HTTP/2 Protocol

Salvi Niccolò

18 gennaio 2021

Indice

1 HTTP Protocol

1.1 Introduzione

1.1.1 Definizione di Protocollo

Un **protocollo** è un insieme di regole che permette di trovare uno standard di comunicazione tra i diversi computers attraverso la rete[1].
I calcolatori, per potersi scambiare informazioni, devono predisporre di protocolli che permettano loro di attribuire ad un determinato comando un significato univoco per tutti.

I punti principali, descritti da un protocollo, sono i seguenti:

- Il *formato* che il *messaggio* scambiato deve avere
- Il *modo* in cui i computers devono scambiarsi i messaggi

1.1.2 World Wide Web

Il **WWW** si interfaccia con un vasto numero di server, che colloquiano tra di loro attraverso protocolli standard *TCP/IP* ed altri, che si servono di altrettanti protocolli di livello superiore.
Tutti i clients ed i servers web devono essere in grado di gestire il protocollo HTTP, affinché possano scambiarsi i documenti di ipertesto.

1.1.3 Uniform Resource Locator

Ogni risorsa disponibile sulla rete deve essere identificabile univocamente, tramite indirizzi **URL**.

protocollo://server:socket/pathname

Protocollo: protocollo utilizzato

Server: indirizzo IP (numerico o logico) del calcolatore che detiene la risorsa

Porta: porta a cui il protocollo fa riferimento

Pathname: percorso completo del file cercato

[1] Per rete s'intende l'insieme di due o più calcolatori connessi tra di loro ed in grado di condividersi informazioni.

1.2 Definizione

Si tratta di un protocollo *stateless*[2] che permette la ricerca, il recupero dell'informazione in maniera veloce e di seguire eventuali links.
Per accedere alle informazioni fornite dal server HTTP, è necessario utilizzare un *browser*, che sia in grado di interpretare le informazioni inviate dal server.

1.2.1 Caratteristiche

Si tratta di un protocollo:

- *Request-Reply*: ad ogni richiesta del client corrisponde una risposta da parte del server
- *ASCII*: i messaggi[3] scambiati tra client e server sono sequenze di caratteri ASCII

1.3 Request

Una volta instaurata una connessione tra il browser ed il server, il client, tramite il protocollo HTTP, effettua la richiesta della risorsa al server.

Request Line + Header Lines + CRLF (Carriage Return, Line Feed) + Entity Body

1.3.1 Request Line

La **Request Line** contiene i seguenti elementi:

- *Method*: metodo che si chiede al server di eseguire
  - CONNECT
  - DELETE
  - GET
  - HEAD
  - OPTIONS
  - POST
  - PUT
  - TRACE
- *SP*: white space

[2] La scelta di un protocollo che non conservi memoria è stata presa affinché fosse possibile passare velocemente da un server all'altro, tramite links ipertestuali.

[3] Per messaggio s'intende la richiesta del client o la risposta. Il corpo del messaggio è costituito dai dati effettivi da trasmettere.

- *URL*: risorsa su cui applicare la richiesta:
  - *: la richiesta non deve essere applicata ad una particolare risorsa, ma al server stesso
  - *AbsoluteUrl*: la richiesta è stata eseguita ad un *proxy* e viene indicato il DNS ed il path della risorsa
  - *AbsolutePath*: utile per inoltrare la richiesta ad un server o ad un *gateway*
- *Version*: versione del protocollo HTTP utilizzato

1.3.2 Header Lines

Le informazioni contenute negli headers della richiesta comunicano al server ricevente le informazioni addizionali, inerenti alla richiesta ed al client stesso.

Accept	Indicare *media-type* accettati come risposta, i tipi di *MIME* che il client può gestire[4]. Se il server non può mandare una risposta adeguata manda uno *Status-Code 406*
Accept-Charset	Stesso concetto del campo *accept*, inerente ai caratteri
Accept-Encoding	Stesso concetto del campo *accept*, riferito alle *content-codings* ed indica i processi di codifica che il client può riconoscere nella risposta del server
Accept-Language	Stesso concetto del campo *accept*, riferito ai *linguaggi naturali* dedicati all'uso umano. Indica in quale lingua si deve comunicare all'utente del client
Authorization	Mostra permessi del client
Cache-control	Mostra informazioni sulla cache
Expect	Quando il client richiede particolari operazioni al server[5]
From	Contiene l'indirizzo email dell'utente per usi umani[6]
Host	Specifica *Internet Host* ed il numero della porta attraverso la quale comunicare col server[7]

[4] Se la richiesta è priva di questo campo, significa che il client accetta almeno i tipi MIME *text/plain* e *text/html*.

[5] Se quest'ultimo non le può supportare, deve restituire l'appropriato *Status-Code*.

[6] Tale campo non viene fornito senza l'approvazione dell'utente per fini di privacy

[7] Senza la specificazione della porta, viene utilizzata quella di default, porta 80.

If-Match	Contiene alcune etichette da applicare all'entità da ottenere per poi permettere al client di effettuare dei confronti e riconoscere questa tra le altre entità che provengono dalla stessa risorsa
If-Modified-Since	Contiene una data ed indica al server di restituire una data risorsa solo se è stata modificata, altrimenti non c'è la necessità di una nuova trasmissione[8]
If-None-Match	Il client può verificare che nessuna delle entità ottenute dal server facciano parte di quelle specificate in questo campo
If-Range	Se un client ha una copia parziale di un'entità nella propria cache, può richiedere la porzione mancante dell'entità con l'utilizzo del metodo *GET*
If-Unmodified-Since	Contiene una data e se la risorsa specificata non è stata modificata dalla data presente nel campo, il server effettua la risposta normalmente come se questo campo non esistesse, altrimenti restituisce uno *Status-Code 412*
Max-Forwards	Contiene un numero decimale che indica il rimanente numero di volte che un messaggio può essere inoltrato
Proxy-Authorization	Contiene parametri che permettono l'autenticazione del client e l'autorizzazione ad operare da parte del proxy in questione
Range	Il client informa il server di quale range di bytes del corpo dell'entità necessità
Refer	Specifica URL della risorsa a cui si deve far riferimento e permette al server di generare dei link per la manipolazione efficiente dei dati
TE	Valori di *transfer coding* accettati nella risposta dal client[9]
User-agent	Contiene informazioni riguardanti nome e versione dell'applicativo che svolge la funzione di client

[8] Il server restituisce lo *Status-Code 304* senza trasmettere nessun corpo del messaggio

[9] Valori sono usati per indicare il tipo di codifica del corpo del messaggio e per rendere la comunicazione più sicura ed efficiente

1.4 Response

Una volta ricevuta la richiesta, il server risponde con un messaggio[10] di risposta.

Status-Line + Header lines + CRLF + Entity Body

1.4.1 Status Line

La **Status Line** è composta dai seguenti elementi:

- *Version*: versione del protocollo HTTP utilizzato
- *Status Text*: specifica testuale dello stato
- *Status Code*: codice a 3 cifre che ha la funzione di fornire al client le informazioni di stato, riguardanti l'esito della ricezione della richiesta.

  Il primo digit definisce 5 classi di risposta, mentre gli ultimi due non hanno nessuna categorizzazione:

  - 1xx: *Informazione* - Richiesta avvenuta e continuo processo

100	Continue
101	Switching Protocols

  - 2xx: *Successo* - Azione ricevuta, capita ed accettata

200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content

  - 3xx: *Ridirezione* - Servono altre informazioni per completare la richiesta

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

  - 4xx: *Client Error* - Errori nella richiesta

    –

[10] Il corpo contiene i dati richiesti dal client, ovvero i documenti ipertestuali

400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
416	Requested range not satisfiable
417	Expectation Failed

– 5xx: *Server Error* - Fallimento del server

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

1.4.2 Header Lines

Questi campi servono per passare al client ulteriori informazioni che non possono risiedere nello *Status-Code*.

Accept-Ranges	Ranges accettati e computati
Age	Intero positivo che indica il tempo, in secondi, trascorso da quando il server ha inviato la risposta
Etag	Valori di alcune etichette applicata all'entità restituita
Location	Indica al ricevente di ridirezionare la richiesta ad una locazione diversa da quella specifica dal URL
Proxy-Authenticate	Serie di parametri e schemi di autenticazione del proxy emerso dall URL[11].
Retry-After	Indica quanto tempo deve attendere il client prima di rieffettuare la richiesta, quando il servizio non è al momento disponibile
Server	Parametri che specificano il software utilizzato dal server
WWW.Authenticate	Parametri per l'autenticazione del URL

1.5 Definizione Metodi

1.5.1 CONNECT

Usato per instaurare una semplice connessione con un proxy.

1.5.2 DELETE

Richiede che il server ricevente elimini la risorsa specifica dall'URL, anche se non vi è nessuna garanzia che l'operazione vada a buon fine.

1.5.3 GET

Tende di recuperare le informazione localizzate da URL.

- GET per risorsa statica –> *GET /foo.html*
- GET per risorsa dinamica –> *GET /cgi-bin/query?q=foo*

Si tratta di un metodo *sicuro* ed *idempotente* e può essere:

- *Assoluto*: risorsa viene richiesta senza altre specificazioni
- *Condizionale*: risorsa corrisponde ad un criterio indicato negli headers *If-Modified-Since*, *If-Unmodified-Since*, *If-Match*, *If-None-Match* o *If-Range*
- *Parziale*: risorsa richiesta è una sottoparte di una risorsa memorizzata, con l'utilizzo dei campi *Range*

1.5.4 HEAD

Identico al metodo *GET*, eccetto il fatto che il server non deve ritornare il corpo del messaggio, ma gli header inerenti all'entità richiesta[12].
Si tratta di un metodo *sicuro* ed *idempotente*, usato per verificare:

- *Validità URL*: se la risorsa esiste ed è di lunghezza non nulla
- *Accessibilità URL*: se la risorsa è accessibile presso il server e non sono richieste procedure di autenticazione
- *Coerenza di cache URL*: se la risorsa non è stata modificata rispetto a quella in cache

1.5.5 OPTIONS

Rappresenta una richiesta di informazioni inerenti alle opzioni[13] di comunicazione disponibili sul canale.

1.5.6 POST

Metodo né sicuro, né idempotente e permette di trasmettere informazioni dal client al server.
Il server può rispondere in tre modi differenti:

200	OK
201	Created
204	No content

1.5.7 PUT

Serve per allocare nuove risorse sul server, passategli dal client.
Le nuove risorse sono memorizzate in relazione al URL: se una risorsa già esiste in corrispondenza all'URL specificato, la nuova risorsa verrà considerata come un aggiornamento della prima.

1.5.8 TRACE

Indica la richiesta di alcuni dati sul canale per testare e diagnosticare informazioni. Il ricevente può essere sia il server originale che il primo proxy o gateway sul canale.

[12] Usato per scopi di controllo e debugging.
[13] Riferite alla risorsa da utilizzare o alle capacità del server.

1.6 Connessioni

La transazione HTTP è composta da uno scambio di richiesta e risposta: il client apre una connessione TCP separata per ogni risorsa richiesta[14].

1. Client apre la connessione TCP con il server
2. Client invia il messaggio di richiesta HTTP sulla connessione
3. Server invia il messaggio di risposta HTTP sulla connessione
4. Connessione TCP viene chiusa

1.6.1 Problemi

I problemi riscontrati sono:

- *Overhead* per l'instaurazione ed abbattimento della connessione TCP
  - Per ogni trasferimento di risposta, 2 round trip time (RTT) in più
    es: per una risorsa Web che richiede 10 segmenti, sono trasmessi 17 segmenti, di cui 7 (3+4) di overhead.
  - *Overhead* sul SO per allocare risorse per ogni connessione
- *Overhead* per il meccanismo *slow start* del TCP.

Gli approcci proposti per risolvere i problemi:

- Connessioni TCP *persistenti* (HTTP/1.1)
- Connessioni multiple in parallelo: apertura simultanea di più connessioni TCP, il cui vantaggio è l'apparente riduzione del tempo di latenza percepito dall'utente.

  Problemi:

  - Aumento congestione sulla rete
  - Server serve un minor numero di client diversi contemporaneamente
  - Richieste di pagine interrotte dall'utente
  - Non garantita la riduzione della latenza: ogni richiesta è indipendente dalle altre
- Nuovi metodi per ottenere risorse multiple sulla stessa connessione

[14] La connessione TCP è una connessione non persistente e consente la concorrenza massima tra le varie richieste HTTP

1.6.2 Caching

Il caching lato server riduce i tempi di processamento della risposta e del carico sui server, mentre quello lato client riduce il carico di rete ed in parte il carico sui server.
Nel protocollo HTTP/1.0 vi sono tre differenti headers che riguardano tale tecnica:

- *Expires*: server specifica la scadenza della risorsa
- *If-Modified-Since*: client o proxy richiede la risorsa solo se modificata dopo la data indicata nella richiesta
- *Pragma*: no-cache: client indica ai proxy di accettare solo la risposta del server

1.7 HTTPS

La differenza tra HTTP ed HTTPS deriva dal diverso modo in cui trattano la sicurezza della comunicazione, in termini di autenticazione degli attori coinvolti e di protezione da eventuali intrusioni esterne alla comunicazione.
HTTPS è l'acronimo di *Hypertext Transfer Protocol Secure* (noto anche come *HTTPS su TLS* o *HTTP su SSL*), cioè protocollo di trasferimento ipertesto sicuro.
Anch'esso utilizza protocollo TCP per inviare e ricevere pacchetti di dati, ma lo fa tramite la porta *443*, all'interno di una connessione crittografata ds SSL, *Secure Sockets Layer*[15].

[15] Più recentemente è stato introdotto TLS, *Transport Layer Security*.

2 HTTP/1.1 Protocol

2.1 Problemi HTTP/1.0

I principali problemi riscontrati dall'utilizzo del protocollo HTTP/1.0 sono:

- Lentezza e congestione nelle connessioni
- Limitatezza nel numero di indirizzi IP
- Limiti nel meccanismo di autenticazione
- Limiti nel controllo dei meccanismi di caching

2.2 Novità

Le principali novità introdotte dalla versione successiva sono:

- Meccanismo *hop-by-hop*
- Connessioni persistenti e *pipelining*
- Hosting virtuale[16]
- Autenticazione crittografata *digest*
- Nuovi metodi di accesso[17]
- Miglioramento meccanismi di *caching*
  - Gestione molto più sofisticata delle cache
  - Maggiore accuratezza nella specifica validità
  - Header *Cache-Control* per direttive di caching
- *Chunked encoding*[18]

2.3 Header Fields

[19] Header generale	Header di entità	Header di richiesta	Header di risposta
Date	Allow	Authorization	Location
Pragma	Content-Encoding	From	Server
Cache-Control	Content-Length	Referer	WWW-Authenticate
Connection	Content-Type	User-Agent	Proxy-Authenticate

[16] Ad uno stesso IP possono corrispondere nomi diversi e server diversi e rende necessario introdurre header di richiesta *Host* che specifica nome e la porta del server.
[17] Aggiornamento delle risorse sul server e diagnostica.
[18] La risposta può essere inviata al client suddivisa in sottoparti.
[19] In rosso le novità derivate dall'utilizzo del nuovo protocollo.

Trailer	Expires	If-Modified-Since	Retry-After
Transfer-Encoding	Last-Modified	Accept	Accept-Ranges
Upgrade	Content-Language	Accept-Charset	Age
Via	Content-Location	Accept-Encoding	ETag
Warning	Content-MD5	Accept-Language	Vary
	Content-Range	TE	
		Proxy-Authorization	
		If-Match	
		If-None-Match	
		If-Range	
		If-Unmodified-Since	
		Expect	
		Host	
		Max-Forwards	
		Range	

2.4 Connessioni

Introduzione di connessioni *persistenti* e *pipelining* e della possibilità di trasferire coppie multiple di richiesta-risposta in una stessa connessione TCP.

2.4.1 Vantaggi

- Riduzione dei costi (instaurazione ed abbattimento) delle connessioni TCP: 3-way handshake del solo per instaurare la connessione iniziale
- Riduzione della latenza: aumenta la dimensione della finestra di congestione del TCP
- Controllo di congestione a regime
- Migliore gestione degli errori

2.4.2 Problemi

- Concorrenza minore
- Quando chiudere la connessione TCP
  - Può essere aperta sia dal client che dal server
  - Il server può chiudere la connessione
    - Allo scadere di un timeout, applicato sul tempo totale di connessione o sul tempo di inattività di una connessione

    - Allo scadere di un numero massimo di richieste da servire sulla stessa connessione

2.5 Pipeling

Tramissione di più richieste senza attendere l'arrivo della risposta alle richieste precedenti. Le risposte devono essere fornite dal server nello stesso ordine in cui sono state fatte le richieste.

- HTTP non fornisce un meccanismo di riordinamento specifico
- Il server può processare le richieste in un ordine differente da quello di ricezione

Tale tecnica porta ad una riduzione del tempo di latenza ed ottimizzazione del traffico di rete, in particolare per richieste che riguardano risorse molto diverse per dimensioni o tempi di elaborazione.

2.6 Caching

- Header generale *Cache-Control* per permettere a client e server di specificare direttive per il caching
- Header di entità *ETag* per identificare univocamente la versione di una risorsa
- Header di risposta *Vary* per elencare un insieme di header di richiesta da usarsi per selezionare la versione appropriata in una cache
- Header *Via* per conoscere la catena proxy tra client e server

3 HTTP/2 Protocol

3.1 Introduzione

L'obiettivo principale della ricerca e sviluppo di una nuova versione di HTTP è incentrato su tre qualità differenti:

- Semplicità
- Alte prestazioni
- Robustezza

Questi obiettivi vengono raggiunti introducendo funzionalità che riducono la latenza nell'elaborazione delle richieste dei browser con le seguenti tecniche:

- Multiplexing
- Compressione
- Prioritizzazione delle richieste
- Push del server

I meccanismi come il controllo del flusso, l'aggiornamento e la gestione degli errori funzionano come miglioramenti del protocollo HTTP.
Il sistema collettivo consente ai server di rispondere in maniera efficiente con più contenuti di quelli originariamente richiesti di client, eliminando l'intervento dell'utente per chiedere continuamente informazioni, fino a quando il sito web non viene caricato completamente sul browser.
Un'efficiente compressione dei file header HTTP riduce al minimo l'overhead del protocollo e migliora le prestazioni ad ogni richiesta del browser e risposta del server.
Le modifiche ad HTTP/2 sono progettate per mantenere l'interoperabilità e la compatibilità con HTTP/1.1.

3.2 Problemi di HTTP/1.1

HTTP/1.1 era limitato all'elaborazione di una sola richiesta in sospeso per connessione TCP costringendo i browser ad utilizzare più connessioni TCP per elaborare più richieste contemporaneamente.
L'utilizzo di un numero eccessivo di connessioni TCP in parallelo porta ad una congestione di TCP che provoca un eccessivo utilizzo delle risorse di rete: i browser che utilizzano connessioni multiple per elaborare richieste aggiuntive occupano una quantità maggiore delle risorse di rete disponibili, riducendo quindi le prestazioni della reti per altri utenti.
L'emissione di più richieste dal browser provoca anche la duplicazione dei dati sui cavi di trasmissione dei dati, che a loro volta richiedono protocolli

aggiuntivi per estrarre, in corrispondenza dei nodi finale, le informazioni desiderate prive di errori.
L'uso inefficace delle connessioni TCP sottostanti con HTTP/1.1 conduce anche ad una scarsa priorità delle risorse, causando un degrado esponenziale delle prestazioni man mano che le applicazioni web crescono, in termini di complessità, funzionalità e portata.

3.3 Novità HTTP/2

3.3.1 Flussi Multiplex

La sequenza bidirezionale dei frame in formato di testo inviati tramite il protocollo HTTP/2 e scambiati tra il server ed il client è nota come *stream*. Le versioni precedenti del protocollo HTTP erano in grado di trasmettere solo un flusso alla volta, con un certo ritardo per ciascuna trasmissione del flusso. La ricezione di grandi quantità di contenuti multimediali tramite flussi individuali inviati uno ad uno è inefficiente, oltre che dispendiosa, in termini di risorse. Le modifiche introdotte con HTTP/2 hanno contribuito a stabilire un nuovo livello di *framing binario* per risolvere problemi di tale genere.
Questo livello consente a client e server di disintegrare il payload HTTP in una piccola sequenza di frame, indipendente e gestibile, la quale verrà poi riassemblata all'altra estremità.
I formati di *frame binari* consentono lo scambio di sequenze bidirezionali indipendenti multiple, aperte contemporaneamente, senza latenza fra flussi successivi.
I vantaggi che ne derivano sono i seguenti:

- Richieste e risposte multiple in parallelo non si bloccano a vicenda
- Una singola connessione TCP viene utilizzata per garantire un utilizzo efficace delle risorse di rete, nonostante la trasmissione di più flussi di dati
- Non è necessario applicare hack di ottimizzazione, come sprite di immagini, concatenazione e condivisione del dominio, che compromettono altre aree delle prestazioni della rete
- Latenza ridotta, prestazioni web più veloci, posizionamento nei motori di ricerca migliore

Con tale funzionalità, i pacchetti, divisi all'estremità di ricezione e presentati come flussi di dati individuali, provenienti da più flussi vengono essenzialmente mixati e trasmessi su una singola connessione TCP.

3.3.2 Server Push

Tale funzionalità consente al server di inviare al client ulteriori informazioni memorizzabili nella cache che non sono richieste, ma sono previste nelle richieste future.
Tale meccanismo consente di salvare un round trip di richiesta-risposta e riduce la latenza delle rete. Le conseguenze a lungo termine sono incentrate sulla capacità dei server di identificare possibili risorse che possano essere spinte, ma che il client in realtà non desidera.
L'implementazione di HTTP/2 presenta prestazioni significative per le risorse push ed i seguenti vantaggi:

- Client salva le risorse nelle cache
- Client può utilizzare queste risorse memorizzate nella cache su pagine diverse
- Client può rifiutare le risorse inviate per mantenere una repository efficace o disabilitare del tutto Server Push
- Client può anche limitare il numero di flussi push multiplexing contemporaneamente
- Server può eseguire il multiplexing delle risorse inviate insieme alle informazioni richieste all'interno della stessa connessione TCP
- Server può dare priorità alle risorse inviate

Sono presenti funzionalità simili con tecniche non ottimali, come le risposte del server *Inline to Push*, mentre *Server Push* presenta una soluzione ottimale a livello di protocollo.

3.3.3 Protocolli Binari

HTTP/1.x elaborava i comandi di testo per completare i cicli richiesta-risposta, mentre la versione successiva sfrutta comandi binari per eseguire le stesse attività.
Questo attributo riduce le complicazioni con l'inquadramento e semplifica l'implementazione di comandi che sono stati mixati in modo confuso, a causa di comandi contenenti testo e spazi opzionali.
Nonostante i file binari siano più complessi da leggere rispetto ai comandi di testo, è più facile per la rete generare ed analizzare i frame disponibile in binario, anche se la semantica effettiva rimane invariata. I browser che utilizzano HTTP/2 convertiranno gli stessi comandi di testo in binari prima di trasmetterli sulla rete; il livello di framing binario non è retrocompatibile con client e server HTTP/1.x.

I vantaggi della nuova versione del protocollo sono i seguenti:

- Basso overhead nei dati di analisi
- Meno incline agli errori
- Impronta di rete più leggera
- Utilizzo efficace delle risorse di rete
- Eliminazione problemi di sicurezza, come gli attacchi di response splitting
- Rappresentazione compatta dei comandi, per una facile elaborazione ed implementazione
- Efficiente e robusto in termini di elaborazione dei dati tra client e server
- Riduzione latenza di rete e miglioramento del throughput

3.3.4 Proprietà Flussi

L'implementazione di HTTP/2 consente al client di fornire la preferenza per determinati flussi di dati.
Sebbene il server non sia tenuto a seguire queste istruzioni provenienti dal client, il meccanismo consente al server di ottimizzare l'allocazione delle risorse di rete, in base ai requisiti dell'utente finale.
Il server ha la capacità di portare all'arrivo simultaneo di richieste del server che differiscono effettivamente in termini di priorità dal punto di vista dell'utente finale. Il blocco delle richieste di elaborazione del flusso di dati su base casuale compromette efficienza ed esperienza dell'utente finale.

Allo stesso tempo, un meccanismo di prioritizzazione dei flussi intelligente, presenta i seguenti vantaggi:

- Utilizzo efficace delle risorse di rete
- Riduzione tempi di consegna delle richieste del contenuto primario
- Miglioramento velocità di caricamento della pagina e dell'esperienza dell'utente finale
- Comunicazione dati ottimizzata tra client e server
- Ridotto effetto negativo della latenza della rete

3.3.5 Compressione Header con Stato

Il protocollo HTTP è *statless*, il che significa che ogni richiesta del client deve includere tutte le informazioni necessarie al server per eseguire l'operazione desiderata: tale meccanismo fa sì che i flussi di dati trasportino più frame ripetitivi di informazioni, causando un consumo inutile di risorse di rete limitate.
L'implementazione di HTTP/2 risolve questi problemi data la capacità di comprimere un gran numero di frame di intestazione ridondanti, utilizzando le specifiche *HPACK* come approccio semplice e sicuro alla compressione delle intestazioni.
Sia il client che il server mantengono un elenco di intestazioni utilizzate nelle precedenti richieste client-server.
HPACK comprime il valore individuale di ciascun header prima che venga trasferito sul server, il quale cerca le informazioni codificate nell'elenco dei valore dell'header trasferiti in precedenza, per ricostruire le informazioni dell'header complete.

Tale tecnica porta i seguenti vantaggi:

- Priorità esclusiva del flusso
- Utilizzo efficace dei meccanismi multiplexing
- Ridotto sovraccarico di risorse
- Codifica le intestazioi di grandi dimensioni ed utilizzate di frequente

3.4 Similitudini tra HTTP/1.x e SPDY

La semantica delle applicazioni sottostanti al protocollo HTTP rimane la stessa nell'ultima iterazione dell'HTTP/2. Questo si basa su SPDY.

HTTP/1.x	SPDY	HTTP/2
SSL non richiesto, ma raccomandato	SSL richiesto	SSL non richiesto, ma raccomandato
Crittografia lenta	Crittografia veloce	Crittografia ancora più veloce
Una richiesta client-server per ogni connessione TCP	Richiesta client-server multipla per ogni connessione TCP. Si verifica su un singolo host alla volta	Multi-host multiplexing. Si verifica su più host in un singolo istante
Nessuna compressione dell'header	Introdotta la compressione dell'header	Compressione dell'header, attraverso algoritmi che migliorano le prestazioni e la sicurezza
Nessuna priorità di flusso	Introdotta la priorità di flusso	Migliori meccanismi di priorità di flusso

3.5 HTTP/2 con HTTPS

Il supporto di HTTP/2 dei browser include la crittografia HTTPS e completa di fatto le prestazioni generali di sicurezza delle implementazioni HTTPS.
Caratteristiche come il minor numero di handshake TLS, il basso consumo di risorse sia sul lato client che sul lato server e le migliori capacità di utilizzo delle sessioni web esistenti, eliminando al contempo le vulnerabilità associate a HTTP/1.x presentano HTTP/2 come fattore chiave per la sicurezza in ambiti di rete sensibili.

3.5.1 Vantaggi

Il web sta diventando sempre più lento man mano che si popola di volumi crescenti di contenuti multimediali irrilevanti.
Sia per le aziende online, che per le persone online, per accedere più velocemente a contenuti web migliori , vengono sviluppate modifiche HTTP/2 per migliorare l'efficienza nella comunicazione client-server dei dati.

3.5.2 Prestazioni Web

Il confronto tra i benchmark delle performance di HTTPS, SPDY, HTTP/2 dimostrano i miglioramenti nelle prestazioni di HTTP/2, rispetto sia ai predecessori che alle alternative.
La capacità del protocollo di inviare e ricevere più dati per ogni ciclo di comunicazione client-server non è un hack di ottimizzazione, ma un reale e pratico vantaggio di HTTP/2.
Tecnologie come il multiplexing creano uno spazio aggiuntivo per trasportare e trasmettere più dati contemporaneamente.

3.5.3 Prestazioni Web Mobile

HTTP/2 è stato progettato nel contesto delle attuali tendenza di utilizzo del web: multiplexing e compressione dell'header funzionano bene per ridurre la latenza di accesso ai servizi internet su reti dati mobile, che offrono una larghezza di banda limitata.

3.5.4 Internet più economico

HTTP/2 permette di aumentare throughput e di migliorare l'efficienza della comunicazione dati, consentendo ai fornitori di servizi web di ridurre i costi operativi, mantenendo gli standard di Internet ad alta velocità.

3.5.5 Sicurezza

HTTP/2 contiene comandi in binario e consente di comprimere i metadati nell'header, seguendo un approccio *Security by Obscurity* per proteggere i dati sensibili trasmessi tra client e server.
Tale protocollo supporta, in maniera completa, la crittografia e richiede una versione migliorata di *Transport Layer Security* (TLS 1.2) per una migliore protezione dei dati.

Altri minori vantaggi sono, per motivi simili appena citati, Vantaggi SEO di HTTP2, Innovazione, Esperienza Mobile migliorata e ricca di contenuti ed un raggio d'azione esteso.