A.A 2021 - 2022

M. CECCARELLI - E. D'ANGELIS - P. ZARRI

LIBRARY SEAT RESERVATION

WEBSOCKET VS RSOCKET

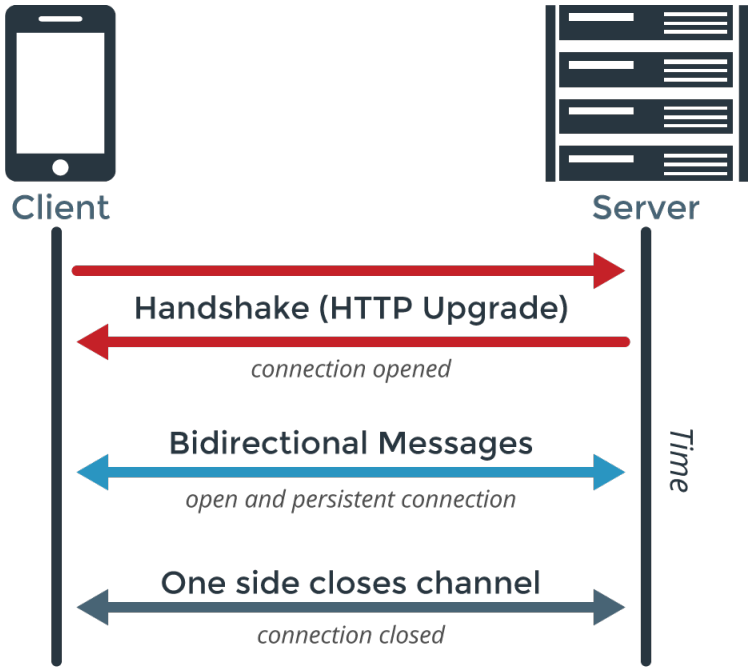
LIBRARY SEAT RESERVATION

INTRODUZIONE

Sia **WebSocket** che **RSocket** sono protocolli di comunicazione bidirezionale, multiplex e duplex, in grado di inviare dati da un server a un client riutilizzando lo stesso canale di connessione. Il vantaggio di questi protocolli, rispetto ad HTTP classico, consiste nella possibilità da parte del server di inviare contenuti ad un client (es. browser) senza prima dover essere sollecitato dal client stesso (o viceversa).
I due protocolli funzionano però a livelli diversi, e le differenze vengono elencate di seguito.

WEBSOCKET

WebSocket è un protocollo di comunicazione di basso livello basato su Transmission Control Protocol (**TCP**). In pratica definisce come un flusso di byte viene trasformato in frame, ma senza offrire nessuna *semantica* a livello di applicazione: spetta allo sviluppatore creare un protocollo applicativo per interagire con il Websocket.



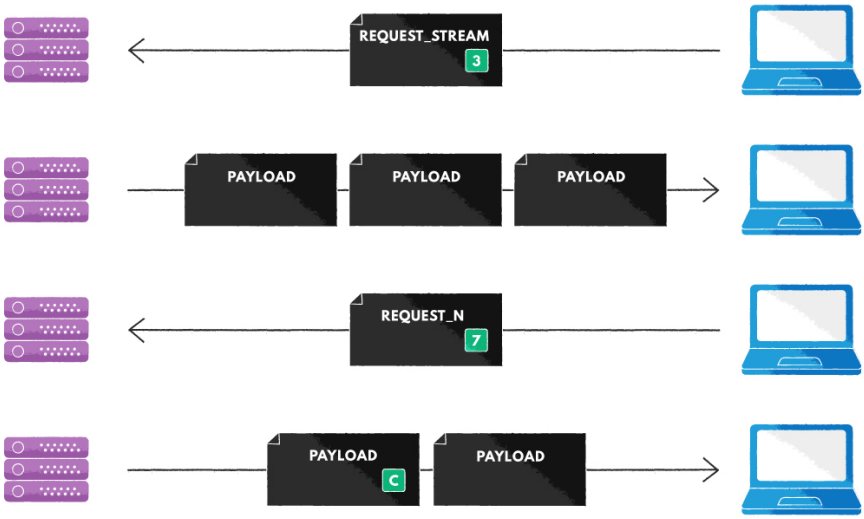
Per stabilire una connessione Websocket, il client invia una richiesta di handshake ed il server invia una risposta di avvenuta connessione.

Una volta stabilita la connessione, il client ed il server possono inviare dati tramite il Websocket in entrambe le direzioni.

RSOCKET

RSocket è un protocollo a livello applicativo, totalmente asincrono, che implementa la specifica Reactive Streams, con supporto a caratteristiche avanzate come *framing*, ripresa automatica della sessione (*session resumption*) e *backpressure* a livello di applicazione. Lo scopo di questo protocollo è quello di ridurre la latenza nelle comunicazioni e i costi dell'infrastruttura.

RSocket è inoltre **agnostico rispetto al livello di trasporto utilizzato**: può infatti essere eseguito su diversi trasporti di flusso di byte come TCP, WebSocket, HTTP/2 o Aeron, e permette quindi di cambiare il livello di trasporto sottostante in base, ad esempio, alla capacità del dispositivo ricevente o alle esigenze di prestazione — a differenza di WebSocket che invece è vincolato all'utilizzo del livello di trasporto TCP.



La gestione dei flussi di dati, in particolare i dati "live" il cui volume non è predeterminato, richiede un'attenzione particolare in un sistema asincrono. Il problema più rilevante riguarda il consumo di risorse, che deve essere controllato in modo tale da evitare situazioni in cui se un server invia degli eventi più velocemente di quanto il client sia in grado di elaborarli, alla fine il client esaurirà le risorse e sarà costretto a chiudere la connessione.
RSocket ha un supporto integrato per il controllo del flusso, che aiuta a evitare queste situazioni riducendo il carico, chiamato **backpressure** non bloccante: Il destinatario può specificare la quantità di dati che desidera consumare, entro un intervallo di tempo predefinito, e non ne riceverà di più fino a che non notifica al mittente che è pronto ad elaborarne altri.

RSocket supporta le seguenti interazioni:

- **Fire-and-forget**: un'ottimizzazione del classico metodo *request/response* HTTP, da utilizzare quando non è necessario attendere e associare una risposta a ogni relativa richiesta;
- **Request-Response**: simili ad HTTP, sono flussi di una sola risposta ottimizzati in modo che Il consumatore attenda un messaggio di risposta dal server, ma senza mai bloccarsi in modo sincrono;
- **Request-Stream**: il client invia un singolo frame al server e riceve l'intero stream di dati, in maniera asincrona (*push*);
- **Channel**: canale bidirezionale, con un flusso di messaggi in entrambe le direzioni.