Università di Verona, Dipartimento di Informatica

Marco De Bona, Davide Quaglia

INTRODUZIONE AI WEB SOCKET

1 Introduzione

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.

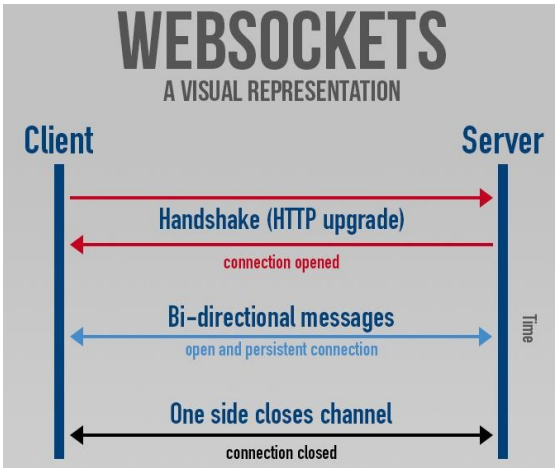

Figura 1. Protocollo WebSocket

Comunicazione bidirezionale simmetrica. *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.

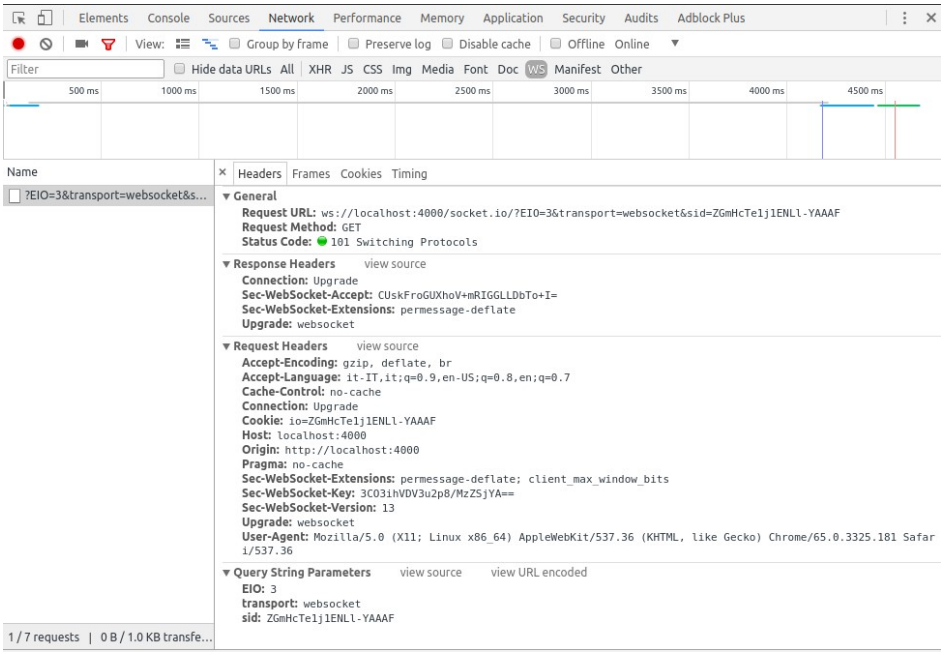

Figura 3. Ispezione dell'Upgrade request avvenuto tra browser e server visualizzata attraverso la console di sviluppo del browser.

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 4 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.

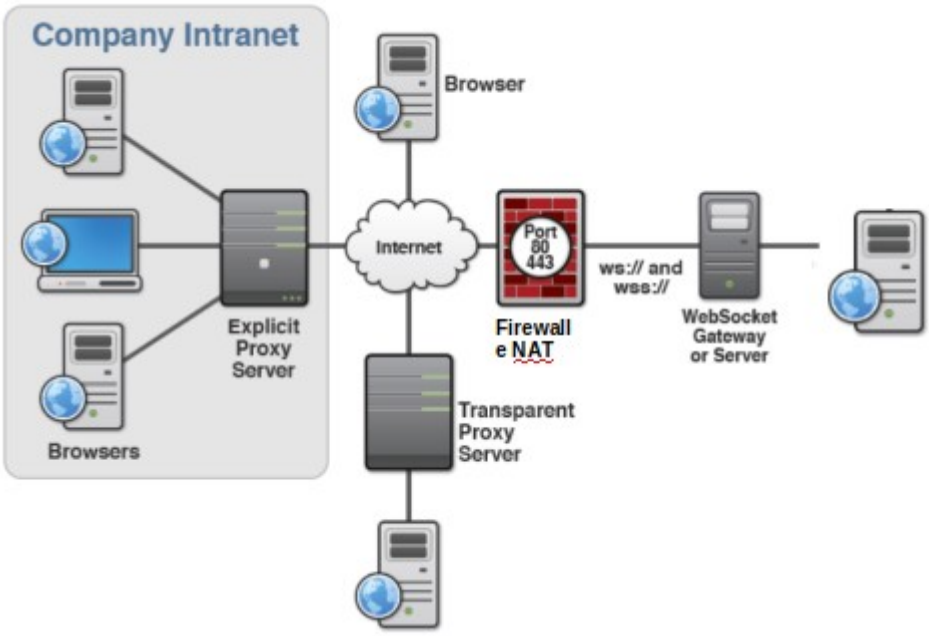

Figura 4 Architettura del Websocket con proxy server espliciti e trasparenti

2 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.

3 Esempio pratico: Websocket-Chat

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. http://www.html.it/guide/guida-javascript-di-base/
- **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”. http://www.html.it/guide/guida-nodejs/

- **HTML**: linguaggio di markup. Nato per la formattazione e impaginazione di documenti ipertestuali. http://www.html.it/guide/guida-html/
- **CSS**: linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML ad esempio i siti web e relative pagine web. http://www.html.it/guide/guida-css-di-base/
- **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, mentre quando parliamo del server intendiamo Node.js.

3.1 Introduzione a NODE.JS

Node.js è un 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”.
La piattaforma è basata sul JavaScript Engine V8, che è il runtime di Google utilizzato anche da Chrome e disponibile sulle principali piattaforme.

3.2 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.

3.3 Sviluppo

Verrà sviluppata una chat multi-utente a cui collegarsi tramite il browser. In particolare sono stati sviluppati i seguenti punti:

- Registrazione del nome di contatto che si vuole avere quando si accede alla chat.
- Invio dei messaggi in broadcast a tutti gli utenti attualmente collegati alla chat.
- Visualizzazione dei messaggi inviati col nome della persona che lo ha inviato.

4 Installazione del software ed esecuzione della chat

Per questa esercitazione è necessario un sistema operativo Linux. Se non si dispone di esso in maniera nativa, si può utilizzare la VM creata in una delle esercitazioni precedenti. E’ necessario aprire una shell ed eseguire i seguenti passi:

Preparazione della Chat:

- Scompattare dall’archivio dell’esercitazione la cartella WebSocket_Chat/ nella propria cartella di lavoro e aprire una shell mettendosi nella cartella WebSocket_Chat/

Installazione di NodeJS:

- sudo apt-get update
- curl -sL https://deb.nodesource.com/setup_NN.x | sudo -E bash -
  (dove NN è la versione più recente per la propria distribuzione Linux
  si veda https://deb.nodesource.com/)
- sudo apt-get install -y nodejs
- npm config set strict-ssl false
- sudo apt-get install build-essential
- sudo npm install --global nodemon
- sudo npm install express
- sudo npm install socket.io
- sudo npm install base64id

Durante l'installazione potreste notare dei Warning di NodeJS. Dovrebbe funzionare tutto ugualmente ma in caso voleste sistemare occore dare i comandi

- sudo npm audit fix
- sudo npm audit fix --force

Avvio della Chat:

- Il file server.js è contenuto nella cartella WebSocket_Chat/ mentre i file index.html e chat.js sono contenuti nella cartella WebSocket_Chat/public/
- Entrare nella cartella WebSocket_Chat/ contenente il file server.js e lanciare il comando

  $ nodemon server.js

- aprire browser alla pagina http://localhost:4000

NOTA: la chiamata

$ nodemon server.js

può essere anche sostituita con

$ nodejs server.js

5 Descrizione del codice

Di seguito la spiegazione del codice al fine di poter fare gli esercizi.

5.1 Introduzione a EXPRESS

Express è una libreria di Node.js che permettere di costruire molto facilmente applicazioni web. La variabile `var express=require('express')` crea una variabile che per funzionare necessita che la libreria Express sia installata. Successivamente questa variabile viene utilizzata per creare il server web in ascolto sulla porta 4000.

La documentazione di EXPRESS è disponibile qui
https://expressjs.com/

5.2 Creazione WEBSOCKET

Socket.IO è una libreria JavaScript utilizzata per implementare il protocollo WebSocket e racchiude numerose funzioni, incluse il broadcasting a tutti i socket collegati, il salvataggio dei dati riguardanti ciascun utente e l'approccio asincrono di I/O.
Come per la creazione della variabile Express, `var socket=require("socket.io")` crea una variabile che per funzionare ha bisogno che la libreria socket.io sia installata.

La documentazione di SOCKET.IO è disponibile qui
https://socket.io/

```
//NOME FILE: server.js

var express = require('express');
var socket = require('socket.io');

//Chat setup
var app = express();

// in questo momento il server è in attesa delle connessioni HTTP sulla porta 4000
var server = app.listen(4000, function(){
   console.log('waiting for HTTP requests on port 4000,');
});

// Static files
/*con questa funzione viene specificato a Nodejs che una volta ricevuta una
connessione deve andare a cercare nella cartella public il file html da fornire
al client
*/
app.use(express.static('public'));

// Socket setup & pass server
/*una volta che la connessione è stata ricevuta qui viene effettuato l'upgrade ad
una connessione websocket e il server si mette in attesa degli eventi ai quali
rispondere
```

```
*/
var io = socket(server);

io.on('connection', function(webSocket){

   console.log('made webSocket connection', webSocket.id);

   // Ricezione di un messaggio da inoltrare ai client
   webSocket.on('message', function(data)
   {
       io.sockets.emit('UploadChat', data);
   });
});
```

5.3 Funzionamento del Server

Alla riga 10 viene creato il socket HTTP. Dopo la creazione del socket, il server si mette in attesa delle connessioni HTTP da parte dei client. Quando viene aperta la connessione dal browser viene effettuato l'upgrade a WebSocket (riga 28). Successivamente il server si mette in ascolto del successivo evento che richiederà il suo intervento, ovvero quello di ricevere un messaggio con il tag "message", che farà sì che il server inoltri il messaggio a tutti i client connessi mediante l'oggetto chiamato "sockets" che raggruppa tutti i socket creati.

5.4 Implementazione HTML

Il codice HTML crea l'interfaccia utente della chat. Esso contiene dei tag controllati da Javascript per estrarre le informazioni che devono essere inviate al server e visualizzare i messaggi proveniente dagli altri utenti.
Il tag <script> contiene invece il link del file .js da eseguire all'apertura del file HTML; esso rappresenta il codice di implementazione della chat lato client (vedere sezione successiva).

```
//NOME FILE: index.html

<!DOCTYPE html>
<html>
<head><meta charset="utf-8">
<title>WebSocket Chat</title>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.1.0/socket.io.dev.js">
</script>
<link href="/styles.css" rel="stylesheet" />
</head>

<body>
       <div id="mario-chat">
           <h2>Chat</h2>
           <div id="chat-window">
               <div id="output"></div>
               <div id="feedback"></div>
           </div>
```

```
            <h4 id="sender" style="padding-left: 20px">Handle</h4>
            <input id="message" type="text" placeholder="Message" />
            <button id="send">Send</button>
        </div>
</body>
<script src="/chat.js"></script>
</html>
```

5.5 Funzionamento della chat

Una volta che la chat è stata aperta sul browser, è richiesto il nome utente. Si noti poi l'uso che Javascript fa degli identificatori dei tag HTML5. Alla riga 15 viene usata la funzione inner.HTML per modificare il contenuto della pagina con il nome utente con la quale si è effettuata la registrazione. A questo punto alla riga 19 viene creato il socket che deve connettersi con il server. Alla riga 22 viene aggiunto un listener all'evento "click" del bottone che, quando verrà premuto, invierà al server il proprio nome utente e il contenuto del testo del messaggio, così da poterlo inoltrare agli altri utenti. Importante è notare che viene passata la stringa "message" che il server usa per riconoscere e gestire questo evento. Alla riga 32 viene gestito l'evento di I/O 'UploadChat' che arriva dal server che rilancia un dato scritto da qualche altro client usando le istruzioni alle righe 25-26; infatti si nota che l'oggetto "data" contiene i campi "sender" e "message" che sono stati valorizzati da qualche altro client usando le istruzioni alle righe 25-26.

```
//NOME FILE: chat.js

var name= prompt("What's your name?");
while(name==""){
    name=prompt("You have to choose a name. \n What's your name?")
}

// Query DOM
var message = document.getElementById('message'),
   sender = document.getElementById('sender'),
   btn = document.getElementById('send'),
   output = document.getElementById('output'),
   feedback = document.getElementById('feedback');

sender.innerHTML=name;
sender.value=name;

// Invio richiesta di connessione al server
var webSocket = io.connect();

// Listen for events
btn.addEventListener('click', function(){
   if (message.value!="" ){
           webSocket.emit('message', {
            sender: sender.value,
           message: message.value,
           });
           message.value = "";
   }
});

webSocket.on('UploadChat', function(data){
   feedback.innerHTML = '';
   output.innerHTML += '<p><strong>' + data.sender + ': </strong>' + data.message + '</p>';
});
```

Esercizio 0

Effettuare una cattura Wireshark relativa al funzionamento della chat. Occorre attivare la cattura prima di aprire la URL dalla finestra del browser e ascoltare non solo sull'interfaccia di loopback ma anche su quella verso l'esterno (opzione ANY).

Esercizio 1

Modificare a piacimento il contenuto del file public/index.html e valutarne l'impatto grafico.

Esercizio 2

Modificare il sorgente del codice in modo da far ascoltare il server sulla porta 80 invece che 4000. Quali altri accorgimenti sono necessari per farlo funzionare: lato server? Lato client?

Esercizio 3

Modificare il sorgente del codice in modo da cambiare nome ai seguenti eventi

- message → messaggio
- UploadChat → aggiornamento

Esercizio 4

Se si dispone di due PC in grado di dialogare sulla stessa rete IP (oppure un PC per il server e uno smartphone per il client sempre in grado di dialogare tra loro a livello IP) provare ad accedere al server che è su Linux mediante diversi tipi di browser e di sistemi operativi. Cosa si può notare?

Esercizio 5

Modificare il sorgente del codice per fare in modo che ad ogni utente connesso alla chat arrivi nella console il messaggio "l'utente sta scrivendo..." .

NOTA: Lato client, bisogna spedire al server un evento apposito (ad es. "typing") quando l'utente scrive sulla tastiera (catturando l'evento di sistema "keydown"). Lato server, la chiamata webSocket.**broadcast**.emit('typing', data) rilancia l'evento "typing" a tutti i client connessi tranne che a quello dalla quale si è ricevuto il messaggio. Lato client occorre infine gestire la ricezione del messaggio "typing" che arriva dal server (vedere gestione del messaggio "UploadChat").

Esercizio 6

Pensando all'esercizio sulla chat dell'esercitazione sulla programmazione mediante socket, quali differenze/vantaggi/svantaggi si hanno con le tecnologie impiegate in questa esercitazione?