Università di Verona, Dipartimento di Informatica
Programmazione e Sicurezza delle Reti
Marco De Bona, Davide Quaglia

INTRODUZIONE AI WEB SOCKET

1 Introduzione

Il **WebSocket** è un protocollo di comunicazione web che fornisce un canale di **comunicazione bidirezionale** attraverso una singola connessione TCP inizialmente utilizzata per il protocollo HTTP. 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 WebSockets oltre a fornire al server la possibilità di rispondere alle richieste del browser, possono anche effettuare dei "*push*" al browser per aggiornarlo cosa che non può avvenire con il tradizionale protocollo HTTP.

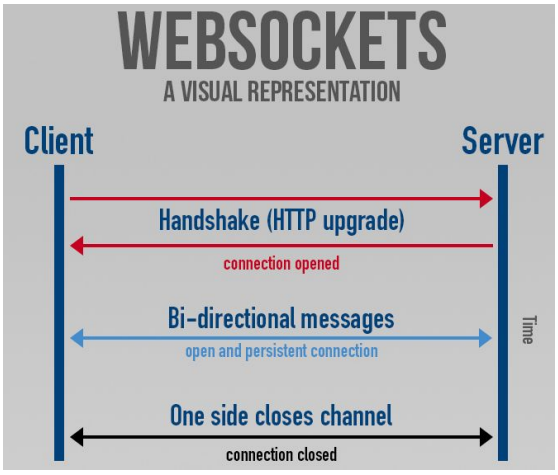

Figura 1. Protocollo WebSocket

> ***Comunicazione bidirezionale*** *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.*

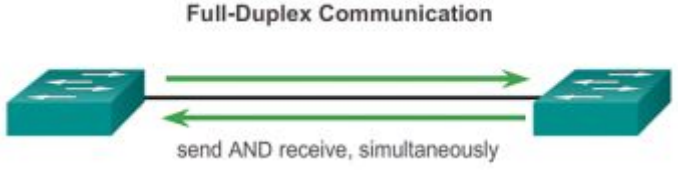

Figura 2. Comunicazione bidirezionale

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.

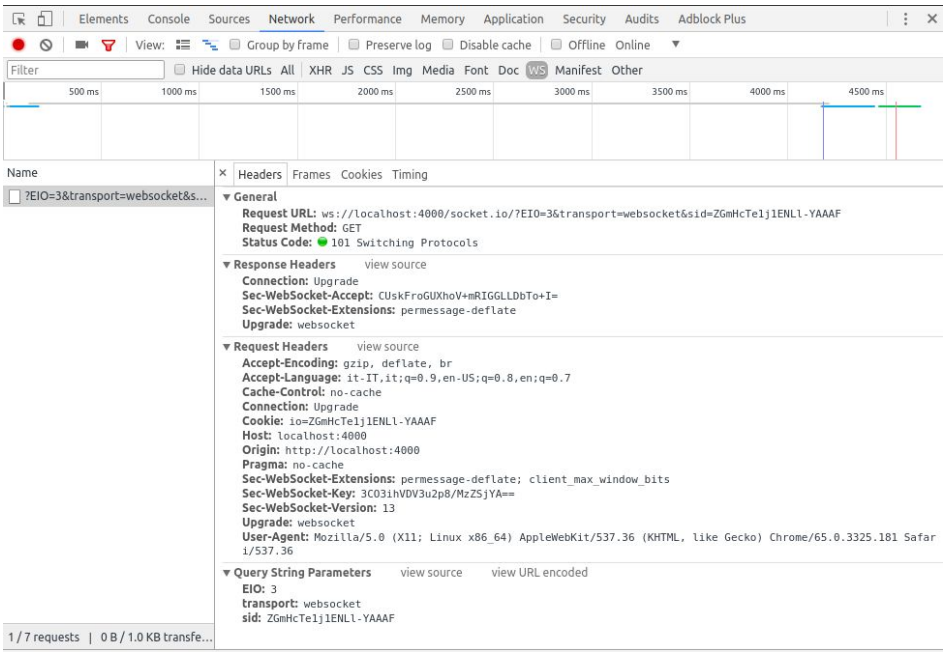

Figura 3. Ispezione dell'Upgrade request avvenuto tra browser e server visualizzata attraverso la console di sviluppo del browser

Il protocollo WebSocket è supportato attualmente da numerosi browser, inclusi Google Chrome, Internet Explorer, Firefox, Safari e Opera.

Diamo un'occhiata a come i WebSocket interagiscono con i server proxy. 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 espliciti o noti, che possono fornire memorizzazione e sicurezza del contenuto. Le richieste del client possono essere instradate attraverso server proxy trasparenti o sconosciuti.

A differenza del normale traffico HTTP, che utilizza un protocollo di richiesta/risposta, le connessioni WebSocket come abbiamo già visto possono rimanere aperte per un lungo periodo. I server proxy possono sia consentire la connessione prolungata sia interromperla.

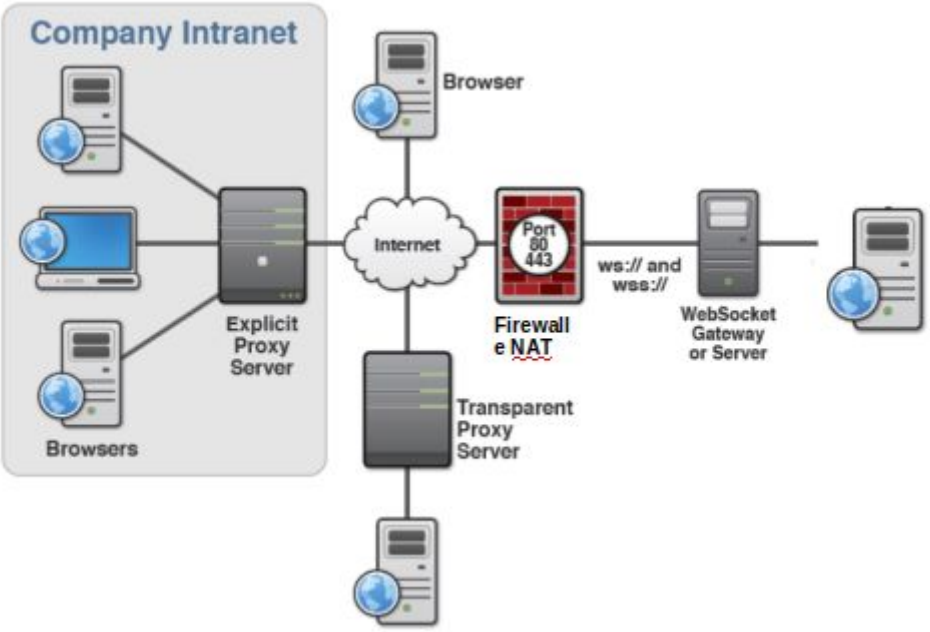

Figura 4 Architettura del Websocket con proxy server espliciti e trasparenti

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 del WebSockets, chiudendo le connessioni una volta completate piuttosto che mantenendo lo stato della connessione.

Inoltre, i WebSockets possono essere utilizzati solo se gli utenti utilizzano i moderni browser con JavaScript abilitato.

Dovrebbero poi essere considerati possibili impatti sull'architettura di rete. WebSockets, 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 Websocket-Chat

Per la realizzazione di questa Chat verranno usati vari strumenti che non sono stati approfonditi nel corso, ma ci servono per analizzare il funzionamento dei websocket. Per gli studenti interessati vengono allegati dei link di approfondimento:

- **JavaScript**: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, di siti web e 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 quindi 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, anche se maggiormente performante su sistemi operativi UNIX-like.

3.2 Approccio Asincrono

La caratteristica principale di Node.js risiede nella possibilità di accedere alle risorse del sistema operativo in modalità event-driven e non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server.
Il modello event-driven, o "programmazione ad eventi", si basa su un concetto piuttosto semplice: si lancia una 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 essa è stata completata. Ciò dovrebbe garantire una certa efficienza delle applicazioni grazie ad un sistema di callback gestito a basso livello dal runtime. 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 può gestire qualcos'altro che ha a che fare con la logica applicativa, ad esempio.

3.3 Sviluppo

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

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

4 Codice del programma

Per la guida all'esecuzione dell'applicazione si rimanda all'Appendice A.

4.1 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.

4.2 Creazione WEBSOCKET

Socket.IO è una libreria JavaScript utilizzata per implementare il protocollo WebSocket e racchiude numerose funzione, 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.

```
//Server

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
```

```
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);
   });
});
```

4.2 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. Successivamente il server esegue la funzione on "connection" e si mette in ascolto del successivo evento che richiederà il suo intervento, ovvero quello di ricevere un messaggio con il tag "message", che farà si che il server inoltri il messaggio a tutti i client connessi mediante l'oggetto "d'insieme" chiamato "sockets".

4.3 Implementazione HTML

Il codice HTML, che crea l'interfaccia utente nella quale si potrà visualizzare la chat, viene utilizzato da Javascript per estrarre il contenuto di alcuni tag, in cui vengono inserite le informazioni che devono essere inviate al server e in cui vengono visualizzati i messaggi proveniente dagli altri utenti.
Lo <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).
Oltre alla struttura generale di HTML5, si veda in particolare l'uso che Javascript fa degli identificatori dei tags.

```
//Codice HTML

<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset="utf-8">
        <title>WebSockets Chat</title>
                                                                    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.1.0/socket.io.dev.j
s"></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>
```

4.4 Funzionamento della chat

Una volta che la chat è stata aperta sul browser e si ha effettuato la registrazione col nome utente, alla riga 15 viene usata la funzione inner.HTML per modificare il contenuto della pagina con il nome con la quale si è effettuata la registrazione.
A questo punto alla rigo 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" in modo che il server sappia come gestire questo evento.

```
//Client

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', {
        message: message.value,
         sender: sender.value,
         });
         message.value = "";
   }
});

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

Esercizio 1

Lanciare l'applicazione dopo aver fatto partire l'ispezione del Network tramite la console di sviluppo del browser. Ogni quanto tempo il client fa sapere al server che è ancora connesso? E' un'azione dovuta all'implementazione della chat o insita nel WebSocket? A cosa serve tale procedura?

Esercizio 2

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 "keypress"). 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 3

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

Appendice A: creazione della websocket-chat

Nel caso si stia usando il proprio computer per installare e avviare la Webchat sono necessari i seguenti passi:

Parte 1:

- sudo apt-get update
- curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
- sudo apt-get install nodejs
- sudo apt-get install npm
- npm config set strict-ssl false <<<<< vale solo per VM in Delta!!!
- sudo apt-get install build-essential
- npm install --global nodemon (usare sudo in caso di problemi coi permessi)

Parte 2 :

- A questo punto una volta entrati nella directory contenente il file server.js lanciare il comando

  $ nodemon server.js

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

NOTA: Se l'ultimo comando della Parte 1 dà problemi allora nella Parte 2 sostituire

$ nodemon server.js

con

$ nodejs server.js

NOTA 2: se, lanciando il server, vengono richiesti moduli aggiuntivi usare i comandi

- npm install express
- npm install socket.io