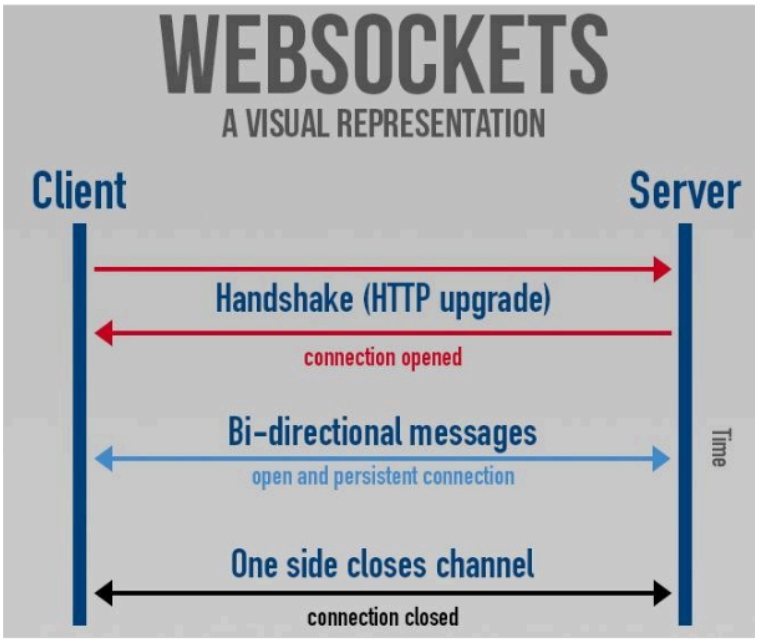
WEB SOCKET

Il WebSocket è un protocollo di livello applicativo che fornisce un canale di comunicazione bidirezionale simmetrico attraverso una singola connessione TCP inizialmente utilizzata per il protocollo HTTP che ha un comportamento bidirezionale ma asimmetrico. Il protocollo WebSocket permette maggiore interazione tra un browser e un server, facilitando la realizzazione di applicazioni web che devono fornire contenuti in tempo reale. Questo è reso possibile poichè i WebSocket permettono al server la possibilità di “prendere l’iniziativa” ed effettuare dei push autonomi di dati verso il browser per aggiornarlo, cosa che non può avvenire con il tradizionale protocollo HTTP.

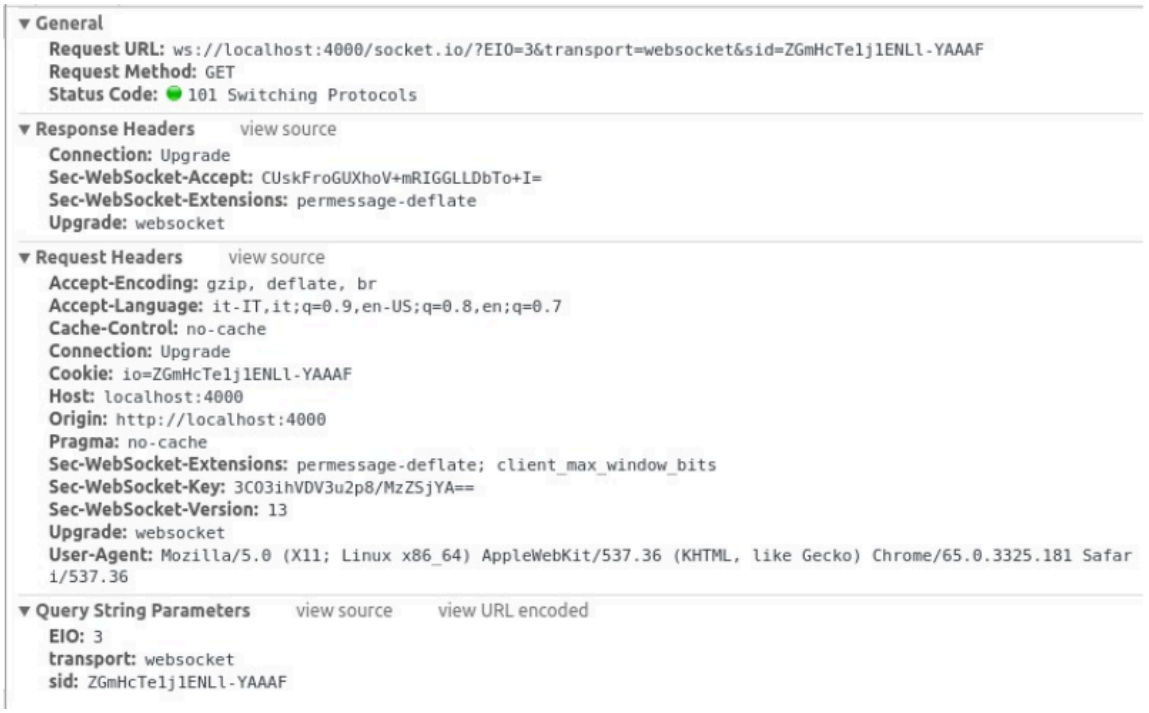


Un sistema bidirezionale (detto anche full-duplex) permette la comunicazione in entrambe le direzioni simultaneamente. Una buona analogia per il full-duplex potrebbe essere una strada a due corsie con una corsia per ogni direzione. Una connessione TCP è bidirezionale non solo come direzione dei dati ma anche come libertà di entrambi gli agenti coinvolti di trasmettere per primi. Questa possibilità è poi persa nel protocollo HTTP dove è sempre il client a fare il primo passo creando un comportamento asimmetrico. Con i WebSocket si vuole far ritornare simmetrico il rapporto tra i due interlocutori come se fosse una connessione TCP di tipo base col vantaggio che è stata instaurata inizialmente per l’HTTP e quindi è compatibile con molte politiche di sicurezza già presenti su Internet.

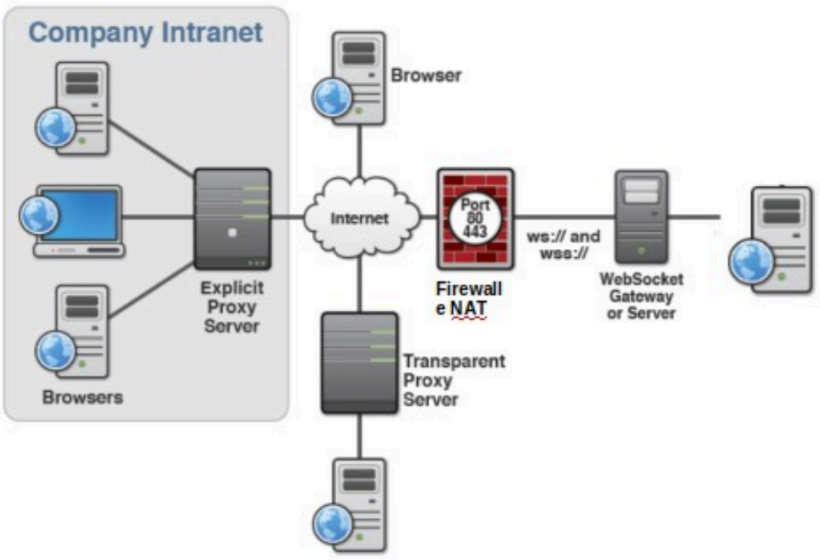
I WebSocket sono basati sul protocollo TCP e nascono da una connessione HTTP attraverso una Upgrade request verso il server. Per permettere una compatibilità iniziale tra browser e server, si utilizza l ’HTTP Upgrade header’ in cui viene richiesto di passare dal

protocollo HTTP a quello WebSocket. Il protocollo HTTP fornisce un meccanismo speciale che permette di stabilire un upgrade del protocollo da usare durante la connessione. Questo meccanismo può essere inizializzato solo dal client, mentre il server, se è in grado di fornire il servizio col protocollo stabilito dal client, decide se accettare o meno questo cambio di protocollo.

Ispezione dell'Upgrade request avvenuto tra browser e server visualizzata attraverso la console di sviluppo del browser (FIGURA):



Il protocollo WebSocket è supportato attualmente da numerosi browser, inclusi Google Chrome, Internet Explorer, Edge, Firefox, Safari e Opera. Diamo un'occhiata a come i WebSocket interagiscono con gli apparati di rete. Le connessioni WebSocket utilizzano porte HTTP standard (80 e 443). Pertanto, i WebSocket non richiedono l'apertura di nuove porte sulle reti e quindi sono compatibili con le impostazioni di sicurezza dei firewall, proxy web e il meccanismo del NAT. Un'immagine vale più di mille parole. La Figura successiva mostra una topologia di rete semplificata in cui il browser accede ai servizi tramite la porta 80 (o 443) utilizzando una connessione WebSocket full-duplex. Alcuni client si trovano all'interno di una rete aziendale, protetti da un firewall aziendale e configurati per accedere a Internet tramite server proxy che possono fornire memorizzazione e sicurezza del contenuto. A differenza del normale traffico HTTP, che utilizza un protocollo di richiesta/risposta, le connessioni WebSocket possono rimanere aperte per lungo periodo e permettono un scambio paritetico di dati tra browser e server. I firewall, NAT e Proxy eventualmente interposti devono consentire questo tipo di comunicazioni.



Limitazioni:

I WebSockets non rappresentano la soluzione a tutto. L'HTTP riveste ancora un ruolo chiave nella comunicazione tra browser e server come via per inviare e chiudere connessioni per trasferimenti di dati di tipo one-time, come i caricamenti iniziali. Le richieste HTTP sono in grado di eseguire questo tipo di operazioni in modo più efficiente dei WebSocket, chiudendo le connessioni una volta utilizzate piuttosto che mantenendo lo stato della connessione.
Inoltre i WebSocket possono essere utilizzati solo se gli utenti utilizzano i moderni browser con JavaScript abilitato. Questo potrebbe non essere vero in caso di sistemi embedded. Dovrebbero poi essere considerati possibili impatti sull'architettura di rete. WebSocket, essendo una connessione persistente, potrebbe richiedere molte più risorse rispetto ad un server Web standard. Per capire meglio come funziona questa tecnologia si è implementato un breve tutorial in cui si vuole sviluppare una chat online utilizzando la tecnologia offerta dai WebSocket.

Backend e Frontend nello specifico:

L'utilizzo dei WebSocket in un esempio di programmazione event-driven viene mostrato attraverso l'implementazione di una semplice chat in cui ciascun utente, attraverso il proprio client collegato al server, è in grado di mandare messaggi a tutti gli altri. Per la realizzazione di questa chat verranno usati vari strumenti software:

- JavaScript: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di siti web ed applicazioni web;
- Node.js: framework event-driven per realizzare applicazioni Web in JavaScript che ci permette di utilizzare questo linguaggio, tipicamente utilizzato nella "client-side", anche per la scrittura di applicazioni "server-side";

- HTML: linguaggio di markup. nato per la formattazione e impaginazione di documenti ipertestuali.
- CSS: linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML ad esempio i siti web e relative pagine web;
- Console con ispezione network: monitoraggio da parte del browser dello scambio tra i pacchetti.

Da qui in avanti, quando parliamo del Client intendiamo l'insieme costituito dal Browser e Javascript (spesso delegato quindi alla parte di frontend), mentre quando parliamo del server intendiamo Node.js (framework per realizzare applicazioni Web in JavaScript, permettendoci di utilizzare questo linguaggio, tipicamente utilizzato nella "client-side", anche per la scrittura di applicazioni "server-side").

Approccio Asincrono/Event-driven/Reattivo:

La caratteristica principale di Javascript risiede nella possibilità di accedere alle risorse del sistema operativo in modalità event-driven e non sfruttando il classico modello basato su programmazione sequenziale, processi e thread concorrenti, utilizzato dai classici linguaggi di programmazione. Il modello event-driven, o "programmazione ad eventi", o "programmazione reattiva" si basa su un concetto piuttosto semplice: si esegue un'azione quando accade qualcosa. Ogni azione quindi risulta asincrona a differenza dello stile di programmazione tradizionale in cui una azione succede ad un'altra solo dopo che la prima è stata completata. Le applicazioni vengono concepite come un insieme di funzioni da eseguire al manifestarsi di certi eventi. Tali funzioni sono chiamate indifferentemente "callback", "listener" o "event handler". L'efficienza deriva dal considerare che le azioni tipicamente effettuate riguardano il networking, ambito nel quale capita spesso di lanciare richieste e di rimanere in attesa di risposte che arrivano con tempi che, paragonati ai tempi del sistema operativo, sembrano ere geologiche. Grazie al comportamento asincrono, durante le attese di una certa azione il runtime engine può gestire qualcos'altro che ha a che fare con la logica applicativa, ad esempio.