy to use and are act as a boon in the systems where real time nature is requested. Let say you are working on an application, where you need to display data in real-time that is changing continuously. `You first choice should be implementation of web sockets, There are other way as well, but that will increase the overhead on your servers and eventually you have to find some other reliable way to do it.`

So in order to save you that overhead, you can consider websockets as a choice.



Once the handshake between client and server is established, connection remain persistent and last until any parts disconnects (**WEBSOCKET**)

Every request with result in new TCP connection and after the response is received , connection will be terminated(**HTTP**)

A general use case for websockets can be :

- Stock exchange (for real time updates on ticker prices, order book, etc)
- Chat application (as Sending and receiving message should be quick)
- Web application where real time changes are required
- Gaming industry
- and much more

HOW WEB SOCKET WORKS?

Websocket works by making a connection between client and server. In

order to start the connection, clients sends a http GET request with a upgrade header, so that the server know that this is a upgrade request and it responds with status 101 if the server supports the upgrade properties and return error code if not.

If any code other than 101 is returned from the server, Clients has to end the connection.

Websockets are consider as they make a single connection between client and server and there is no overhead of making those handshakes with the server every time we have to communicate.

Connection Setup in Web sockets

Connection Setup starts with a request initiated from the client for the handshake.

Client initiate the handshake

There are certain headers that client needs to send , in order to tell server that we want to create a websocket connection. As the request is make using http GET method.

Client side request headers look like this:

Header fields in the handshake may be sent by the client in any order.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

- We are making a `GET` Request over http. (version needs to be greater or equals to 1.1)
- `Host` name is added in the header so that both the client and server verify that they agree on which host is in use.
- The `Connection` and `Upgrade` header fields complete the HTTP Upgrade.
- `Sec-WebSocket-Protocol` is send by the client to specify which protocol to use (OPTIONAL)
- `Sec-WebSocket-Version` is send to indicate what subprotocols (application-level protocols layered over the WebSocket Protocol) are acceptable to the client(OPTIONAL)
- `Origin` header is used to protect against Unauthorised cross-origin use of a WebSocket server by scripts using the WebSocket API in a web browser. Server will only accept connections from listed origins.
- `Sec-WebSocket-key` is the base64 encoded value that is generated by randomly selecting 16-byte value as a nonce.

When the server receive these request, it has to validate and respond to the client that it has received the handshake request and they can form a connection.

Server will do following things

- In order to prove that handshake was received, server has to take the `Sec-WebSocket-key` from the request header , combine it with the `Globally Unique Identifier` , create a SHA-1 hash of this

concatenation string.
- Then it encodes that string using base64 and return as server handshake.

The handshake from the server looks like this:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

- Responds with a `101 status code`, any code other than 101 results in error and means that websocket handshake was not completed.
- The `Connection` and `Upgrade` field indicate the HTTP upgrade.
- `Sec-WebSocket-Accept` indicates whether the server is willing to accept the connection request or not. if this field is present, this field contains the base64 encoded hash of `Sec-WebSocket-key` and `Globally Unique Identifier` combined.
- `Sec-WebSocket-Protocol` is an optional field, it indicates which protocol is selected by the server for communication.

These fields are checked by the WebSocket client for scripted pages.If the `Sec-WebSocket-Accept` value does not match the expected value, if the `header field` is missing, or if the `HTTP status code is not 101`, the connection will not be established, and WebSocket frames will not be sent.

Websocket has a default URI format

```
ws-URI = "ws:" "//" host [ ":" port ] path [ "?" query ]
wss-URI = "wss:" "//" host [ ":" port ] path [ "?" query ]
```

Port component is optional as default port 80 is used for `ws` and 443 is used for `wss` .

`wss is used as a secure URI` as secure flag is set and TLS handshake is done between server and client for secure communication.

WEBSOCKET FRAMES

Frames in WebSocket are the basic units of data that are exchanged between the client and server. In web-socket protocol, data is transmitted using a sequence of frames. Client must mask the frames before sending it to the server and if an unmasked frame is received by the server, the server should close the connection.

> *In this case, a server MAY send a Close frame with a status code of 1002 (protocol error).*

Websocket Frame

- FIN (1 bit) — Indicates if this is the final fragment in a message or not. Value 1 represent Yes. First fragment can also be the final fragment
- `RSV1`, `RSV2, RSV3` (1 bit each) — This must be a zero value. If this is a non-zero value, it has to define an extension that define the meaning of non-zero value.
- `Opcode(4 bits)` — It define the payload data, it must have one of the following value.

```
%x0 denotes a continuation frame

*  %x1 denotes a text frame

*  %x2 denotes a binary frame

*  %x3-7 are reserved for further non-control frames

*  %x8 denotes a connection close

*  %x9 denotes a ping

*  %xA denotes a pong
```

```
*  %xB-F are reserved for further control frames
```

- `Mask (1 bit)` — It defines if marking is used or not, if set to 1, it must have a masked key define as well. which will be used for unmasking the “Payload data”, every client should send the mark value to 1 and masking-key as well.
- `Masking-key(0 or 4 bytes)` — Every frame sent from the client is masked using this key. Masked keys are 32 bit values.
- `Payload length (7 bits, 16 bits, 64 bits)` — The length of the “Payload data”, in bytes: if 0–125, that is the payload length. If 126, the following 2 bytes interpreted as a 16-bit unsigned integer are the payload length. If 127, the following 8 bytes interpreted as a 64-bit unsigned integer (the most significant bit MUST be 0) are the payload length.
- `Payload Data` — The “Payload data” is defined as “Extension data” concatenated with “Application data.

> *The smallest valid **full** WebSocket message is two byte i.e- close message sent by the server with no message.*
>
> *The longest possible header is 14 bytes for a client-to-server message with a payload larger than 16KB: 8+1 bytes for the length and 4 bytes for the mask (plus the first fin/type byte).*

FRAGMENTS

When sending a message in WebSocket, it is common for the message to be split up into multiple frames, especially if the message is large or complex. **This is where fragments come in**. Fragments allow a message to be split up into smaller pieces, each of which is sent as a separate frame.

When a message is split up into fragments, each fragment is sent with the FIN bit set to 0 for all but the final frame in the sequence. The FIN bit

indicates whether the current frame is the final frame in the message or whether more frames will follow.

If the FIN bit is set to 0, it means that there are more frames coming and the receiver should continue to wait for additional frames before processing the message.

When the final fragment of a message is sent, it is sent with the FIN bit set to 1. This signals to the receiver that this is the last frame in the message.

Conceptually, WebSocket is really just a layer on top of TCP that does the following:

- *Adds a web origin-based security model for browsers.*
- *Adds an addressing and protocol naming mechanism to support multiple services on one port and multiple host names on one IP address*
- *Layers a framing mechanism on top of TCP to get back to the IP packet mechanism that TCP is built on, but without length limits*
- *Includes an additional closing handshake in-band that is designed to work in the presence of proxies and other intermediaries*

HOW WEB SOCKET DIFFERENT THEN HTTP?

Http uses distinct connection for separate request. It increase the load on server as server has to create a new handshake for every request. Once a request is completed, Connection is closed. On the other hand, web-socket connection is persistent as long as ,not interrupted by either of the parties.

> The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.

> *It's also designed in such a way that its servers can share a port with*

> *HTTP servers, by having its handshake be a valid HTTP Upgrade request.*

The WebSocket Protocol is designed on the principle that there should be **minimal framing** (the only framing that exists is to make the protocol frame-based instead of stream-based and to support a distinction between Unicode text and binary frames).

It is expected that **metadata would be layered on top of WebSocket** by the application layer, in the same way that **metadata is layered on top of TCP** by the application layer (e.g., HTTP).

> *By default, the WebSocket Protocol uses port 80 for regular WebSocket*
> *connections and port 443 for WebSocket connections tunneled over Transport Layer Security (TLS)*

Conclusion

Websockets are very powerful when it comes to bi-directional communication and read time updation with minimum overhead. Fragmentation makes it even more powerful and light weight. You should understand web socket protocol and what are the underline priniciples of it and how it works.

I hope this blog, gives you an idea of how it works!

Happy reading.