



# Tecnologie e applicazioni web

## Websocket

---

Filippo Bergamasco ( [filippo.bergamasco@unive.it](mailto:filippo.bergamasco@unive.it))

<http://www.dais.unive.it/~bergamasco/>

DAIS - Università Ca' Foscari di Venezia

Anno accademico: 2017/2018

# WebSocket

Protocollo di comunicazione che permette una semplice comunicazione **full-duplex** al di sopra di una connessione TCP/IP

Studiato per essere “compatibile” con il protocollo HTTP

- Utilizza le stesse porte
- Utilizza il protocollo HTTP per l'handshake iniziale
- Supporta proxy e altri intermediari

# Comunicazione full-duplex

A differenza del protocollo HTTP, una volta instaurata una connessione WebSocket lo scambio di messaggi può avvenire indifferentemente e contemporaneamente tra client e server

Permette di superare le limitazioni di HTTP che forzano un meccanismo di tipo request-response

# Comunicazione full-duplex

Essendo un protocollo completamente funzionale di per sé, può essere utilizzato anche al di fuori del browser.

Solitamente, la sua applicazione principale è quella di permettere il trasporto di messaggi in modo **bidirezionale** in applicazioni web-based (all'interno quindi del browser)

# WebSocket

Composto da due componenti di alto livello:

1. Un sistema di handshake basato su HTTP per negoziare i parametri di connessione e instaurare quindi il canale di comunicazione
2. Un meccanismo di framing per gestire lo scambio di dati binari e/o testuali con:
  - a. Poco overhead
  - b. Bassa latenza

# WebSocket handshake

Il canale di comunicazione websocket viene stabilito a partire da una connessione HTTP esistente.

- Si sfrutta l'header **Upgrade** di HTTP per negoziare il cambio di protocollo
- Il meccanismo consente l'attraversamento di proxies che supportano il protocollo websocket
- Contiene accorgimenti per impedire attacchi malevoli

# Handshake: client->server

```
GET /socket HTTP/1.1  
Host: thirdparty.com  
Origin: http://example.com  
Connection: Upgrade  
Upgrade: websocket  
Sec-WebSocket-Version: 13  
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==  
Sec-WebSocket-Protocol: appProtocol,appProtocol-v2  
Sec-WebSocket-Extensions:  
x-webkit-deflate-message, x-custom-extension
```



L'handshake inizia con una request di tipo GET ad una certa risorsa del server

# Handshake: client->server

```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: http://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: appProtocol,appProtocol-v2
Sec-WebSocket-Extensions:
x-webkit-deflate-message, x-custom-extension
```



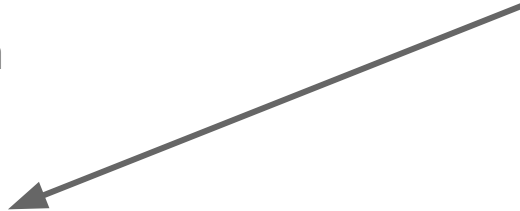
Il client richiede di modificare il protocollo utilizzato da HTTP a websocket



# Handshake: client->server

```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: http://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: appProtocol,appProtocol-v2
Sec-WebSocket-Extensions:
x-webkit-deflate-message, x-custom-extension
```

Versione del protocollo  
utilizzata dal client



# Handshake: client->server

```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: http://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: appProtocol,appProtocol-v2
Sec-WebSocket-Extensions:
x-webkit-deflate-message, x-custom-extension
```


Chiave composta da una stringa di bytes random codificati in base-64 utilizzata per:

- Verificare il supporto del server al protocollo
- impedire ai proxy di effettuare caching e inviare handshake duplicati

# Handshake: client->server

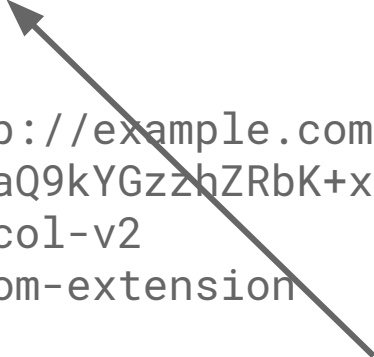
```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: http://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: appProtocol,appProtocol-v2
Sec-WebSocket-Extensions:
x-webkit-deflate-message, x-custom-extension
```

Lista di sotto-protocolli ed  
eventuali estensioni  
utilizzate dall'applicazione



# Handshake: client←-server

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Access-Control-Allow-Origin: http://example.com
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
Sec-WebSocket-Protocol: appProtocol-v2
Sec-WebSocket-Extensions: x-custom-extension
```



Codice di risposta 101  
conferma il cambio di  
protocollo

# Handshake: client←-server

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

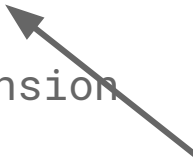
Connection: Upgrade

Access-Control-Allow-Origin: http://example.com

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=

Sec-WebSocket-Protocol: appProtocol-v2

Sec-WebSocket-Extensions: x-custom-extension



Hash della chiave inviata  
nella precedente request  
del client (a cui è stata  
concatenata una stringa  
fissa che dipende dal  
protocollo)

# Handshake: client←-server

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

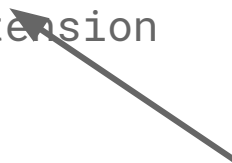
Access-Control-Allow-Origin: http://example.com

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=

Sec-WebSocket-Protocol: appProtocol-v2

Sec-WebSocket-Extensions: x-custom-extension

Sotto-protocollo da  
utilizzare ed estensioni  
supportate dal server



# Handshake

Dopo lo scambio di messaggi HTTP request-response in cui client e server negoziano il tipo di sotto-protocollo da utilizzare:

- La connessione TCP (o SSL/TLS) viene lasciata aperta
- I messaggi scambiati nel canale sono quelli definiti dal protocollo websocket (non più HTTP)

# Messaggi websocket

- Il protocollo prevede lo scambio di **messaggi** testuali (UTF-8) o binari di lunghezza arbitraria
- L'invio di messaggi può avvenire in entrambe le direzioni. L'altro soggetto viene notificato ogni qual volta un messaggio viene ricevuto
- Messaggi sono divisi in uno o più frames, vengono inviati in sequenza e riassemblati dall'altra parte



# Frames

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						

Frame overhead variabile da 2 a 10 bytes. Tutti i messaggi inviati dal client contengono anche una masking key da 0-4 bytes comportando un overhead variabile da 6 a 14 bytes.

# Frames


Bit	+0..7		+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...
32	...					
64	...				Masking key (0–4 bytes) ...	
96	...				Payload ...	
...	...					

0: vi sono altri frames da ricevere per il messaggio corrente

1: Il messaggio è completo con questo frame

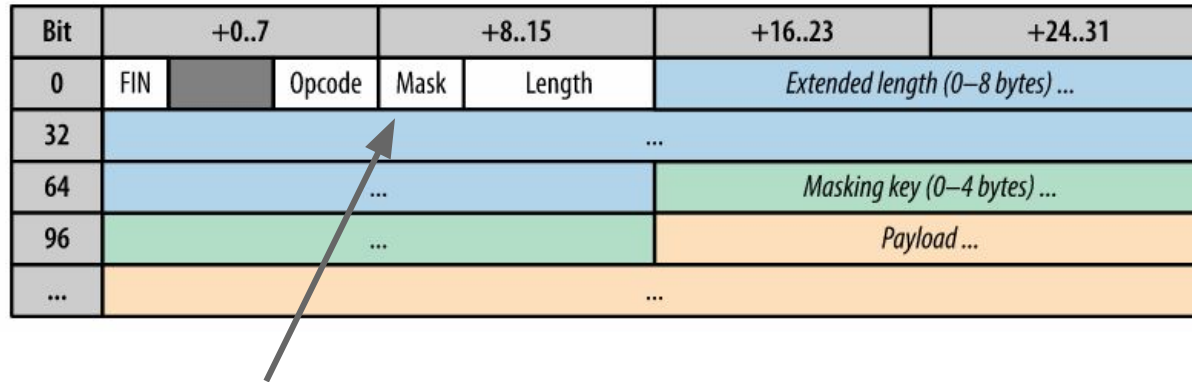
# Frames

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						



Tipo di messaggio: testuale (1), binario (2), close (8), ping (9), pong (10)

# Frames



0: il frame non è masked

1: il frame è masked

# Frames

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						

Lunghezza del messaggio (composto da uno o più bytes)

# Frames

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						

Il payload di tutte le comunicazioni iniziate dal client sono mascherati con questa chiave per evitare attacchi di tipo “cache poisoning”

# Frames

Bit	+0..7			+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	Extended length (0–8 bytes) ...	
32	...						
64	...					Masking key (0–4 bytes) ...	
96	...					Payload ...	
...	...						

Contenuto del messaggio



# Framing

Il protocollo websocket prevede la suddivisione dei messaggi in frames per due motivi:

1. E' possibile iniziare a trasferire il contenuto dei messaggi senza conoscere a priori la dimensione dei dati
2. I frames di messaggi diversi possono essere opzionalmente interposti per evitare che un messaggio causi ritardi a messaggi più brevi



# WebSocket in Javascript

WebSocket è supportato dai più comuni browser oggi esistenti, sia desktop che mobile

In ambiente Javascript, può essere utilizzato attraverso una semplice libreria chiamata socket.io

<https://socket.io/>



socket.io

Permette l'invio in modo asincrono di eventi da client a server, viceversa