http2 explained

Daniel Stenberg

Sommario

http2 spiegato

Questo documento descrive HTTP/2 (RFC 7540), il suo background, i concetti base, il protocollo in dettaglio e qualche accenno alle implemetazioni esistenti, oltre ad un rapido sguardo a cosa il futuro potrebbe riservare.

Consultare https://daniel.haxx.se/http2/, pagina ufficiale di questo progetto.

Su https://github.com/bagder/http2-explained il codice sorgente di tutti i contenuti del libro.

CONTRIBUTING

Incoraggio ed accolgo aiuti e contributi da parte di chiunque abbia migliorie da proporre. Accetto pull requests, altrimenti puoi aprire una issue o mandare una email a daniel-http2@haxx.se con i tuoi suggerimenti!

/ Daniel Stenberg

1. Background

Questo documento descrive http2 ad un livello di tecnicismo e protocollo. Tale documento è nato a partire da una presentazione di Daniel a Stoccolma in Aprile 2014, presentazione che è stata successivamente convertita ed estesa fino a diventare un vero e proprio documento, ben più dettagliato e contentente esplicazioni piu proprie.

RFC 7540 è il nome ufficiale della specifica finale http2, pubblicata il 15 Maggio 2015: https://www.rfc-editor.org/rfc/rfc7540.txt

Ogni eventuale errore in questo documento è il risutato della mia pigrizia e stoltezza, pregasi segnalarne eventuale presenza; tali errori saranno corretti in versioni successive.

In questo documento ho provato ad usare in maniera estensiva il termine "http2" per desrcivere il nuovo protocollo, mentre in pura termininologia tecnica il vero nome sarebbe (è) HTTP/2. Ho fatto questa scelta per migliorare la leggibilità e per favorive una migliore scorrevolezza del linguaggio.

1.1 L'autore

Il mio nome è Daniel Stenberg e lavoro per Mozilla. Ho lavorato in ambito open source e networking per più di vent'anni all'interno di svariati progetti. Molto probabilmente sono meglio conosciuto per essere il principale sviluppatore di curl e libcurl. Sono stato coinvolto nel gruppo di lavoro IETF HTTPbis per svariati anni ed assieme al gruppo ho avuto la possibilità di rimanere sempre aggiornato a proposito del progetto HTTP 1.1 per poi in seguito partecipare alla standardizzazione di http2.

Email: daniel@haxx.se

Twitter: @bagder

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2 Aiuto!

Se dovessi trovare errori, omissioni, sbagli o falsità in questo documento, ti pregherei di mandarmi una versione corretta del paragrafo in questione; mi occuperò di redigerne una versione corretta. Darò tutto il credito a chiunque aiuti concretamente! Spero di rendere questo documento sempre più completo col passare del tempo.

Questo documento è disponibile su https://daniel.haxx.se/http2

1.3 Licenza

1.4 Storia del documento

La prima versione del presente documento fu pubblicata il 25 Aprile 2014. Di seguito le migliorie e bugfix delle più recenti versioni.

Versione 1.13

- Convertita la sezione principale del documento a sintassi Markdown
- 13: Citare più risorse, link e descrizioni aggiornate
- 12: Aggiornata la descrizione di QUIC con riferimento alla draft "ufficiale"
- 8.5: Aggiornata alla numerazione attuale
- 3.4: La media è al momento 40 connessioni TCP
- 6.4: Aggiornato per aderire a quanto affermato nella specifica

Versione 1.12

- 1.1: HTTP/2 è oramai una RFC ufficiale
- 6.5.1: Collegamento alla RFC su HPACK
- 9.1: Menzionare come Firefox 36+ configuri http2 "on by default"
- 12.1: Aggiunta sezione a proposito di QUIC

Versione 1.11

- Molti miglioramenti dal punto di vista del linguaggio, grazie a molti contributi amichevoli
- 8.3.1: Citare le attività specifiche di nginx e Apache httpd

Versione 1.10

- 1: Il protocollo ha ricevuto l'OK
- 4.1: Accordare i tempi passati rispetto al fatto che 2014 è oramai tempo passato
- Front: Aggiunta immagine con nome “http2 explained”, link corretto
- 1.4: Aggiunta sezione "Storia del documento"
- Molti errori grammaticali e di spelling corretti
- 14: Aggiunto ringraziamento ai bug reporters
- 2.4: Migliori etichette per il grafico sulla evoluzione di HTTP
- 6.3: Corretto l'ordine dei vagoni nel "treno multiplexato"
- 6.5.1: draft-12 HPACK

Version 1.9

- Aggiornato alla draft-17 HTTP/2 e draft-11 HPACK
- Aggiunta sezione "10. http2 in Chromium" (== più lungo di una pagina)
- Un sacco di fix spelling
- A quota 30 implementazioni ad oggi
- 8.5: Aggiunti alcune cifre relative all'utilizzo
- 8.3: Citare anche internet explorer
- 8.3.1 Aggiunto "implementazioni mancanti"
- 8.4.3: Menzionare come TLS incrementi anche il tasso di successo

2. HTTP oggi

HTTP 1.1 si è trasformato in un protocollo utilizzato (virtualmente) per qualsiasi scopo su Internet. Grandi investimenti sono stati fatti su questo protocollo e su infrastrutture che possano trarre beneficio da quest'ultimo, al punto che ad oggi è più semplice appoggiarsi ad HTTP piuttosto che creare da zero qualcosa di nuovo.

2.1 HTTP 1.1 è vasto, vastissimo

Al tempo in cui HTTP fu concepito e messo a disposizione dell'umanità, fu più probabilmente percepito come un protocollo semplice e diretto; tale teoria è stata poi negata nel corso del tempo. La RFC 1945 del 1996 su HTTP 1.0, consta di 60 pagine. La RFC 2616 che descrive HTTP 1.1 -rilasciata solo tre anni dopo, nel 1999- conta ben 176 pagine. Ebbene, quando la IETF ha approciato il lavoro sull'aggiornamento delle specifiche ha sentito il bisogno di suddividere il documento in sei parti, risultando così in un conteggio di pagine ancora più elevato (vedi RFC 7230 e correlate). In parole povere, HTTP 1.1 è IMMENSO ed include una miriade di dettagli, sottigliezze e -non meno importante- un elevato numero di parti opzionali.

2.2 Un mondo di opzioni

Alla luce della vastità di dettagli ed opzioni disponibili per estensioni future e dunque in perfetta relazione con la natura di HTTP 1.1, un vasto ecosistema software si è sviluppato; possiamo tuttavia notare come quasi nessuna implementazione implementi davvero tutto - ed è infatti quasi impossibile definire cosa questo "tutto" in effetti rappresenti. Questo ha fatto sì che certune caratteristiche (non utilizzate fin dall'inizio) non abbiano mai suscitato l'interesse di alcuni sviluppatori, e che di converso altri gruppi abbiano implementato esattamente tutte le funzioni, riscontrando però un basso utilizzo delle stesse.

Tutto ciò ha causato problemi di inter-operabilità, al momento in cui clients e servers hanno cominciato ad utilizzare sempre più estensivamente tali suddette caratteristiche ed estensioni. Il pipelining HTTP è un esempio principe di tale feature (e del suo stato di implementazione).

2.3 Uso inappropriato del TCP

Per HTTP 1.1 non è stato facile arrivare a trarre pieno beneficio dalla potenza e dalle performances che TCP mette teoricamente a disposizione. Client e server HTTP devono essere creativi per riuscire a diminuire il tempo di caricamento delle pagine.

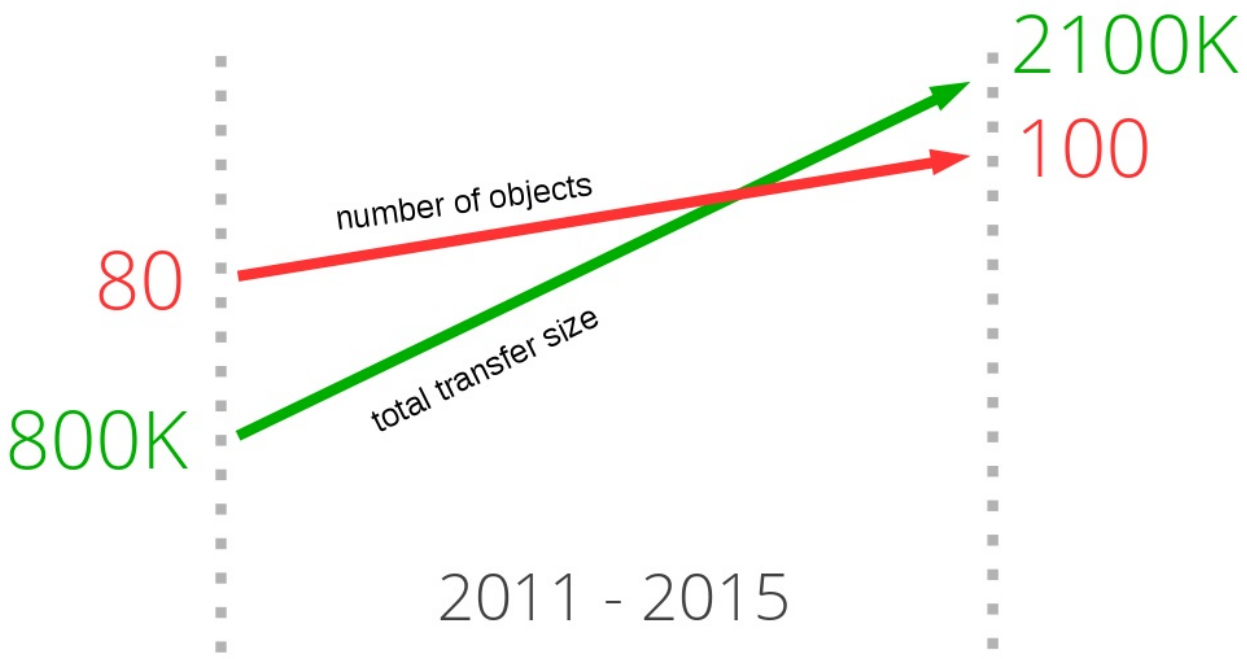
Altri tentativi perpetrati in parallelo nel corso degli anni hanno confermato quanto il TCP non sia facilmente rimpiazzabile; si continua perciò a lavorare sul miglioramento del TCP e dei protocolli su di esso costruiti.

Mettiamola in termini semplici, il TCP può essere impiegato per evitare pause, intervalli o tempi morti in genere, frazioni di tempo che potrebbero altrimenti essere utilizzate per spedire e ricevere quantità di dati. Le sezioni seguenti evidenzieranno alcuni di questi prevedibili effetti.

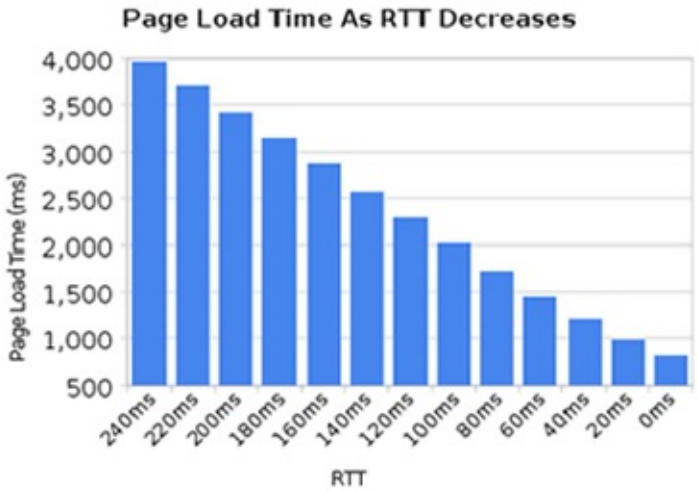
2.4 Dimensione dei trasferimenti e numero di oggetti

Se osserviamo i trend per alcuni dei più popolari siti web odierni e quanto tempo prenda il download iniziale di tali "welcome pages" emerge chiaramente un pattern. Nel corso degli anni la quantità di dati di cui necessitiamo non ha fatto che aumentare verso e oltre gli 1.9MB. Cosa ancora più importante emerge da questo contesto: in media, più di 100 risorse individuali sono necessarie per visualizzare una determinata pagina.

Come mostra il grafico sotto, il trend è in corso da un pò di tempo e non mostra alcun segno di arresto o inversione sul breve periodo. Il grafico mostra la crescita della dimensione del trasferimento (in verde), il numero totale di richieste/risposte (in rosso) necessarie per servire il contenuto web dei siti più popolari al mondo, e come tali cifre si siano evolute nel corso degli ultimi quattro anni.



2.5 La latenza uccide



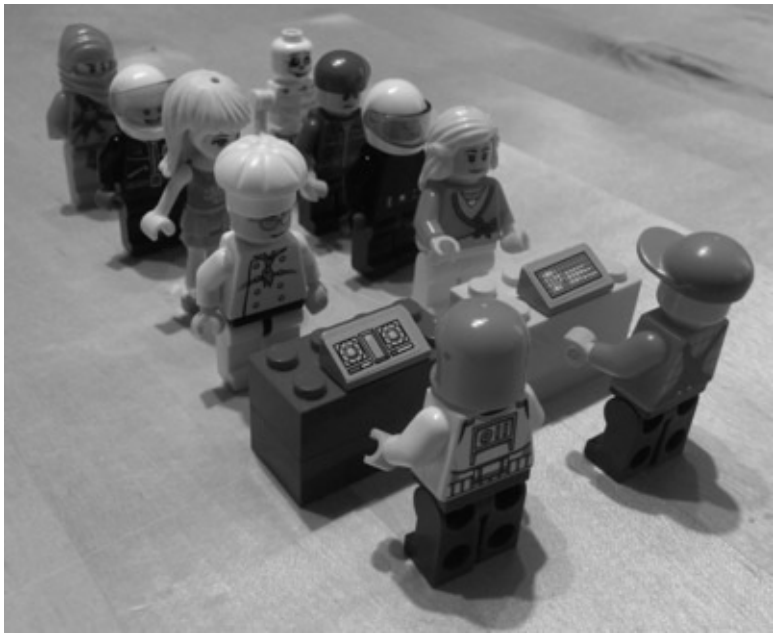
HTTP 1.1 è molto sensibile alla latenza, in parte perchè il pipelining HTTP non ha mai sufficientemente risolto i propri problemi, tanto da -infine- meritare la disattivazione presso la maggior parte dell'utenza.

Mentre abbiamo potuto osservare un enorme incremento di larghezza di banda disponibile "per utente" durante il corso degli ultimi anni, non abbiamo potuto osservare lo stesso incremento di prestazione volto a ridurre la latenza. Su link ad elevata latenza come ad esempio la maggior parte delle tecnologie mobili, è perciò difficile godere di buone prestazioni web pur disponendo di una connessione a banda larga.

Un altro use-case che richiede bassa latenza è il video, video conferencing, gaming e simili, là dove non vi è la necessità di inviare uno stream "prestabilito o ordinato".

2.6. Bloccaggio "head-of-line" (inizio della fila)

Il cosiddetto "pipelining HTTP" è un metodo che permette di inviare una ulteriore richiesta mentre stiamo ancora aspettando la risposta alla domanda precendente. In un certo senso è come fare la coda alla banca o al supermarket: non sai in anticipo se la persona davanti si rivelerà essere un cliente veloce o uno di quei/quelle fastidiosi/e che impiegheranno un'eternità. Questo problema è noto come "head-of-line", inizio della coda (o della fila).



Certo, puoi provare a scegliere la fila che pensi essere la più rapida o potresti addirittura iniziarne una nuova tu. Alla fine, non possiamo evitare di prendere decisioni. Una volta compiuta la scelta non si puè più cambiare.

Creare una nuova coda è allo stesso tempo associato ad una penalità in termini di risorse e performance; non consiste dunque in una soluzione scalabile oltre ad un piccolo numero di code. Non esiste soluzione ideale a questo problema.

Ancora oggi, la maggior parte dei browser "desktop" ha scelto di mantenere il pipelining disattivato.

Ulteriori risorse sull'argomento disponibili ad esempio sul bugzilla bugzilla entry 264354 di Firefox.

3. Tecniche applicate al contrasto della latenza

Quando si trovano faccia a faccia con questo tipo di problema, le persone si ritrovano e cercano una scappatoia. Alcune di queste soluzioni saranno intuitive ed intelligenti, altre orrendi accrocchi.

3.1 Spriting

"Spriting" è il termine usato per descrivere la tecnica che combina immagini multiple in un'unica immagine piu ampia. Attraverso JavaScript o CSS si può in seguito provare a decomporre l'immagine riassemblata nei singoli componenti.

Un sito utilizzerà spesso questa astuzia per migliorare la percezione di velocità. Acquisire una singola grande immagine in HTTP 1.1 è molto più veloce che richiedere 100 minuscole immagini separate.

Naturalmente, metodo svantaggioso se si desidera mostrare solo una o due immagini frammentarie. Lo spriting (spezzettamento?) implica anche che tutte le immagini vengano cancellate dalla cache allo stesso momento, piuttosto che permettere la persistenza dei frammenti richiamati più frequentemente, impattando quindi le performances..

3.2 Inlining

Lo "inlining" è un'altra tecnica utilizzata per evitare di spedire immagini individuali (tramite connessioni separate); tale tecnica consiste nell'utilizzare URL di tipo "data" incorporati nel CSS, con benefici e svantaggi simili al caso di "spriting" precedente.

```
.icon1 {
    background: url(<data>) no-repeat;
}

.icon2 {
    background: url(<data>) no-repeat;
}
```

3.3 Concatenazione

Un sito abbastanza vasto può contenere numerosi file JavaScript differenti. Gli sviluppatori possono utilizzare dei cosiddetti "frontend" per concatenare -o combinare- molteplici scipts e far sì che il browser possa richiedere tutto iu un singolo grande file piuttosto che segmentando la richiesta su dozzine di file minuscoli. Molti dati sono inviati benchè ne possano servire

meno, di conseguenza una quantita sempre maggiore di oggetti deve essere invalidata e "rinfrescata" al modificarsi di ogni singola risorsa inclusa (effetto negativo sulla cache).

Questa prassi è -ovviamente- un inconveniente per gli sviluppatori.

3.4 Sharding (partizionamento. differenziamento)

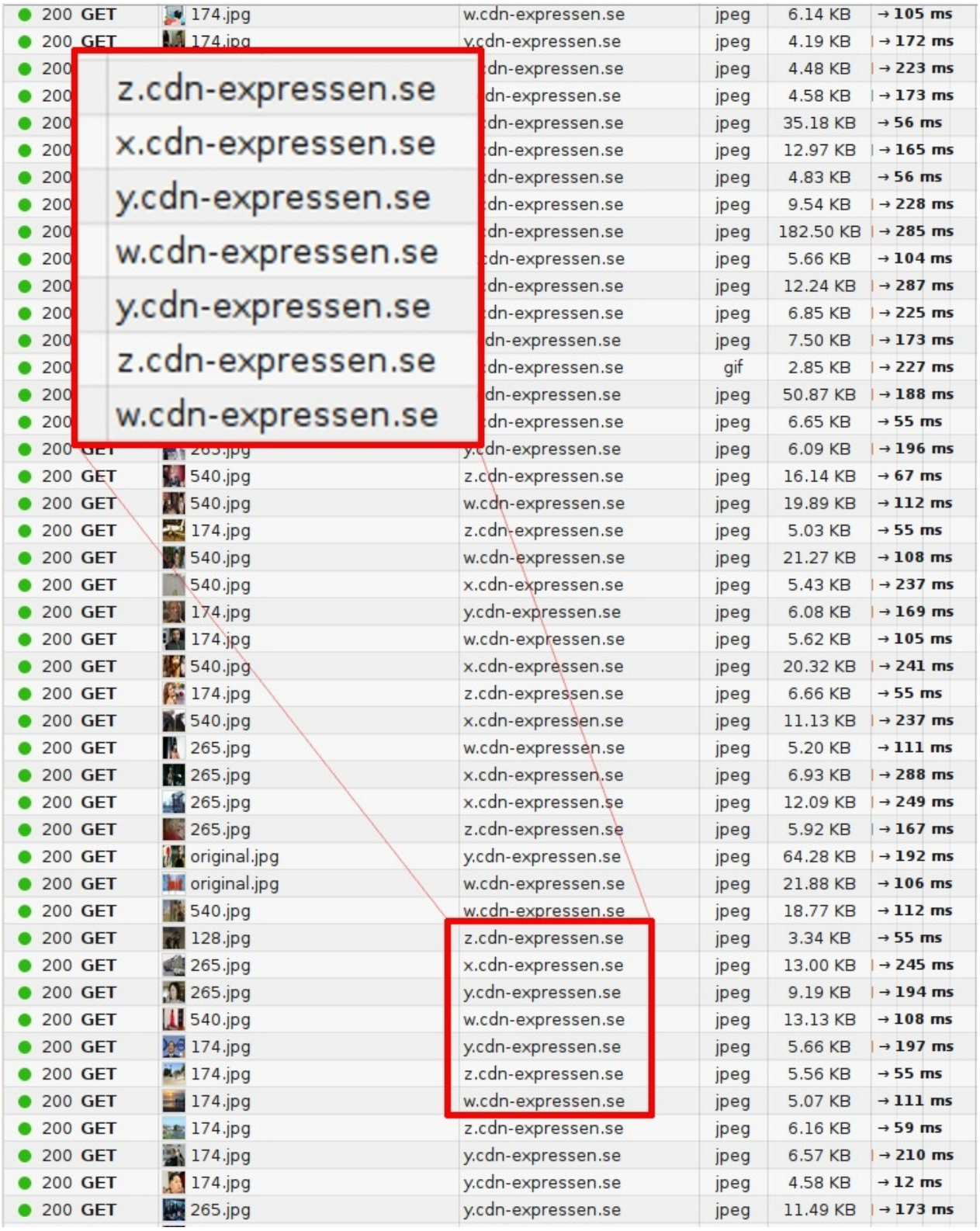
L'ultimo trick per incrementare le performance che citerò è spesso chiamato "sharding". In breve consiste nell'utilizzare il maggior numero di hosts per fornire un determinato servizio. Anche se a prima vista può sembrare strano vi è una ragione ben precisa dietro l'utilizzo di questa tecnica.

All'inizio le specifiche HTTP 1.1 dichiaravano che un client fosse autorizzato ad utilizzare un massimo di due [2] connessioni TCP per server/host; dunque per non violare le specifiche, gli sviluppatori hanno iniziato ad utilizzare alias e nuovi nomi alternativi -et voilà- molte più connessioni disponibili verso il proprio sito e maggior velocità di caricamento (page load, page speed).

Nel tempo la restrizione è stata rimossa; ad oggi un client utilizza facilmente dalle sei alle otto connessioni per nome-host. Sono tuttavia ancora limitate, perciò i siti continuano ad utilizzare questa vecchia tecnica per incrementare il numero di connessioni disponibili per ogni singolo client. Visto che il numero di oggetti richiesti via HTTP continua a crescere -come mostrato sopra- l'ampio ammontare di connessioni è utilizzato per garantire che HTTP continui a rispondere con prestazioni decorose, permettendo al sito di caricare velocemente. Non è infrequente per un sito utilizzare 50, 100 o più connessioni. Statistiche del httparchive.org mostrano come i top 300mila URL al mondo in media necessitino almeno di 40 (!) connessioni TCP ciascuno; tale trend sembra aumentare lentamente nel tempo.

Un altro scenario nel quale si può utilizzare lo sharding è l'hosting di immagini, appunto su macchine (hosts) separate, in modo da non dover utilizzare o memorizzare alcun cookie, tenendo ben presente che il volume dei cookie è anch'esso in aumento. Utilizzando hosts senza cookie associati permette di aumentare le performances semplicemente riducendo il volume della richiesta HTTP !

L'immagine qui sotto mostra una trace (packe capture) acquisita durante la navigazione su uno dei piu famosi siti Svedesi, e dimostra come le richieste siano distribuite su svariati hostnames.



200 GET	174.jpg	w.cdn-expressen.se	jpeg	6.14 KB	→ 105 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.19 KB	→ 172 ms
200		dn-expressen.se	jpeg	4.48 KB	→ 223 ms
200		dn-expressen.se	jpeg	4.58 KB	→ 173 ms
200		dn-expressen.se	jpeg	35.18 KB	→ 56 ms
200		dn-expressen.se	jpeg	12.97 KB	→ 165 ms
200		dn-expressen.se	jpeg	4.83 KB	→ 56 ms
200		dn-expressen.se	jpeg	9.54 KB	→ 228 ms
200		dn-expressen.se	jpeg	182.50 KB	→ 285 ms
200		dn-expressen.se	jpeg	5.66 KB	→ 104 ms
200		dn-expressen.se	jpeg	12.24 KB	→ 287 ms
200		dn-expressen.se	jpeg	6.85 KB	→ 225 ms
200		dn-expressen.se	jpeg	7.50 KB	→ 173 ms
200		dn-expressen.se	gif	2.85 KB	→ 227 ms
200		dn-expressen.se	jpeg	50.87 KB	→ 188 ms
200		dn-expressen.se	jpeg	6.65 KB	→ 55 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	6.09 KB	→ 196 ms
200 GET	540.jpg	z.cdn-expressen.se	jpeg	16.14 KB	→ 67 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	19.89 KB	→ 112 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.03 KB	→ 55 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	21.27 KB	→ 108 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	5.43 KB	→ 237 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.08 KB	→ 169 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.62 KB	→ 105 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	20.32 KB	→ 241 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.66 KB	→ 55 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	11.13 KB	→ 237 ms
200 GET	265.jpg	w.cdn-expressen.se	jpeg	5.20 KB	→ 111 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	6.93 KB	→ 288 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	12.09 KB	→ 249 ms
200 GET	265.jpg	z.cdn-expressen.se	jpeg	5.92 KB	→ 167 ms
200 GET	original.jpg	y.cdn-expressen.se	jpeg	64.28 KB	→ 192 ms
200 GET	original.jpg	w.cdn-expressen.se	jpeg	21.88 KB	→ 106 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	18.77 KB	→ 112 ms
200 GET	128.jpg	z.cdn-expressen.se	jpeg	3.34 KB	→ 55 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	13.00 KB	→ 245 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	9.19 KB	→ 194 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	13.13 KB	→ 108 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	5.66 KB	→ 197 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.56 KB	→ 55 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.07 KB	→ 111 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.16 KB	→ 59 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.57 KB	→ 210 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.58 KB	→ 12 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	11.49 KB	→ 173 ms

4. Aggiornare HTTP

Non sarebbe carino creare un protocollo più potente ? Si, si ..

1. Meno sensibile alla latenza
2. Con un miglior pipelining e fine dei bloccaggi "inizio della fila"
3. Che elimini il bisogno di incrementare il numero di connessioni per host
4. Che mantenda tutte le interfacce esistenti, i contenuti, i formati URI e gli schemi
5. Che sia sviluppato a contatto con il gruppo di lavoro IETF HTTPbis

4.1. L'IETF e il gruppo di lavoro HTTPbis

La Internet Engineering Task Force (IETF) è una organizzazione che sviluppa e promuove gli standard dell'internet, soprattutto a livello di definizione di protocolli. Sono da sempre consosciuti per le serie di memo RFC riguardanti argomenti quali TCP, DNS e FTP, best practices, HTTP e numerose varianti che non si sono mai trasformate in niente di più.

All'interno di IETF, gruppi di lavoro dedicati sono stati istituiti al fine di lavorare in direzione di un obiettivo comune, in un dominio limitato. Stabiliscono un "charter" ed alcune linee-guida (e limiti) rispetto alle aspettative. Chiunque è benvenuto ad aggiungersi, a partecipare alla discussione e allo sviluppo. Chiunque partecipi e dica qualcosa di concreto ha lo stesso peso nell'influenzare l'esito del lavoro finale ed intermedio. Tutti i membri sono considerati allo stesso livello, senza troppa importanza rispetto al datore di lavoro.

Il gruppo di lavoro HTTPbis (vedi più avanti per la spiegazione del nome) è stato formato durante l'estate 2007 ed incaricato di creare un aggiornamento delle specifiche HTTP 1.1. All'interno del gruppo la discusssione a proposito di una futura versione è iniziata in tardo 2012. Il lavoro di aggiornamento su HTTP 1.1 è stato completato nel 2014 ed ha prodotto come risultato la serie di RFC 7230

Il meeting finale per il GdL HTTPbis inter-op si è tenuto a New York City ad inizio Giugno 2014. Le restanti discussioni e le procedure IETF svolte per pubblicare definitivamente la RFC si sono svolte fino all'anno successivo.

Alcuni dei più grandi attori in campo di HTTP sono spariti dalle discussioni e dagli incontri del gruppo di lavoro. Non voglio citare nessuna compagnia in particolare o alcun nome di prodotto ma è chiaro come alcuni protagonisti dell'Internet di oggi sembrino essere convinti che la IETF se la caverà da sola, senza bisogno che tali grosse compagnie siano coinvolte...

4.1.1. La parte "bis" nel nome

Il gruppo si chiama HTTPbis dove "bis" deriva dall'avverbio latino significante "secondo" Latin adverb for two. Bis è comunemente usato in contesto IETF come suffisso o parte del nome per un aggiornamento o una review di una specifica; in tal caso si tratta dell'aggiornamento di HTTP 1.1.

4.2. http2 ha preso vita da SPDY

SPDY è un protocollo sviluppato e diffuso da Google. Lo hanno sicuramente sviluppato in maniera aperta, invitando chiunque a partecipare; era ovvio sin dall'inizio che Google ne avrebbe tratto beneficio, mantenendo il controllo su una popolare implementazione di browser e su una significativa popolazione di server con servizi arcinoti.

SPDY aveva gia ampiamente provato la sua validità concettuale nel momento in cui HTTPbis decise che fosse tempo di iniziare a lavorare su http2. SPDY ha mostrato come fosse possibile innovare internet, c'erano statistiche e numeri a riprova delle sue ottime performances. Il lavoro su http2 è cominciato con la draft di SPDY/3 che di fatto è stata trasformata nella draft-00 di http2 con un colpo di sed/replace.

5. http2 a grandi linee

Dunque, che innovazione porta http2? Fino a che punto si spingono i limiti imposti dal gruppo HTTbis ?

I confini sono abbastanza limitati e pongono al team molte restrizioni sull'abilità di innovare.

- http2 deve mantenere i paradigmi HTTP. Rimane un protocollo in cui un client manda una richiesta ad un server via TCP.
- gli URL http:// e https:// non possono essere modificati. Non può esistere un nuovo schema. La quantità di contenuti che utilizzano tali URL è troppo grande per pensare ad un cambiamento.
- servers e clients HTTP1 esistono da decadi, dobbiamo essere in grado di "proxyficare" (traghettare, incapsulare) tutto verso servers http2.
- di conseguenza un proxy deve essere capace di mappare le features di http2 uno-a-uno su un client HTTP 1.1.
- Rimuovere o ridurre parti opzionali del protocollo. Non esattamente un requisito ma piuttosto un mantra proveniente dalla filosofia SPDY e dal team Google. Se si fa in modo che tutto sia obbligatorio fin da subito non si arriverà mai ad "implementare oggi" senza cadere in un possibile bug domani.
- Basta con le versioni minori. È stato deciso che i client ed i server o sono compatibili con http2 oppure no, in maniera assoluta. Se ci sarà bisogno di estendere il protocollo o modificare le cose, http3 prenderà forma. Non ci sono più versioni intermedie in http2.

5.1. http2 per gli schemi URI gia esistenti

Come gia detto, gli schemi URI esistenti non possono essere modificati, dunque http2 deve utilizzare gli esistenti. Visto che sono utilizzati per HTTP 1.x ad oggi, abbiamo necessariamente bisogno di una maniera per upgradare il protocollo ad http2, o altrimenti domandare al server di utilizzare http2 piuttosto che protocolli precedenti.

HTTP 1.1 possiede gia un modo di procedere a tale upgrade; l'omonimo header "Upgrade:" permette al server di inviare una risposta utilizzando il nuovo protocollo ogni volta che ricevesse una richiesta sul vecchio, al costo di un solo round-trip aggiuntivo.

La penalità imposta da tale round-trip non era una condizione che il team SPDY avrebbe potuto accettare, e visto che hanno implementato SPDY solo su TLS, hanno sviluppato una nuova estensione TLS che raccorcia la nettamente la negoziazione. Utilizzando questa estensione detta NPN (Next Protocol Negotiation) il server istruisce il client a proposito di quali protocolli siano di sua conoscenza e lascia al client la libera scelta fra quelli supportati.

5.2. http2 per https://

Molta attenzione è stata messa per far si che http2 si comporti degnamente su TLS. SPDY necessita di TLS, di conseguenza si è attuata una discreta pressione per rendere TLS obbligatorio per http2, ma infine tale mozione non ha ricevuto il consenso dunque per http2, TLS rimane opzionale. Ciononostante, due dei partecipanti hanno chiaramente espresso la volontà di implementare http2 esclusivamente su TLS: i capi di Mozilla Firefox e Google Chrome, due dei principali browser esistenti.

Le ragioni per propendere per il TLS-esclusivo includono sicuramente il rispetto della privacy dell'utente e le misurazioni iniziali, che hanno mostrato come http2 abbia un tasso di successo superiore quando in presenza di TLS. Ciò a causa del fatto che molte box intermedie (router, firewall etc) sono convinte che tutto ciò che passa per la porta 80 sia sempre e solo traffico HTTP 1.1, il che crea interferenze e drop in caso si cerchi di utilizzare un altro protocollo.

Tale argomento (TLS obbligatorio) ha causato svariati disaccordi sulle mailing lists e durante gli incontri - è buono o cattivo ? Rimane un giudizio controverso - attenzione a non tirare fuori questo argomento davanti ad un membro della HTTPbis !

Un altro estenuante dibattito ha avuto luogo, riguardo alla decisione se http2 debba o meno fornire una lista dei cifrari che dovrebbero obbligatoriamente essere impiegati durante un dialogo TLS, o se almeno dovesse fornire una blacklist, o se invece non dovesse richiede assolutamente nulla a livello di TLS, lasciando tale decisione al gruppo di lavoro TLS. Risultato del dibattito, la specifica detta una versione minima di TLS 1.2 e menziona restrizioni rispetto agli algoritmi di cifratura.

5.3. negoziazione di http2 su TLS

NPN (Next Protocol Negotiation) è il protocollo impiegato per negoziare SPDY quando si ha a che fare con server TLS. Visto che non era un vero e proprio standard, è stato incorporato per mano della IETF fino ad arrivare a diventare ALPN: Application Layer Protocol Negotiation. L'uso di ALPN è incoraggiato (richiesto) da http2, mentre SPDY rimane basato su NPN.

Il fatto che NPN esistesse gia da prima e che poi ALPN abbia preso forma tramite standardizzazione, ha fatto sì che diversi clients e servers abbiano implementato (e pretendano di utilizzare) entrambe le estensioni in presenza di http2. Inoltre come se non bastasse, NPN è utilizzato da SPDY, molti server supportano sia http2 sia SPDY: è dunque logico che tali server supportino entrambi, NPN e ALPN.

ALPN è diverso da NPN principalmente rispetto a chi prenda la decisione sul protocollo da adottare durante un dialogo. Con ALPN, il client fornisce al server una lista di protocolli che è in grado di supportare, in ordine di preferenza; il server si occupa poi di scegliere quelle che preferisce, mentre in NPN è il client che prende tale decisione finale.

5.4. http2 per http://

Come detto prima, per negoziare http2 sul vecchio HTTP/1.1, dobbiamo inviare il famoso header "Upgrade:". Se il server parla http2, risponderà con un messaggio di stato "101 Switching" e da quel momento in poi tenterà di parlare solo http2 nel contesto di quella connessione. Di certo, questa procedura di upgrade ci costa un intero round-trip ma il vantaggio è che poi è possibile mantenere e riutilizzare una connessione http2 per molto più tempo rispetto ad una tipica HTTP/1.1 vecchio stile.

Mentre un portavoce di una casa produttrice di browser ha annunciato di non voler implementare questa maniera di parlare http2, il team di Internet Explorer ha affermato di volerlo fare, benchè poi non abbiano mai mantenuto la promessa. Solo curl e pochi altri non-browsers supportano http2 in clear-text.

Ad oggi, nessuno dei maggiori browser supporta http2 al di fuori di TLS.

6. Il protocollo http2

Si è detto abbastanza della storia e delle ragioni politiche che ci hanno accompagnati fin qui. Passiamo all'analisi delle specifiche del protocollo: i fondamenti e i concetti di cui http2 si nutre.

6.1. Binario

http2 è un protocollo binario.

Lasciamo questo concetto da parte per qualche minuto. Se sei stato coinvolto in protocolli internet prima d'ora, ci sono probabilità che reagirai istintivamente a questa scelta argomentando con fervore quanto un protocollo text/ascii sia più adatto e più versatile, visto che ci permette di instaurare richieste e sessioni via telnet e strumenti simili...

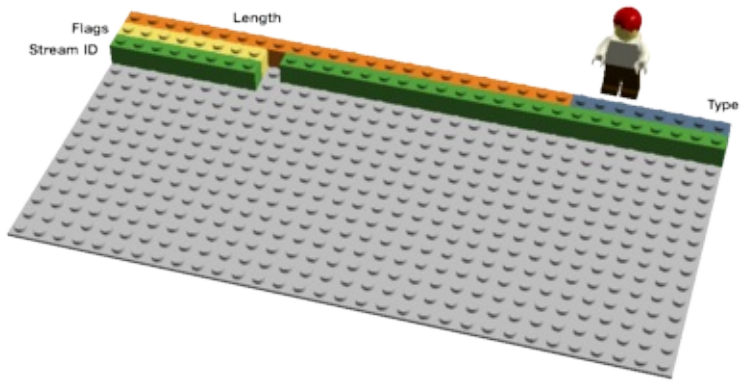
http2 è binario per permettere un incapsulamento più dinamico. Trovare inizio e fine di un frame è uno dei compiti più complessi in HTTP 1.1 e per tutti i protocolli text-based in generale. L'implementazione si rende più semplice allontanandosi da "spazi opzionali" e modi diversi di scrivere la stessa cosa.

Allo stesso tempo rende più semplice separare protocollo da incapsulamento - cosa che in HTTP 1.1 presenta interdipendenza.

Il supporto nativo per compressione assieme all'onnipresente TLS, diminuiscono il valore di un protocollo puramente testuale, dato che non si vedrebbe alcun testo in una capture on-the-wire. Dobbiamo semplicemente abituarci all'idea di usare uno strumento di cattura/analisi tipo Wireshark al fine di comprendere cosa succeda a livello di http2.

Per operare un buon debug su questo protocollo si avrà probabilmente bisogno di strumenti quali curl, analisi di traffico IP con dissector http2 tipo Wireshark e simili.

6.2. Il formato binario



https spedisce frame binari. Diversi tipi di frame possono essere inviati, pur condividendo la stessa fase di inizializzazione: Length, Type, Flags, Stream Identifier, e payload del frame.

Esistono dieci tipi diversi di frame definiti nella specifica http2. Probabilmente due dei più fondamentali e che più intuitivamente possono essere ricondotti ad HTTP 1.1 sono DATA e HEADERS. Li descriverò in dettaglio successivamente.

6.3. Flussi multiplexati

L'ID dello Stream menzionato nella sezione precedente associa ogni frame inviato via http2 ad un preciso "stream". Durante una connessione http2 ogni stream è indipendente, una sequenza bidirezionale di frame viene scambiata fra client e server.

Una sola connessione http2 può contenere molteplici stream -aperti allo stesso tempo- all'interno dei quali ogni endpoint accavalla frame provenienti da streams differenti. Gli streams possono essere aperti ed utilizzati unilateralmente oppure essere condivisi sia dal client sia dal server e possono essere chiusi da entrambi gli endpoint. L'ordine in cui tali frames sono inviati nel contesto dello stream è importante, caratteristico. I destinatari elaborano i frame nell'ordine in cui essi sono ricevuti.

"Multiplexare" gli streams significa che i contenuti di diversi streams sono mescolati nella stessa connessione. Due (o più) flussi di dati sono riuniti in uno singolo, per poi essere nuovamente divisi e riassemblati (individualmente) all'altro capo della connessione. Ecco due flussi (treni di dati, NdR):

Ecco due "treni" multiplexati all'interno di una sola connessione:

6.4. Priorità e dipendeze

Ogni stream è anche dotato di priorità, o "peso", priorità che viene impiegata per rendere noto all'altro estremo quale sia lo stream da considerare di maggior importanza, ad esempio in caso la penuria di risorse dovesse imporre al server di scegliere quale flusso inviare per primo.

Utilizzando il frame PRIORITY, un client può anche istruire il server su quali altri stream dipendano dallo stesso. Permette al client di definire un sistema di priorità (albero) ove diversi rami figlio possano dipendere dal completamento di altri rami padre (parent streams).

Le priorità e le dipendenze possono essere modificate dinamicamente a runtime, cosa che dovrebbe permettere ai browsers di poter decidere -quando un utente stesse scrollando una pagina piena di immagini- quali di queste immagini siano prioritarie, più importanti, o abbiano lo scopo di accelerare lo scorrimento fra diversi tab (con conseguente miglioramente a livello di reattività del focus).

6.5. Compressione degli header

HTTP è un protocollo senza memoria di stato. In brebe, ogni singola richiesta deve portarsi con sè più dettagli possibili, liberando così il server dall'incombenza del dover memorizzare molte informazioni e metadati relativi a richieste precedenti. Dato che http2 non modifica questo paradigma, deve poter continuare a lavorare nello stesso modo.

Ciò fa si che HTTP sembri (e sia) ripetitivo. Quando un client richiede una risorsa dallo stesso server, come i.e. le immagini di una pagina web, un alto numero di richieste verrò generato, e tali richieste sembreranno tutte molto simili fra di loro. Per tale serie sempre ripetitiva, la compressione sarebbe una benedizione.

Come il numero di oggetti per pagina è aumentato, così l'utilizzo dei cookies e la dimensione delle richieste non sono da meno, continuando ad aumentare nel tempo. I cookies devono essere inclusi in ogni richiesta, decine e decine di volte gli stessi elementi sono ritrasmessi.

Le richeste HTTP 1.1 sono al momento diventate così voluminose da eccedere la dimensione della finestra TCP iniziale, cosa che a suo turno rende abbastanza lenta l'operazione, vista la necessita di interi round-trip prima di ricevere ACK, e dunque ben prima che l'intera richiesta sia inviata (e di conseguenza processata). Questo è un altro punto a favore della compressione.

6.5.1. Parlare di compressione è un argomento spinoso

HTTPS e SPDY hanno sofferto a causa di vulnerabilità BREACH e attacchi CRIME. Inserendo un testo noto nello stream ed osservandone i mutamenti in funzione dell'operazione richiesta, un attaccante può scoprire il contenuto del payload.

Creare un meccanismo di compressione dinamica per un protocollo - senza renderlo vulnerabile ad uno di questi attachi- richiede molta cura e considerazione. Proprio quello che HTTPbis ha cercato di fare.

Parliamo quindi di HPACK, Header Compression for HTTP/2, che – come il nome suggerisce - è un formato di compressione creato su misura per gli headers http2, descritto e specificato in una internet draft separata. Il nuovo formato, cosi come le contromisure adottabili (quali un bit che richiede agli intermediari di non comprimere un header specifico ne padding opzionale), dovrebbe rendere più difficile lo sfruttamento della vulnerabilità nel contesto di un attacco.

Nelle parole di Roberto Peon (uno dei creatori di HPACK):

> “HPACK è stato concepito per impedire ad una implementazione non-conforme di facilitare la fuga di informazioni, per rendere ecoding e decoding rapidi e semplici, per permettere al ricevente di controllare la compressione sulla base delle dimensioni del contesto, per permettere ad un proxy di poter re-indicizzare (i.e. condivisione dello stato della sessione fra backend e frontend proxies) e per facilitare la comparazione di stringhe Huffman-encoded.

6.6. Reset - change your mind

Uno degli svantaggi di HTTP 1.1 è che, una volta che un messaggio HTTP è stato inviato con un determinato Content-Length associato, non è possibile fermarlo. Certo, puoi spesso disconnetter la sessione TCP (non sempre), ma tale pratica implica il costo di una nuova negoziazione TCP (handshake).

Una soluzione più intelligente consisterebbe nel poter fermare il messaggio ed inviarne uno nuovo, correto. Ciò potrà essere realizzato con il nuovo frame http2 RST_STREAM che aiuterà a risparmiare banda ed evitare di interrompere la connessione a livello TCP.

6.7. Push da parte del server

Questa funzione è nota come “cache push”. In pratica, quando un client richiedesse una risorsa X, il server potrebbe voler proporre anche una risorsa Z e mandarla al client senza avenre ricevuto domanda. Ciò aiuta il client ad inserire la risorsa Z in cache, così da averla a disposizione in caso di bisogno.

Il server push deve essere cocesso esplicitamente per volontà del client. Anche a quel punto, il client può terminare istantaneamente la ricezione di uno stream in ogni momento, tramite il messaggio RST_STREAM in caso non dovesse desiderare una determinata risorza Z.

6.8. Controllo di Flusso

Ogni singolo stream http2 possiede e pubblicizza la propria "finestra di flusso" verso la quale l'altro estremo è autorizzato ad inviare dati. Se conoscete il funzionamento di SSH, questo modo di fare è in linea con lo stesso spirito e principio.

Per ogni singolo stream i due estremi devono potersi avvertire l'un l'altro quando ci dovesse essere abbastanza spazio per accogliere dati in entrata; allo stesso tempo un estremo è abilitato a spedire tanti dati quanti siano quelli ammessi dalla finestra di scambio. Solo i frame DATA sono sottoposti a flow-control (controllo di flusso).

7. Estensioni

Il protocollo http2 impone che il ricevente sia obbligato a leggere e ignorare tutti i frame sconosciuti (riportanti un tipo di frame sconosciuto). Due estremi possono negoziare l'uso di nuovi tipi di frame su base hop-by-hop ma tali frame non hanno il permesso di cambiare stato, e il loro flusso non beneficerà di flow-control.

Il fatto che http2 possa supportare una estensione è stato dibattuto a lungo durante lo sviluppo del protocollo con opinioni variabili, pro e contro. Dopo la draft-12 il pendolo ha ocillato un ultima volta in favore delle estensioni.

Le estensioni non fanno parte integrante ma sono e saranno documentate all'esterno del documento che specifica i fondamenti di protocollo. Ci sono gia due nuovi tipi di frame per i quali viene discussa l'inclusione ufficiale, i primi frame ad essere spediti come estensioni. Li descriverò vista la loro popolarità e il loro precedente stato di frame "nativi":

7.1. Servizi Alternativi

Con l'adozione di http2 abbiamo ragione di sospettare che le connessioni TCP siano ben più lunghe, durevoli, e che saranno mantenute ben più a lungo rispetto alla durata media delle anziane connessioni HTTP 1.x. Un client dovrebbe essere in grado di fare praticamente tutto all'interno di una sola connessione per ogni host/sito; tale connessione potrebbe potenzialmente permanere aperta a lungo.

Questo fattore influenzerà il funzionamento dei load-balancers HTTP fino ad arrivare ad una situazione in cui sarà lo stesso sito a consigliare al client di riconnettersi attraverso un altro host, per motivi di performance, maintenance, etc.

Il server manderà un header Alt-Svc: (o un frame ALTSVC in http2) istruendo il client a proposito del servizio alternativo: un'altra rotta verso lo stesso contenuto, rotta che però utilizza un servizio ed un numero di porta differenti.

Un client dovrebbe dunque provare a connettersi a tale servizio in maniera asincrona ed utilizzare tale alternativa solo in caso che la connessione abbia successo.

7.1.1. TLS opportunistico

L'header Alt-Svc permette ad un server di contenuti via http:// di informare il client che lo stesso contenuto è disponibile anche attraverso una connessione TLS.

Questa è in qualche modo una feature discutibile. Tale connessione utilizzerebbe TLS non-autenticato e non sarebbe ritenuta "sicura" comunque, non essendo predisposta per avvertire in alcun modo l'interfaccia utente (UI) della disponibilità di una connessione cifrata. Oltretutto non avremmo modo di distinguere ove l'utente stia utilizzando il caro e vecchio HTTP. In effetti molti esperti sono ancora fermamente contrari a tale opportunismo in ambito TLS.

7.2. Bloccato

Si suppone che un frame di questo tipo sia inviato esattamente una sola volta da parte di un peer http2 quando esso dovesse avere dati da spedire, ma il flow- control glielo avesse impedito. In pratica, se la tua implementazione riceve un frame di questo tipo, evidenzierebbe un problema a livelo di configurazione e/o transfer-rate non ottimale.

Una citazione dalla draft-12, prima che questo frame diventasse una estensione a parte:

> "Il frame BLOCKED è incluso in questa draft per facilitare la sperimentazione. Se i risultati non dovessero mostrare un vantaggio o miglioramento tale frame potrà essere rimosso"

8. Un mondo di http2

Quindi, come appariranno le cose quando http2 sarà adottato ? Sarà mai usato ?

8.1. Come http2 influenzerà l'utente tipo?

http2 non è ancora ne distribuito ne adodatto. Non possiamo ancora dire come le cose evolveranno. Abbiamo visto come SPDY sia stato messo in campo; possiamo solo fare qualche ipotesi basata su quella ed altre esperienze passate, ed sugli esperimenti in corso.

http2 riduce il numero di round-trip ed evita definitivamente l'eterno dilemma del "head-of-line blocking" attraverso l'utilizzo di multiplexing e fast-discarding degli stream non necessari.

Permette un vasto ammontare di streams paralleli, numero ampiamente sufficiente anche per il sito più "sharded" (parallelizzato) del momento.

Utilizzando correttamente le priorità sugli streams il client sarà in grado di scaricare i dati importanti prima dei dati quelli meno utili. Visto tutto ciò, sono sicuro che beneficieremo di siti più reattivi, che si caricano in tempi più rapidi. Mettiamola così: una miglior esperienza del web.

Quanto celere sarà questo miglioramento, staremo a vedere, non penso si possa ancora dire. Per prima cosa, la tecnologia è ancora in fase di lancio e non si sono ancora visti client/server che implementino a dovere questa tecnologia per sfruttarne tutte le potenzialità.

8.2. Come http2 influenzerà lo sviluppo web?

Nel corso degli anni, gli sviluppatori e gli IDE si sono riuniti a formare una immensa cassetta degli attrezzi piena di utensili e trucchi per circuire HTTP 1.1; come ricorderete, ne ho menzionati alcuni all'inizio di questo documento come giustificazioni per passare a http2.

Molti workaround oggi utilizzati da tool e sviluppatori in maniera automagica, avranno probabilmente effetti negativi sulle performance di http2, o perlomeno non aiuteranno a sfuttare i superpoteri di http2. Spriting e inlining non dovrebbero più essere utilizzati in http2. Anche lo sharding a suo modo dovra essere abbandonato in favore di un minor numero di singole connessioni.

Il problema risiede nel fatto che i siti web devono essere mantenuti ed evoluti di continuo, a breve termine; garantire performance adeguate per entrambi gli utilizzatori di HTTP 1.1 e http2 sarà una grande sfida per gli sviluppatori che vorranno evitare di offrire due frontend separati.

Anche solo per questi motivi, sospetto che passerà un pò di tempo prima che tutti possiamo apprezzare il pieno potenziale di http2, dappertutto.

8.3. Implementazioni di http2

Provare a documentare le implementazioni specifiche in un documento come questo sarebbe un impresa futile e prona al fallimento, tale documento diventerebbe desueto in breve tempo. Cercherò piuttosto di spiegare la situazione in termini più ampii, oltre ad indirizzare i lettori verso il sito di http2, list of implementations.

Un grande numero di implementazioni hanno preso vita fin dall'inizio ed altre si sono aggiunte durante il lavoro di definizione di http2. Al momento attuale, sono citate più di 40 implementazioni, la maggior parte delle quali rispetta la versione finale delle specifiche.

8.3.1 Navigatori

Firefox è il browser che più di altri ha seguito da vicino i rapidi sviluppi delle ultimissime drafts, Twitter è rimasta al passo offrendo i suoi servizi su http2. Google ha iniziato verso Aprile 2014 ad offrire supporto per http2 su un numero limitato di server di test e a partire da Maggio 2014 ha fornito supporto per http2 nelle versioni per sviluppatori Chrome. Microsoft da parte sua ha dato una dimostrazione del supporto http2 in Internet Explorer. Safari (su iOS 9 e Mac OS X El Capitan) e Opera hanno entrambi annunciato supporto per http2 in future versioni.

8.3.2 Servers

Esistono gia molte implementazioni di server http2.

Il popolare Nginx offre supporto per http2 gia dalla 1.9.5 rilasciata il 22 Settembre 2015 (dove rimpiazza il modulo SPDY, impedendo l'utilizzo contemporaneo di entrambi all'interno della stessa istanza.

Apache's httpd server has a http2 module mod_http2 since 2.4.17 which was released on October 9, 2015.

H2O, Apache Traffic Server, nghttp2, Caddy e LiteSpeed hanno tutti dimostrato di essere capaci di gestire richieste http2.

8.3.3 Altri

curl e libcurl supportano http2 non-sicuro (testuale) e la versione TLS servendosi di una delle svrariate librerie TLS disponibili.

Anche Wireshark supporta http2; strumento professionale per l'analisi del traffico di rete http2.

8.4. Critiche comuni a http2

Durante lo sviluppo di questo protocollo, il dibattito ha preso pieghe diverse, avanti e indietro rispetto a determinate posizioni intellettuali, è ovvio che a detta di alcune persone questo protocollo abbia fatto davvero una brutta fine. Vorrei citare alcune delle critiche più frequenti e rispondere tono su tono:

8.4.1. “Il protocollo è progettato o prodotto da Google”

Vi sono anche varianti in cui si afferma che il mondo diventa sempre più incline al controllo globale di Google. Ciò non è assolutamente vero. Il protocollo è stato sviluppato in seno alla IETF nello stesso modo in cui altri protocolli hanno visto luce durante gli ultimi 30 anni. Ciononostante, riconosciamo e prendiamo atto che l'incredibile lavoro di Google su SPDY non solo ha dimostrato come sia possible diffondere un nuovo protocollo in maniera "non-standard" ma ha anche fornito i numeri per valutarne futuri guadagni in prestazioni.

Google ha pubblicamente annunciato che rimuoverà il supporto per SPDY e NPN su Chrome nel 2016, incoraggiando la migrazione a http2.

8.4.2. “È un protocollo utile soltanto ai browser”

This is sort of true. One of the primary drivers behind the http2 development is the fixing of HTTP pipelining. If your use case originally didn't have any need for pipelining then chances are http2 won't do a lot of good for you. It certainly isn't the only improvement in the protocol but a big one.

As soon as services start realizing the full power and abilities the multiplexed streams over a single connection brings, I suspect we will see more application use of http2.

Small REST APIs and simpler programmatic uses of HTTP 1.x may not find the step to http2 to offer very big benefits. But also, there should be very few downsides with http2 for most users.

8.4.3. “Il protocollo è utile solo ai siti più grandi”

Not at all. The multiplexing capabilities will greatly help to improve the experience for high latency connections that smaller sites without wide geographical distributions often offer. Large sites are already very often faster and more distributed with shorter round-trip times to users.

8.4.4. “Il fatto che usi TLS lo rende ancora più lento”

This can be true to some extent. The TLS handshake does add a little extra, but there are existing and ongoing efforts on reducing the necessary round-trips even more for TLS. The overhead for doing TLS over the wire instead of plain-text is not insignificant and clearly notable so more CPU and power will be spent on the same traffic pattern as a non-secure protocol. How much and what impact it will have is a subject of opinions and measurements. See for example istlsfastyet.com for one source of info.

Telecom and other network operators, for example in the ATIS Open Web Alliance, say that they need unencrypted traffic to offer caching, compression and other techniques necessary to provide a fast web experience over satellite, in airplanes and similar. http2 does not make TLS use mandatory so we shouldn't conflate the terms.

Many Internet users have expressed a preference for TLS to be used more widely and we should help to protect users' privacy.

Experiments have also shown that by using TLS, there is a higher degree of success than when implementing new plain-text protocols over port 80 as there are just too many middle boxes out in the world that interfere with what they would think is HTTP 1.1 if it goes over port 80 and might look like HTTP at times.

Finally, thanks to http2's multiplexed streams over a single connection, normal browser use cases still could end up doing substantially fewer TLS handshakes and thus perform faster than HTTPS would when still using HTTP 1.1.

8.4.5. “Il fatto che non sia ASCII è un grosso svantaggio”

Si, amiamo poter vedere i protocolli cleartext in azione, rende più facile la diagnostica. Però tali protocolli sono più proni all'errore, oltre ad aprire le porte ad errori di parsing.

Se davvero non sopporti i protocolli binarii, allora non hai mai interagito con TLS e compressione in HTTP 1.x, pratica che peraltro è in atto da lunghissimo tempo.

8.4.6. “Non è più veloce di nessun HTTP/1.1”

Questo è ovviamente oggetto di dibattiti e discussioni animate riguardo la definizione di "velocità", benchè svariati test eseguiti al tempo di SPDY abbiano gia mostrato che il tempo di "page load" sia deavvero minore (per esempio "How Speedy is SPDY?" dell'Università di Washington e "Evaluating the Performance of SPDY-enabled Web Servers" di Hervé Servy) e chiaramente tali esperimenti sono stati ripetuti su http2.

Aspetto con ansia che altri risultati ed esperimenti simili vengano pubblicati. Un primo semplice test eseguito da httpwatch.com sembra indicare che HTTP/2 stia mantenendo le proprie promesse.

8.4.7. “È sviluppato attorno ad una statificazione di violazioni”

Davvero, questa la vostra critica ? Gli strati non sono sacri e intoccabili pilastri di una religione globale. Se ci siamo trovati a passare per zone d'ombra durante lo sviluppo di http2, è sempre stato nell'interesse e allo scopo di creare un protocollo efficace e stabile, a partire dai vinvoli iniziali.

8.4.8. “Non rimedia a svariati limiti intrinsechi a HTTP/1.1”

Vero. Con l'obiettivo preciso di mantenere i paradigmi HTTP/1.1 molte delle antiche features dovevano rimanere incluse, tali quali gli header più comuni, che a loro volta spesso includono fantomatici cookies, autorizzazioni e tanto altro. Il vantaggio nel mantenere questi paradigmi è che ora abbiamo un protocollo che distribuibile e integrabile senza bisogno di un eccessivo ammontare di aggiornamenti, evitando di riscrivere le parti fondamentali o di rimpiazzarle ex-novo. http2 è di fatto solo un nuovo strato di incapsulazione, di framing.

8.5. Sarà http2 massivamente adottato?

È troppo presto per dirlo con certezza, ma posso indovinare e stimare che è quello che succederà.

I detrattori diranno “guarda quanto bene ha fatto IPv6” portandolo come esempio di un nuovo protocollo per il quale ci sono voluti decenni prima che si iniziasse a vederlo implementato massivamente. Ebbene, http2 non è un nuovo IPv6. È un protocollo basato su TCP che fa leva sullo straodinario meccanismo di Update HTTP, sui numeri di porta, su TLS, etc. Non necessiterà di alcuna modifica a router o firewall.

Con il suo SPDY, Google ha dimostrato al mondo come un nuovo protocollo possa essere sviluppato e distribuito a browser e webservices attraverso implementazioni multiple, in un tempo tutto sommato breve. Mentre l'ammontare di server che offrono SPDY su Internet è nel range dell'1%, l'ammontare di dati con i quali questi server hanno a che fare è ben maggiore. Alcuni dei più popolari e visitati siti propongono SPDY gia ad oggi.

Direi che http2, basato sullo stesso principio e paradigma di SPDY, si diffonderà ancora di più per via del fatto che tale protocollo proviene dalla IETF. Il deploy di SPDY ha sempre sofferto dello stigma, essendo esso “un protocollo di Google”.

Vi sono svariati "grandi browser" dietro la distribuzione di http2. Rappresentati di Firefox, Chrome, Safari, Internet Explorer e Opera hanno espresso la volontà di includere http2 nei proprii broswer; essi hanni dimostrato implementazioni funzionanti.

I più grandi portali e operatori del web -fra cui Google, Twitter e Facebook- sono interessati ad offrire http2 quanto prima. Speriamo di vedere questo supporto arrivare anche alle piattaforme server tipo Apache e nginx. Un nuovo velocissimo server HTTP con supporto http2 che mostra un enorme potenziale è H2o.

Molte delle più famose case produttrici di proxy hanno annunciato di voler supportare http2, fra cui HAProxy, Squid e Varnish.

Durante il corso del 2015, il traffico http2 è aumentato. A inizio Settembre il tasso di utilizzazione di Firefox 40 era al 13% su tutto il traffico HTTP e 27% su tutto il traffico HTTPS, mentre verso Google circa il 18% di richieste entranti sono in HTTP/2. Bisogna notare che allo stesso tempo Google è in procinto di sperimentare altri protocolli (vedi QUIC in 12.1) il che abbassa in qualche modo il tasso di connessioni http2 globali.

9. http2 in Firefox

Firefox ha seguito l'evolversi delle draft da vicino, ha reso disponibili test sulle implementazioni http2 per lunghi mesi. Durante lo sviluppo del protocollo http2, client e server hanno dovuto mettersi d'accordo su quale versione della draft usare, cosa perarltro abbastanza fastidiosa in fase di test. Giusto per essere sicuri che client e server siano allineati su quale versione della draft stiano implementando.

9.1. Primo passo, essere sicuro di averlo abilitato

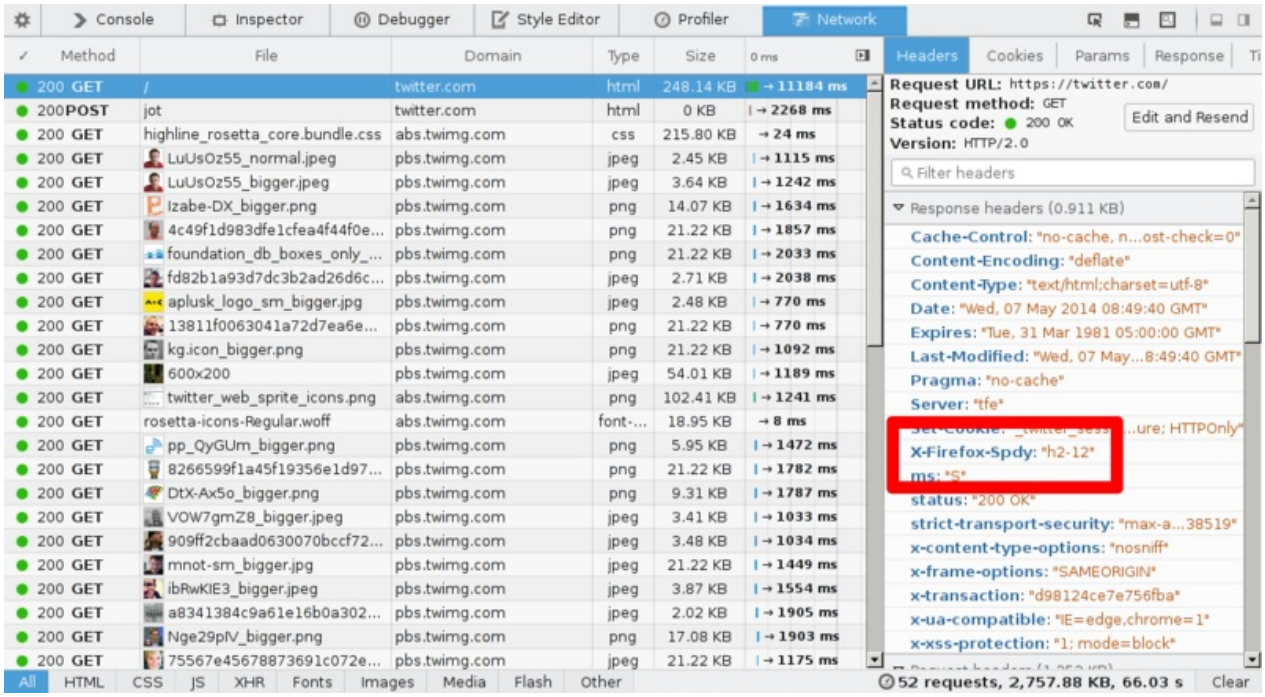
In tutte le versioni di Firefox a partire dalla 35, rilasciata il 13 Gennaio 2015, il supporto nativo http2 è abilitato.

Digita 'about:config' nella barra dell'indirizzo e cerca l'opzione “network.http.spdy.enabled.http2draft”. Assicurati che sia impostata a *true*. In Firefox 36 esiste un'altra voce di configurazione chiamata “network.http.spdy.enabled.http2” che è impostata su *true* per default. Quest'ultima controlla la versione “plain” di http2 mentre la prima abilita o disabilita la negoziazione della versione http2-draft. Entrambe sono abilitate a partire da Fierfox 36.

9.2. Solo TLS

Ricordatevi che Firefox implementa https solo su TLS. Vedrete http2 in azione se e solo se starete utilizzando un sito https:// che offre supporto per http2.

9.3. Trasparente!



Non esiste alcun elemento della UI (User Interface) che indichi quando si stia parlando http2. Non è facile poterlo affermare. Un modo relativamente facile di verificarlo è abilitare la parte “Web developer->Network” e controllare negli header di risposta quanto ottenuto dal server. In caso risposta sia dunque “HTTP/2.0” qualcosa, Firefox inserirà il proprio header “X-Firefox-Spdy:” come possbile osservare nello screenshotqui sopra.

Gli headers che vedete nel tool Network durante un dialogo http2 sono stati convertiti a partire dal formato http2 binario verso il vecchio stile di headers HTTP 1.x

9.4. Visualizzare l'utilizzo di http2

Esistono plugins Firefox che aiutano a capire se un sito utilizzi http2. Uno di questi è “HTTP/2 and SPDY Indicator”.

10. http2 in Chromium

Il team di Chromium ha implementato http2 e ha anche fornito supporto sui canali dev e beta per lungo tempo. A partire da Chrome 40, rilasciato il 27 Gennaio 2015, http2 è abilitato per default per un determinato numero di utenti. Hanno iniziato supportando una utenza ristretta per poi aumentare gradualmente nel tempo.

Il supporto nativo per SPDY verrà eventualmente eliminato. Il progetto ha comunicato tale notizia in un post February 2015:

> “Chrome ha supportato SPDY a partire dalla versione 6, ma dato che la maggior parte dei benefici sono presenti anche in HTTP/2, è tempo di dirsi addio. Pianifichiamo di rimuovere il supporto per SPDY ad inizio 2016”

10.1. Per prima cosa, accertarsi di averlo abilitato

Digitare “chrome://flags/#enable-spdy4" nella barra indirizzi e cliccare su “enable” se non fosse gia abilitato.

10.2. Solo TLS

Ricordate che Chrome implementa http2 solo attraverso TLS. Vedrete http2 in azione solamente quando Chrome verrà utilizzato su un sito https:// che offra supporto nativo http2.

10.3. Visualizzare l'impiego di HTTP/2

Sono disponibili plugin per Chrome che aiutano a visualizzare se un sito stia utilizzando HTTP/2. Uno di questi è “HTTP/2 and SPDY Indicator”.

10.4. QUIC

Gli attuali esperimenti di Chrome e l'impiego di QUIC (vedi sezione 12.1) fanno sì che il numero di connessioni HTTP/2 sia in diminuzione.

11. http2 in curl

Il curl project ha fornito supporto sperimentale ad http2 a partire da Settembre 2013.

Nello spirito di curl, abbiamo intenzione di supportare praticamente ogni singolo aspetto di http2. curl è spesso usato come tool di test per interagire in maniera flessibile con i siti web, dunque vogliamo mantenere lo stesso per http2.

curl utilizza la libreria separata nghttp2 per poter offrire funzionalità a livello di frame. curl necessita di nghttp2 1.0 o superiore.

Notate che al momento su linux, curl e libcurl non sono sempre distribuiti con il supporto HTTP/2 abilitato.

11.1. Sembra HTTP 1.x

Al suo interno, curl converte gli headers entranti http2 in headers stile HTTP 1.x presentandoli all'utente e facendoli apparire molto simili ai pre-esistenti HTTP. Ciò permette una transizione facilitata per tutti gli strumenti che usano curl e HTTP oggi. In modo simile curl convertirà gli header uscenti con lo stesso stile. Passateli a curl in stile HTTP 1.x e lui si occuperà di convertirli al volo durante il dialogo con server http2. Questo permette agli utenti di non doversi occupare troppo di quale particolare versione di headers HTTP si stia usando.

11.2. Plain text, non sicuro

curl supporta http2 su TCP standard attraverso l'header "Upgrade:". Se si esegue una richesta HTTP e si richiede HTTP 2, curl chiederà al server di aggiornare la connessione a http2 ove possibile.

11.3. TLS, quali librerie

curl supporta un vasto numero di librerie TLS per il proprio back-end TLS, ed è ancora il caso con http2. La difficoltà per http2 utilizzando TLS è offirre buon supporto ALPN e talvolta NPN.

Lancia una build di curl con moderne version di OpenSSL o NSS per assicurare il support di ALPN e NPN. Se utilizzi GnuTLS o PolarSSL, avrai ALPN ma non NPN.

11.4. Utilizzo in linea di comando

Per istruire curl ad utilizzare http2 -via plain-text o su TLS- utilizzare la opzione `--http2` (meno meno http2). curl è ancora impostato per utilizzare HTTP/1.1 per default, quindi l'opzione è necessaria se desideriamo http2.

11.5. Opzioni di libcurl

11.5.1 Abilitare HTTP/2

La tua applicazione continuerà ad utilizzare URL di tipo https:// o http:// ma dovrai anche settare la voce curl_easy_setopt `CURLOPT_HTTP_VERSION` a `CURL_HTTP_VERSION_2` per far sì che libcurl provi ad utilizzare http2. Su base best-effort proverà ad utilizzare http2 altrimenti continuerà su HTTP 1.1.

11.5.2 Multiplexing

Dato che libcurl prova a mantenere gli stessi comportamenti di sempre, dovrai abilitare il multiplexing HTTP/2 nella tua applicazione tramite l'opzione CURLMOPT_PIPELINING. In caso contrario, continuerai ad utilizzare una richiesta per volta per ogni connessione disponibile.

Altro piccolo dettaglio da tenere a mente quando si richiedono trasferimenti multipli via libcurl tramite la sua interfacia "multi", una applicazione potrebbe decidere di iniziare un numero infinito di trasferimenti simultanei; se desideriamo veicolarli tutti tramite la stessa connessione piuttosto che utilizzarne una moltitudine, possiamo istruire libcurl affinchè aspetti un determinato lasso di tempo utilizzando l'opzione CURLOPT_PIPEWAIT.

11.5.3 Server push

libcurl 7.44.0 e successivi supportano la server push HTTP/2. Potrete trarne vantaggio impostando un callback tramite l'opzione CURLMOPT_PUSHFUNCTION Se l'applicazione accettase il push, utilizzerebbe un handler di tipo CURL easy per trasmettere il contenuto del trasferimento, così come avverrebbe in ogni altro caso.

12. Dopo http2

A lot of tough decisions and compromises have been made for http2. With http2 getting deployed there is an established way to upgrade into other protocol versions that work which lays the foundation for doing more protocol revisions ahead. It also brings a notion and an infrastructure that can handle multiple different versions in parallel. Maybe we don't need to phase out the old entirely when we introduce new?

http2 ha a che fare con un sacco di eredità di HTTP 1, per via del fatto che si è voluto poter lasciare aperta la possibilità di proxificare avanti e indietro il traffico HTTP 1 e http2. Parte di questa eredità pone un limite a futuri sviluppi e invenzioni. Magari http3 potrà disfarsi di questa eredità ?

Cosa pensate che possa ancora mancare a http?

12.1. QUIC

Il protocollo di Google QUIC (Quick UDP Internet Connections) è un esperimento interessante che si è svolto in linea con lo stesso spirito e stile di SPDY. QUIC è un rimpiazzo per TCP + TLS + HTTP/2 implementato su UDP.

QUIC permette la creazione di connessioni con minor latenza, risolve la perdita di pacchetti bloccando individualmente uno stream piuttosto che tutti insieme come avviene in HTTP/2, olre a rendere possibile un muliplex semplificato su NIC diversi - coprendo quindi aree relative a ciò che MPTCP dovrebbe risolvere.

QUIC è per il momento implementato solo dai server di Google e dal loro browser Chrome; tale codice non è facilmente riutilizzabile altrove, anche se esiste una libreria che cerca di provvedere alla lacuna libquic. Il protocollo è stato presentato al gruppo di lavoro "transport" dela IETF nella draft draft.

13. Altre letture

Se pensate che questo documento sia ancora troppo poco tecnico, qui troverete altre risorse per soddisfare la vostra curiosità:

- La mailing list e gli archivi di HTTPbis: https://lists.w3.org/Archives/Public/ietf-http-wg/
- La specifica http2 odierna in una versione HTMLified: https://httpwg.github.io/specs/rfc7540.html
- Dettagli sul networking http2 di Firefox http2: https://wiki.mozilla.org/Networking/http2
- Dettagli sulla implementazione di http2 in curl: https://curl.haxx.se/docs/http2.html
- Il sito di http2: https://http2.github.io/ e più in particolare la FAQ: https://http2.github.io/faq/
- Il capitolo di Ilya Grigorik's su HTTP/2 nel suo libro “High Performance Browser Networking”: https://hpbn.co/http2/

14. Riconoscimenti, Ringraziamenti

Mark Nottingham per l'ispirazione e l'immagine formato Lego dei pacchetti.

I dati sui trend HTTP provenienti da https://httparchive.org/.

I grafici RTT provengono dalle presentazioni di Mike Belshe.

I miei figli Agnes e Rex per avermi prestato i loro personaggi Lego per l'immagine "inizio della fila".

Grazie agli amici per riletture e ritorni: Kjell Ericson, Bjorn Reese, Linus Swälas e Anthony Bryan. Il vostro aiuto è immensamente apprezzato e ha davvero migliorato questo documento!

Nel corso delle varie iterazioni, le persone seguenti hanno amicalmente inviato bug-report e migliorie a questo documento: Mikael Olsson, Remi Gacogne, Benjamin Kircher, saivlis, florin-andrei-tp, Brett Anthoine, Nick Parlante, Matthew King, Nicolas Peels, Jon Forrest, sbrickey, Marcin Olak, Gary Rowe, Ben Frain, Mats Linander, Raul Siles, Alex Lee, Richard Moore