# HTTP/3 Explained

by Daniel Stenberg

# Sommario

# HTTP/3 spiegato

Lo slancio per scrivere questo libro è partito in Marzo 2018. Il piano consiste nel documentare HTTP/3 e il protocollo sottostante, ossia QUIC. Perchè sono stati concepiti, come funzionano, dettagli sul protocollo, implementazioni, etc.

Questo libro è inteso per essere libero, gratuito; consta dello sforzo condiviso che include chiunque abbia voglia di contribuire.

## Prerequisiti

Al lettore di questo libro è richiesto di avere una comprensione basilare di Reti TCP/IP, di HTTP e del mondo web. Per maggiori dettagli e specifiche a proposito di HTTP/2 vi raccomandiamo di consultare preventivamente le informazioni contenute su http2 explained.

## L'autore

Il presente libro è stato ideato e scritto da Daniel Stenberg . Daniel è fondatore e sviluppatore principale di curl, il client HTTP più usato al mondo. Daniel ha lavorato con e per HTTP ed altri protocolli internet per più di due decenni ed è l'autore di http2 explained.

## Homepage

La homepage del libro si trova su daniel.haxx.se/http3-explained .

## Contribuire

Se dovessi trovare errori, omissioni o falsità in questo documento, perfavore inviaci una versione aggiornata del paragrafo e ci assicureremo che la modifica venga pubblicata. Attribuiremo il credito a chiunque si renda utile. Spero di continuare a migliorare questo libro nel corso del tempo.

Preferibilmente, segnalare errori o richieste pull sulla pagina github del progetto.

## Licenze

# Perchè QUIC

QUIC è un nome, non un acronimo. Si pronuncia esattamente come la parola Inglese "quick" (veloce).

QUIC può essere visto come un nuovo protocollo sicuro in grado di stablire un trasporto adatto a trasportare semantiche simil-HTTP e che possa risolvere alcune delle limitazioni di HTTP/2 su TCP e TLS. Rappresenta un passo avanti nell'evoluzione del trasporto web.

QUIC non si limita al solo trasporto HTTP. Il desiderio di rendere il web e la distribuzione dei contenuti sempre più veloce agli occhi degli utenti è probabilmente la ragione principale ad aver stimolato la creazione di questo nuovo protocollo di trasporto.

Perchè creare un nuovo protocollo di trasporto e perchè mai basarlo su UDP ?

# Ricordate HTTP/2 ?

La specifica HTTP/2 RFC 7540 è stata pubblicata nel Maggio 2015 e da allora il protocollo è stato implementato e largamente distribuito attraverso l'intero Internet e il world wide web.

Ad inizio 2018, quasi il 40% dei top-1000 siti web funziona in HTTP/2: circa il 70% di tutte le richieste HTTPS inviate da Firefox ha ricevuto risposte di tipo HTTP/2. Tutti i principali server, proxy e browser supportano h2.

HTTP/2 cerca di risolvere svariati limiti intrinsechi ad HTTP/1; con l'introduzione di tale seconda versione di HTTP gli utenti non sono più obbligati ad utilizzare work-around e gabole complicate. Tali sotterfugi caricano inutilmente gli sviluppatori di responsabilità non previste.

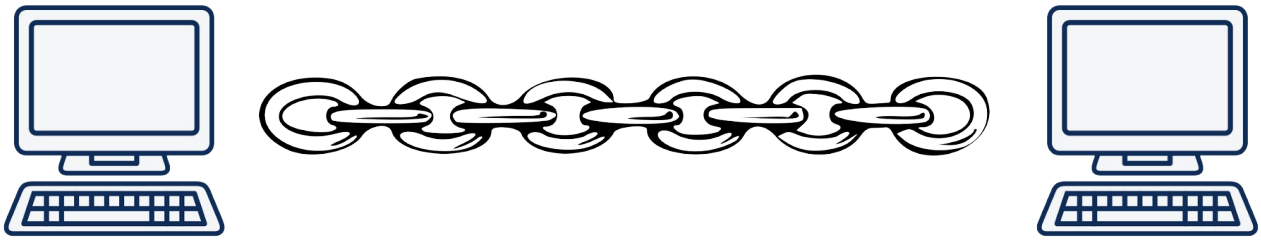
Una delle funzioni principali in HTTP/2 è il multiplexing, tecnica che permette di inviare molteplici flussi logici (indipendenti) all'interno di una stessa connessione TCP. Ciò rendre il tutto più rapido e fluido. Fa si che il controllo di flusso sia applicato in maniera più efficace, permette agli utenti di sfruttare il TCP appieno, di saturare la banda, di rendere le connessioni TCP più durature -il che permette nella maggior parte dei casi di raggiungere la massima velocità. La compressione degli header permette di risparmiare banda.

Con HTTP/2, i browser usano tipicamente *una* connessione TCP per ogni host rispetto alle precedenti *sei*. In effetti, tecniche come il "desharding" e la condensazione ("coalescing") delle connessioni HTTP/2 potrebbero ridurre ulteriormente il numero di connessioni totali.

HTTP/2 risolve il problema del "bloccaggio di inizio fila": il client era costretto ad aspettare la fine dell'elaborazione della richiesta in corso prima di poter prendere in carico la risposta alla domanda in attesa. Ora non più.

# Bloccaggio ad inizio della coda TCP

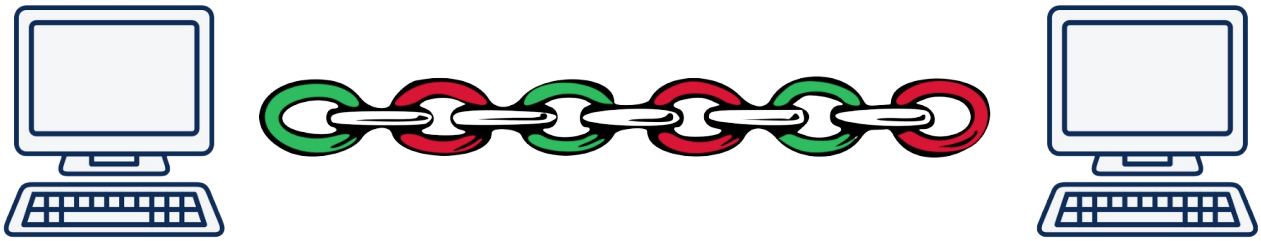
HTTP/2 è comunque ancora basato su TCP seppur utilizzi un minor numero di connessioni rispetto ai predecessori. Il TCP è un protocollo di trasferimento affidabile; lo si può immaginare come una corda immaginaria tesa fra due host. Ciò che viene appeso ad un estremo arriverà all'altro, nello stesso ordine - a costo dell'interruzione della connessione.



Con HTTP/2, il browser avrà la tendenza a soddisfare decine o anche centinaia di trasferimenti attraverso la stessa connessione TCP.

Se un singolo pacchetto venisse perso, abbandonato o scartato da qualche parte sulla rete, fra due endopoints che stessero dialogando in HTTP/2, tale dialogo verrebbe sospeso fino alla ri-trasmissione e all'arrivo di tale pacchetto a destinazione. Osservando la catena con cui è raffigurato il TCP, capiamo come -in caso di interruzione del link- tutti i dati che seguono il pacchetto in sospeso dovranno attendere.

Una illustrazione della metafora della catena nel momento in cui due flussi siano trasmessi sulla medesima connessione. Un flusso in rosso, uno in giallo:



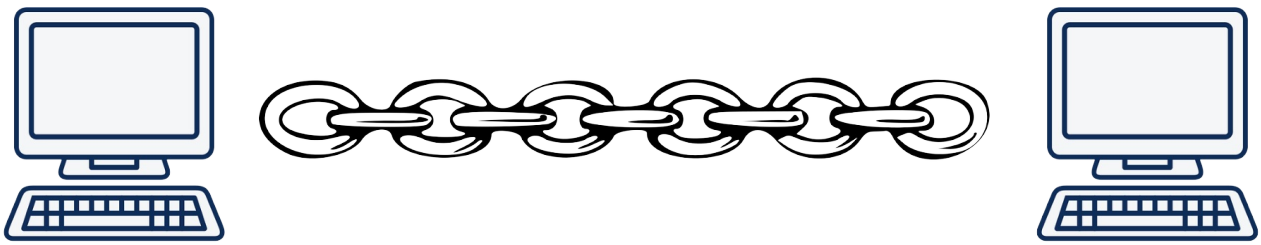
Si parla quindi di bloccaggio ad inizio della coda, basato su TCP!

Se il tasso di perdita di pacchetti dovesse incrementare, HTTP/2 subirebbe un degrado di prestazioni. A 2% di perdita di pacchetti (che comunque rappresenta una qualità infima) svariati test hanno dimostrato come gli utenti di HTTP/1 ottengano risultati di gran lunga migliori, stimando che un tasso di perdita del 2% distribuito su sei connessioni abbia meno influenza che su una sola, permettendo così alle rimanenti connessioni di proseguire senza intralcio.

Porre rimedio a tale empasse non sarà semplice -se non impossibile- finchè si continuerà ad usare il TCP..

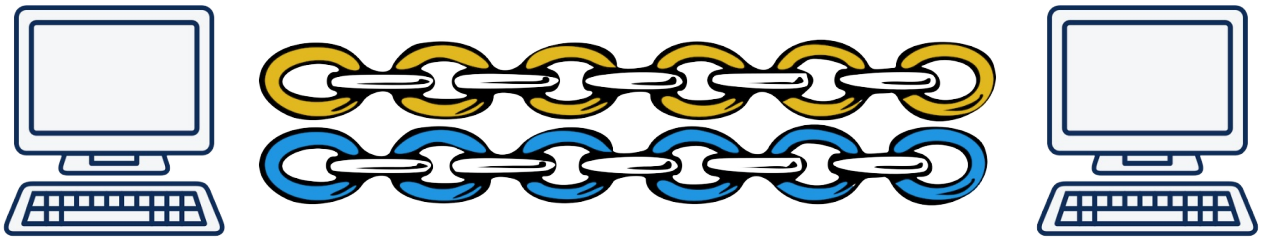
# L'impiego di flussi indipendenti evita il blocco

QUIC è composto da una fase di setup della connessione fra i due estremi, fase che rende la connessione sicura e la consegna dei dati affidabile.



Seppur instradati su una stessa connessione, i due flussi sono trattati in maniera indipendente tanto che se uno dei due collegamenti fosse interrotto, solo il flusso in questione sarà obbligato ad attendere la ri-trasmissione della risorsa perduta, senza alcun impatto sull'altro flusso parallelo.

Qui di seguito illustrati i due flussi (streams) scambiati dai due end-ponts, uno in blu ed uno in giallo.



# TCP o UDP

Dato che non possiamo completamente mettere fine al bloccaggio ad inizio fila all'interno di TCP, dovremmo almeno essere in grado di sviluppare un protocollo di trasporto più vicino a UDP e TCP nello stack di rete. O magari impiegare SCTP che è un protocollo di trasporto standardizzato dalla IETF nella RFC 4960 , contenente la maggior parte delle caratteristiche o funzioni necessarie allo scopo.

Al contrario, nel corso degli ultimi anni, tutti gli sforzi per progettare un nuovo protocollo di trasporto sono stati sospesi a causa delle difficoltà percepite nel distribuire tali protocolli all'interno di infrastrutture reali. La quantità di firewalls, NATs, routers e altri dispositivi di rete che si iterpongono fra l'utente e il server sono semplicemente troppi, e sono a conoscenza dei soli UDP o TCP. Introdurre un altro protocollo di trasporto fa si che un determinato tasso di connessioni N% fallisca, in quanto bloccato dalle suddette middle-boxes che identificherebbero tale proto come malevolo, malformato, nè TCP nè UDP. Un tasso di fallimento N% in ambito internet rappresenta spesso una buona ragione per non intraprendere alcuno sforzo.

In aggiunta, cambiare le cose a livello di "transport protocol layer" nello stack di rete può spesso significare dover mettere mano al kernel del SO. Modificare il kernel è un processo che richiede tempi lunghi, grandi sforzi e non al riparo da pressioni. In anni recenti, abbiamo visto features di TCP diventare standard IETF senza che vi sia stata corrispondenza -ad anni di distanza- con il tasso di distribuzione degli stessi, per via della mancanza di utilizzo o di supporto universale.

# Perchè no, SCTP-over-UDP

SCTP è un protocollo di trasporto di flussi affidabile, ed esistono persino alcune implementazioni in ambito WebRTC che utilizzano SCTP via UDP.

Non fu dunque scelto come valida base per QUIC per via di molteplici ragioni, ossia:

- SCTP non risolve il problema del "blocco di inizio riga" con i flussi
- SCTP richiede una decisione sul numero di streams nella fase di inizializzazione
- SCTP non ha un passato particolarmente brillante in quanto a sicurezza TLS
- SCTP utilizza una negoziazione a 4 vie, mentre QUIC ne offre una da 0-RTT
- QUIC è un flusso di bytes ordinato al pari di TCP, SCTP un protocollo a messaggi
- Una connessione QUIC può essere migrata fra indirizzi IP, ma SCTP non lo permette

Per maggiori dettagli relativi alle differenze, vedere Un raffronto tra SCTP e QUIC .

# Ossificazione

Internet è una rete fatta di reti. In diversi punti del mondo -sulla rotta fra le varie reti interconnesse- sono installati dispositivi che assicurano il buon instradamento e funzionamento della rete. Questi dispositivi -i varii componenti distribuiti della rete- sono comunemente chiamati "middle-boxes". Scatole che semplicemente si trovano in mezzo al percorso fra i due end-points e che a loro volta sono coinvolte nel trasferimento dati via rete.

Questi apparecchi servono molteplici funzioni ma limitiamoci semplicemente a concludere che se qualcuno li ha installati in tali specifici punti della rete, avrà sicuramente avuto i suoi buoni motivi.

Tali "middle-boxes" instradano i pacchetti IP fra le differenti reti globali, bloccano il traffico maligno, si occupano di tradurre gli indirizzi di rete (NAT - Network Address Translation), aumentano le prestazioni, tentano in certi casi di spiare il traffico passante, ed altro ancora.

Al fine di espletare i propri compiti, questi dispositivi devono avere conoscenza di "networking" e dei protocolli oggetto del traffico stesso. Appositi software sono sviluppati allo scopo, software che tuttavia non ricevono aggiornamenti troppo frequenti.

A parte essere ovviamente i "collanti" che tengono insieme Internet, questi dispositivi al cuore della rete non sono mai altrettanto avanzati in termini di tecnologia quanto quelli agli estremi della rete, ossia clients e servers.

Tutti protocolli di rete che queste macchine potrebbero voler ispezionare e di cui potrebbero voler decidere la sorte in base a origine o contenuto, subiscono questo problema: tali macchine sono state installate in conformità con le opzioni di protocollo disponibili all'epoca e non sono flessibili. Aggiunte o modifiche del comportamento non note in precedenza potrebbero fare si che il dispositivo interpreti in maniera non corretta il pacchetto, scartandolo per malformazione o contenuto ritenuto erroneamente illecito. Tale traffico subirebbe rallentamenti o drop improvvisi al punto da rendere tali nuove opzioni di protocollo indesiderabili da un punto di vista utente.

Siamo di fronte a ciò che possiamo definire "ossificazione del protocollo".

Anche le modifiche al TCP soffrono di ossificazione: alcune delle nuove opzioni TCP sono interpretate e bloccate in quanto sconosciute ai più. Se viene permesso al dispositivo di ispezionare in dettaglio il protocollo, il sistema tederà a definire pattern standard di comportamento per un determinato protocollo e nel tempo diventerà difficile se non appunto impossibile modificare tali assunzioni.

La sola vera ed efficace strategia per il contrasto della "ossificazione" è utilizzare in maniera estensiva la cifratura, cosi da impedire ai box intermedi di desumere (distinguere) quale protocollo stia passando sotto il loro ponte.

# Sicuro

QUIC è sempre sicuro. Non esiste alcuna versione clear-text del protocollo, al punto in cui l'aver negoziato QUIC implica l'impiego di crittografia e sicurezza TLS 1.3. Come detto in precedenza, tale pratica impedisce l'ossificazione ed ogni altra sorta di blocco o trattamento speciale, oltre ad assicurare che QUIC rispetti tutte le proprietà di sicurezza HTTPS che gli utilizzatori web oggigiorno si attendono e desiderano.

Solo una parte dei primissimi pacchetti della negoziazione iniziale sono trasmessi in chiaro, prima che il protocollo di cifratura venga negoziato.

# Dati in anticipo

QUIC offre entrambe le negoziazioni 0-RTT e 1-RTT al fine di minimizzare il tempo necessario alla negoziazione e setup della connessione. Paragoniamolo all'handshake a 3 vie del TCP.

In aggiunta a ciò QUIC mette a disposizione il supporto per "dati anticipati" sin da subito, per permettere un flusso di dati più intenso, oltre a risultare di semplice impiego rispetto all'opzione "TCP Fast Open"

Con il concetto di stream, un'altra connessione logica verso uno stesso host può essere effettuata direttamente senza aspettare che la prima abbia finito.

# Il "TCP Fast Open" è problematico

TCP FO è stato pubblicato come RFC 7413 nel Dicembre 2014; tale specifica descrive come le applicazioni possano inviare dati al server gia a partire dal primo pacchetto TCP SYN.

Arrivare ad un supporto mondiale per tale opzione ha necessitato lungo tempo, e siamo ancora vittime di malfunzionamenti nel 2018. Coloro i quali hanno implementato lo stack TCP hanno evidenziato svariati problemi, cosi come le applicazioni che tentavano di trare vantaggio da tale caratteristica - sia nel comprendere su quali versioni del Sistema Operativo attivarla sia nel tentare di risolvere problemi inerenti la mancanza di supporto lato client (backdown). Molte reti sono state segnalate per l'interferenza al traffico TFO, ed hanno quindi inficiato il buon funzionamento della negoziazione TCP.

# Evoluzione

La versione iniziale di QUIC è stata proposta da Jim Roskind di Google, fu implementata nel 2012 e successivamente annunciata al pubblico nel 2013, momento in cui la sperimentazione di Google si espanse fortemente.

A quei tempi l'acronimo QUIC veniva espanso in "Quick UDP Internet Connections" ma fu successivamente abbandonato.

Google ha gestito l'implementazione del protocollo e il successivo deploy su larga scala, tramite il famoso browser Chrome e su tutti i servizi web più famosi, fra cui search, gmail e youtube. Google è stata responsabile per lo sviluppo di numerose versioni del protocollo ed ha dimostrato la fattibilità del progetto utilizzando una vasta porzione di utenti Internet.

Nel Giugno 2015, la prima bozza internet per QUIC fu inviata ad IETF per la standardizzazione ma fu necessario attendere il tardo 2016 prima di osservare la formazione del gruppo di lavoro. Moltissimi vendor e attori del settore IT hanno presto contribuito ad accrescere l'interesse per il nuovo standard.

Nel 2017 i numeri mostrati dagli ingegnieri di QUIC a Google evidenziano come ben il 7% di *tutto* il traffico Internet transiti gia su questo protocollo. La versione "Google" del protocollo.

# IETF

Il gruppo di lavoro QUIC, creato per standardizzare il protocollo all'interno di IETF, decise che QUIC avrebbe dovuto essere in grado di veicolare altri protocolli oltre il "semplice" HTTP. Il QUIC-Google si era finora occupato solo di HTTP, più specificamente di frames HTTP/2, servendosi della sintassi gia disponibile per i frames HTTP/2.

Fu inoltre deciso che il QUIC-IETF dovesse basare il proprio framework di sicurezza e cifratura sullo standard TLS 1.3 al posto delle modifiche "fatte in casa" dal team Google.

Per soddisfare la volontà di veicolare "non solamente HTTP", l'architettura QUIC di IETF fu separata in due livelli: il trasporto QUIC e la parte "HTTP over QUIC" (riferendosi al suddetto livello utilizziamo anche il termine "hq").

Questa ultima separazione -innoqua quanto possa sembrare- ha creato sostanziose differenze fra le versioni originali Google e la versione finale IETF.

Il gruppo di lavoro ha comunque deciso abbastanza presto di concentrarsi sulla consegna -entro i termini preposti- della versione 1 di QUIC, ed ha quindi prediletto lo sviluppo di HTTP, rinviando così lo sviluppo del trasporto non-HTTP ad una seconda fase.

Quando abbiamo iniziato a lavorare allo sviluppo di questo libro nel Marzo 2018 l'idea era di consegnare le specifice di QUIC versione 1 verso Novembre 2018, data finalmente riportata a Luglio 2019.

Mentre il lavoro della IETF è avanzato, il team Google ha integrato i dettagli della nuova versione IETF ed ha iniziato ad avanzare progresivamente nella direzione di ciò che potrebbe finalmente diventare lo standard IETF. Google ha quindi continuato ad usare la propria versione di QUIC sia lato browser sia lato server applicativi.

Le più recenti implementazioni in via di sviluppo hanno deciso di focalizzare gli sforzi in direzione della versione IETF ufficiale; non sono perciò compatibili con le versioni proposte da Google.

# Esperienza da HTTP/2

La specifica HTTP/2 contenuta nella RFC 7540 è stata pubblicata nel Maggio 2015, un mese prima che QUIC fosse presentato ad IETF per la prima volta.

Con HTTP/2, le fondamenta per modificare HTTP furono gettate; il gruppo di lavoro responsabile della creazione di HTTP/2 era gia dell'idea che tale processo di rinnovamento avrebbe aiutato a rilasciare versioni HTTP più rapidamente che in passato, dove il passaggio da v1 a v2 richiese 16 anni.

Nel contesto di HTTP/2 gli utenti ed i software realizzano come non sia più più d'attualita lavorare con un HTTP testuale, in maniera serializzata.

HTTP-over-QUIC fu rinominato HTTP/3 in Novembre 2018.

# Status

Il gruppo di lavoro QUIC ha lavorato alacremente a partire dal tardo 2016 ad una specifica di protocollo e la sua volontà è di completare tale sforzo entro Luglio 2019.

A Novembre 2018, ancora non abbiamo notizia di test di interoperabilità estesi su HTTP/3 - solo fra le due implementazioni esistenti che tuttavia non sono guidate da un browser ne da un software opensource lato server.

Esistono all'incirca quindici Implementazioni di QUIC nella lista wiki del gruppo di lavoro, ma ovviamente non tutte possono operare allo stesso livello dell'ultima bozza delle specifiche.

Implementare QUIC non è affatto semplice, dato che è in continua evoluzione anche quando sembra stabilizzatosi.

# Servers

Il supporto per QUIC e HTTP/3 in NGINX è in fase di sviluppo. Il rilascio delle nuove funzionalità è previsto durante il ciclo di sviluppo di NGINX 1.17.

Non vi è alcun comunicato pubblico rispetto al supporto di QUIC in Apache.

# Clients

Nessun produttore di web-browser ha rilasciato una versione che sia in grado di utilizzare la versione IETF di QUIC o HTTP/3.

Google Chrome contiene la sua propria implementazione di QUIC in salsa Google da svariati anni; tale versione Google non è compatibile con le bozze IETF e l'implementazione di HTTP differisce dalla versione HTTP/3 ufficiale.

Mozilla sta sviluppando Neqo - un'implementazione di QUIC e HTTP/3 scritta in Rust. Neqo sarà integrato in Necko (una libreria di rete usata in molte applicazioni di Mozilla - incluso Firefox).

curl ha rilasciato un primo supporto sperimentale per HTTP/3 (draft-22) con la release 7.66.0 dell' 11 Settembre 2019. Per l'implementazione di HTTP/3, curl può usare sia la libreria Quiche di Cloudflare che la famiglia di librerie ngtcp2.

# Ostacoli all'implementazione

Per non dover re-inventare la ruota e poter contare su standard affermati, QUIC ha deciso di utilizzare TLS 1.3 come fondazione per gli strati di sicurezza e di crittografia. Tuttavia, nel fare ciò, il gruppo di lavoro ha infine riadattato l'utilizzo del protocollo TLS con QUIC, utilizzando solamente "messaggi TLS" e mai "records TLS".

Quello che potrebbe sembrare un'innocente modifica, in realtà ha causato non pochi grattacapi a chi era in procinto di implementare uno stack QUIC. Le librerie TLS esistenti con supporto a TLS 1.3 non dispongono di un numero sufficiente di API per esporre tale funzionalità e permettere quindi a QUIC di accedervi. Se è vero che molti degli sviluppatori/implementatori di QUIC appartengono a grandi organizzazioni che in parallelo sviluppano il proprio stack TLS, ciò non è esattamente vero per tutti.

Ad esempio OpenSSL -leader in campo opensource- non fornisce alcuna API per controllare tale funzione. Il piano per risolvere questo problema è delineato nella PR 8797, che punta ad introdurre delle API simili a quelle di BoringSSL.

Questa situazione porterà ad incontrare ostacoli nella messa in campo della tecnologia QUIC, visto che i differenti stack dovranno basarsi su librerie TLS proprie o di terze parti, utilizzare versioni di OpenSSL modificate o necessitare di aggiornamenti vari.

# Kernel e carico sulla CPU

Google e Facebook hanno entrambi affermato che le loro installazioni QUIC su larga scala consumano circa il doppio della CPU per servire la stessa quantità di contenuti rispetto ad una più classica connessione HTTP/2 su TLS e TCP.

Spiegazioni plausibili a questo fenomeno includono:

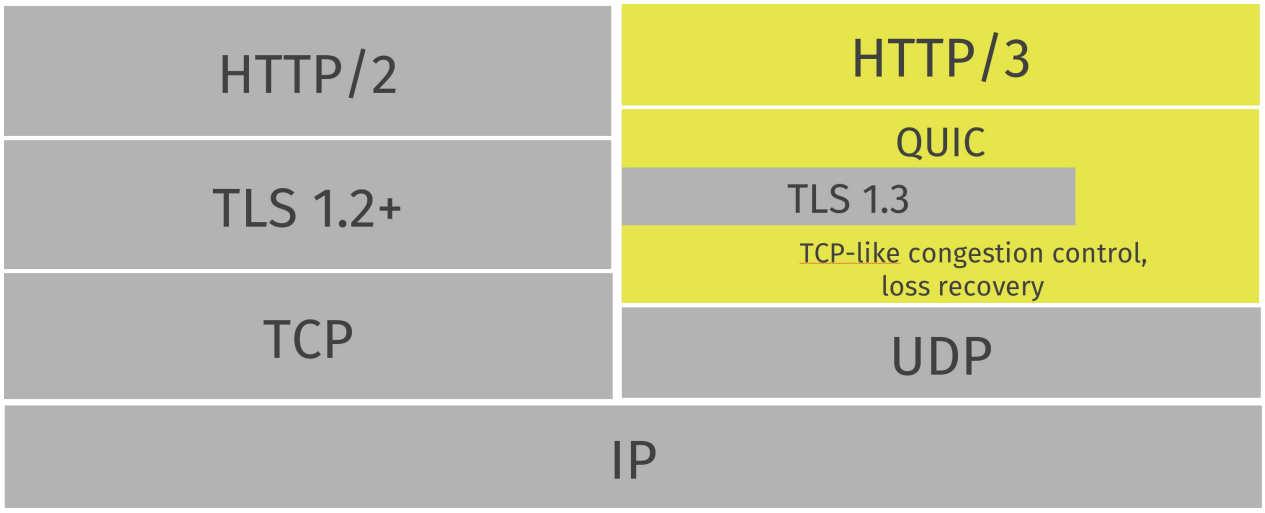
- il fatto che la parte UDP nel kernel Linux non sia altrettanto ottimizzata quanto lo stack TCP, dato che UDP non era finora mai stato utilizzato per trasferimenti ad altissima velocità come questi
- il fatto che per QUIC non esistano dispositivi di accelerazione hardware e offloading come invece è il caso per TCP e TLS. Notiamo rarissime eccezioni per dispositivi dedicati ad UDP.

Non vi sono ragioni per non essere convinti che le performance aumenteranno ed i requisiti CPU si abbasseranno nel tempo.

# Protocol features

Il protocollo QUIC ad alto livello.

Illustrati di seguito, lo stack HTTP/2 sulla sinistra e lo stack rete QUIC sulla destra, quando usati per il trasporto HTTP.



# Protocollo di trasporto via UDP

QUIC è un protocollo di trasporto implementato su UDP. Se si osserva il traffico di rete con un analizzatore di protocollo, il traffico QUIC appare composto di pacchetti UDP.

Basato su UDP, utilizza anche i numeri di porta per identificare con precisione un determinato servizio di rete su uno specifico indirizzo IP.

Tutte le implementazioni correnti e conosciute di QUIC sono sviluppate in "user-space" il che permette una evoluzione più rapida rispetto ad una tipica implementazione a livello di kernel-space.

# Funzionerà?

Ci sono aziende e altri setup di rete che bloccano il traffico UDP sulle porte diverse dalla 53 (DNS). Altri rallentanto i flussi in maniera da influenzare negativamente le prestazioni di QUIC, rendendolo piu lento che il TCP. Non vi è limite alle manipolazioni che gli operatori di rete possono applicare.

Nel futuro prossimo, tutti gli impieghi di trasporto basato su QUIC dovranno prevedere una modalita di fall-back indolore verso una modalità di trasporto alternativa (basata su TCP). Gli ingegneri di Google hanno potuto misurare il tasso di fallimento in percentuali a cifra singola, fra 2 e 5%.

# Apporterà miglioramenti?

Ci sono grosse probabilità che QUIC sia di valido aiuto al mondo Internet, chiunque vorrà poterlo usare e ci si aspetterà di vederlo funzionare con grande efficacia, fino al punto in cui le aziende cominceranno a considerare il cambiamento. Negli anni e grazie agli sviluppi apportati a QUIC, il tasso di successo nello stabilire connessioni QUIC è nettamente aumentato.

# Trasferimento di dati affidabile

Mentre sappiamo che UDP non è un modo di trasporto affidabile, notiamo come QUIC aggiunga uno strato di controllo, garantendo infine l'affidabilità. Offre retrasmissione di pacchetti, controllo della congestione, regolazione di flusso e altre caratteristiche gia presenti nel TCP.

I dati spediti attraverso QUIC da un end-point saranno trasportati prima o poi all'altro capo della connessione, tanto che quest'ultima sia mantenuta.

# Flussi multipli all'interno di una stessa connessione

Similmente a SCTP, SSH e HTTP/2, QUIC sfrutta flussi logicamente separati all'interno di una stessa connessione fisica. Un numero di stream paralleli può trasferire simultaneamente dati sulla stessa connessione senza influenzare gli altri flussi concorrenti.

Una connessione viene negoziata e messa in opera fra due punti remoti, in modo similare a quanto avviene per TCP. Una connessione QUIC è stabilita verso una porta e un indirizzo IP, tuttavia una volta stabilita la connessione viene identificata da un ID detto "ID di connessione".

All'interno di una connessione gia stabilita, ognuno dei due estremi può creare un flusso e spedire dei dati all'altro estremo. Un singolo flusso viene consegnato "in ordine" ed è ritenuto affifabile, mentre altri flussi paralleli potrebbero eventualmente essere ricevuti "fuori ordine".

QUIC offre un controllo di flusso a livello di connessione e di flusso.

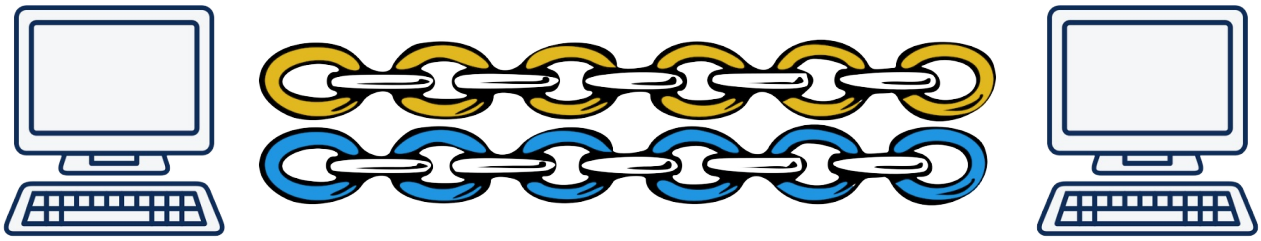
Maggiori dettagli nelle sezioni connessioni e streams.

# Consegna ordinata

QUIC garantisce una consegna ordinata dei flussi, ma non fra i flussi stessi. Questo significa che ogni flusso invierà dati e manterrà un ordine fra tali dati, ma ogni singolo flusso potrebbe raggiungere la destinazione in un ordine diverso da quello in cui l'applicazione lo avesse inizialmente spedito!

Per esempio: lo stream A e B sono trasferiti dal server al client. Il flusso A inizia per primom il B segue. In QUIC, la perdita di un pacchetto influenza solamente lo stream al quale tale pacchetto perso appartiene. Se uno stream A perdesse un pacchetto ma lo stream B no, il flusso B continuerebbe il proprio trasferimento mentre il pacchetto perso dallo stream A verrebbe ritrasmesso. Ciò non era possibile in HTTP/2.

Qui di seguito illustrati in giallo e blu due flussi inviati attraverso QUIC all'interno di una singola connessione. Essi sono indipendenti e potrebbero arrivare in un ordine diverso da quello inizialmente previsto, pur venendo entrambi consegnati correttamente a livello di applicazione, in ordine.



# Negoziazioni veloci

QUIC offre entrambi i metodi di connessione 0-RTT e 1-RTT, nel senso che nel migliore dei casi QUIC non ha bisogno di round-trips aggiuntivi per iniziare una nuova connessione. Il più veloce dei due metodi, la negoziazione a 0-RTT, funziona solo se vi è gia stata una connessione fra i due host in question e solo se il "segreto" per tale connessione è stato aggiunto alla cache.

# Dati in anticipo

QUIC permette ai client di aggiungere dati gia a partire dalla fase di negoziazione a 0-RTT. Questa caratteristica permette ad un client di spedire dati al proprio interlocutore il più veloce possibile; ciò permette al server di rispondere aggiungendo altri dati ancora più velocemente.

# TLS 1.3

La sicurezza del trasporto utilizzato da QUIC si basa su TLS 1.3 (RFC 8446) quindi non esistono connessioni QUIC non criptate.

TLS 1.3 presenta molteplici vantaggi se paragonato alle versioni precedenti; una delle principali ragioni dell'adozione di TLS da parte di QUIC consiste nel fatto che la 1.3 ha modificato la modalità di negoziazione al fine di richiedere meno va-e-vieni (round-trips). Ciò riduce la latenza del protocollo.

La versione oramai "legacy" di Google-QUIC utilizzava un algoritmo di cifratura personalizzato, non-standard.

# Livelli di Trasporto e Applicazione

Il QUIC di IETF è un protocollo di trasporto, sul quale altri protocolli applicativi possono essere innestati. L'applicazione principale (iniziale) di QUIC è al momento HTTP/3 (h3).

Il livello di trasporto supporta sia connessioni sia flussi.

La versione di QUIC sviluppata da Google presentava i layers di trasporto e applicazione (HTTP) come fusi insieme all'interno di un singolo processo, suonava molto come la "versione-speciale-che-manda-frames-http/2-via-udp".

# HTTP/3

Il layer HTTP si occupa dei trasporti in stile HTTP, inclusa la compressione degli header utilizzando QPACK - simile alla compressione usata in ambito HTTP/2, detta HPACK.

L'algoritmo HPACK si basa sulla consegna *ordinata* dei flussi quindi non è stato possibile riciclarlo per HTTP/3 senza apportarvi modifiche, visto che QUIC permette la consegna di flussi al di fuori dell'ordine predefinito. QPACK può essere visto come una versione di HPACK adattata su misura per QUIC.

# Non-HTTP via QUIC

Il lavoro sul tema dell'incapsulazione di protocolli non-HTTP dentro QUIC è stato posticipato: il tema sarà sviluppato in seguito alla consegna della prima versione ufficiale di QUIC.

# How QUIC works

Senza voler dettagliare esattamente bits e bytes °on the wire°, questa sezione si occupa di descrivere i blocchi fondamentali al funzionamento del protocollo di trasporto QUIC. Se vuoi implementare QUIC nel tuo proprio stack, questa descrizione dovrebbe esserti d'aiuto a comprenderne le basi; per maggiori dettagli meglio fare riferimento alle bozze ufficiali e alle RFC pubblicate da IETF.

1. Inizializzare una connessione
2. ... che includa sicurezza TLS
3. Infine usare i flussi

# Connessioni

Una connessione QUIC è una conversazione singola fra due end-points QUIC. La modalità di inizializzazione della connessione QUIC combina la negoziazione della versione con le negoziazioni dell'algoritmo crittografico e del trasporto al fine di ridurre la latenza di setup della connessione.

Per essere in grado di inviare dati attraverso tale connessione, uno o più flussi devono essere istanziati ed utilizzati.

## ID della connessione

Ogni connessione dispone di un insieme di identificatori di connessione, ognuno dei quali può essere utilizzato per identificare la connessione stessa. Gli ID sono selezionati indipendentemente da ognuna delle estremità; ogni endpoint assegna un ID alla connessione con il suo corrispondente.

La funzione primaria dell'ID di connessione è di assicurare che i cambiamenti nell'indirizzamento dei protocolli di livello inferiore (UDP, IP ed oltre) non provochino malfunzionamenti nella consegna dei pacchetti su connessioni QUIC ad esempio consegnandoli dal lato sbagliato.

Traendo vantaggio dall'ID di connessione, le connessioni possono dunque essere migrate attraverso indirizzi IP ed interfacce, in modi mai previsti dal TCP. Per esempio, si potrà migrare una connessione di download in corso su una rete mobile verso una connessione wifi a banda larga, senza interrompere il download. In modo analogo, il download potrà procedere su rete mobile se la connessione wifi dovesse interrompersi.

## Numeri di porta

Essendo QUIC costruito su UDP, un campo a 16bit è utilizzato per distinguere le connessioni in ingresso.

## Negoziazione delle versioni

Una connessione QUIC originata da un client renderà noto al server quale versione QUIC esso preferisca parlare, ed il server risponderà con una lista delle versioni supportate, lasciando la scelta finale al client.

# Connessioni su TLS

Successivamente al pacchetto iniziale che si occupa di istanziare la connessione, l'estremo che ha iniziato la connessione invierà un "crypto-frame" che scaturisce la negoziazione dello strato di sicurezza. Lo strato di sicurezza si basa su TLS 1.3.

Non vi è modo di evitare o declinare l'uso di TLS per una connessione QUIC. Il protocollo è specificamente concepito per evitare che le "middle-boxes" possano intercettare o modificare il traffico, allo scopo di evitare una ossificazione prematura del protocollo.

# Streams (flussi)

I flussi QUIC permettono l'astrazione leggera e ordinata della sequenza di bytes.

Esistono due tipi di flussi (streams) in QUIC:

- Flussi unidirezionali che trasportano dati in una sola direzione: dal produttore dello stream all'altro estremo, il destinario.
- Flussi bidirezionali che permettono di inviare dati in entrambe le direzioni.

Ciascuno dei due tipi di flusso può essere creato da ognuno dei due estremi, può inviare dati allo stesso tempo su flussi alternati, o può essere soppresso.

Per mandare dati all'interno di una connession QUIC, è necessario utilizzare uno o più flussi.

# Controllo di flusso

I flussi sono controllati individualmente, peremettendo ad ogni estremo di limitare l'allocazione della memoria e di applicare una pressione inversa. La creazione dei flussi è anch'essa soggetta a controllo di flusso; si richiede ad ogni peer di dichiarare l'ID di flusso "massimo" al quale si è disposti ad arrivare (dunque l'ID corrisponde al numero totale di flussi gestibili da ogni estremo).

# Identificatori di flusso

I flussi sono definiti da un intero di 62-bit privo di segno, al quale si è soliti riferirsi come "Stream ID". All'interno dello Stream-ID, i due bits meno significativi sono utilizzati per identificare il tipo di flusso (uni o bi-direzionale) e chi abbia inizializzato la comunicazione.

Il bit meno significativo dello Stream-ID (0x1) identifica chi dei due abbia inizializzato il flusso. In caso sia il client ad iniziare la connessione il numero di flusso risulterà parii (LSB a 0); tale numero sarà dispari in caso di flusso iniziato dal server (LSB a 1).

Il secondo bit meno significativo dello Stream-ID (0x2) è utilizzato per distinguere fra flussi uni e bi-direzionali. Flussi unidirezionali avranno impostato il bit a 1, flussi bidirezionali utilizzeranno sempre il valore 0.

# Concorrenza di flussi

QUIC permette ad un numero arbitrario di flussi di operare in concomitanza. Un estremo limita il numero di flussi concorrenti in entrata comunicando lo Stream-ID massimo che è disposto ad accettare.

Il massimo valore di Stream-ID è specifico ad ogni estremo e si applica solo all'estremità che riceve tale impostazione.

# Spedire e ricevere dati

Gli end-points utilizzano i flussi per spedire e ricevere dati. In fondo è questo il loro fine ultimo. Gli streams sono un flusso "astratto" di bytes ordinati. Streams separati non saranno necessariamente consegnati nello stesso ordine di invio.

# Prioritizzazione dei flussi

Il multiplexing dei flussi ha un impatto significativo sulle performance dell'applicazione, in caso le risorse assegnatevi siano correttamente ordinate per priorità. L'esperienza con altri protocolli multiplexati -quale HTTP/2- mostra che una buona strategia di prioritizzazione ha certamente un impatto positivo sulle prestazioni.

QUIC in se non mette a disposizione frames adatti a scambiare informazioni sullo stato delle priorità. Al contrario si accontenta di ricevere tali informazioni direttamente dall'applicazione essa stessa basata su QUIC. I protocolli che utilizzano QUIC sono liberi di definire il proprio schema di prioritizzazione secondo le proprie necessità e semantiche applicative.

Quando si utilizza HTTP/3 su QUIC, la prioritizzazione è gestita all'interno dello strato HTTP.

# 0-RTT

Per ridurre il tempo richiesto per stabilire una nuova connessione, un client che si sia precedentemente collegato ad un server potrebbe mantenere in cache alcuni parametri relativi a tale connessione ed utilizzarli successivamente per iniziare una connessione a **0-RTT** con il server. Questa modalità permette al client di inviare dati fin dal primo pacchetto, senza dover aspettare il termine della negoziazione stessa.

# Spin Bit (Bit rotante)

Una delle discussioni più estenuanti all'interno del gruppo di lavoro QUIC . oggetto di mille emails e ore di dispute verbali- è legata ad un singolo bit: il bit rotante (spin bit).

I fautori dello spin-bit insistono nell'affermare quanto -per operatori ed amministratori di rete- sia fondamentale poter misurare la latenza fra due end-points QUIC.

Gli opppositori dello spin-bit temono una potenziale fuga di informazioni.

# Girare un bit

Entrambi gli estremi mantengono temporaneamente un valore di 0 o 1 per ogni singola connessione QUIC; client e server sono incaricati di impostare un valore all'interno di ciascun pacchetto spedito.

Entrambi gli estremi inviano il pacchetto impostando lo stesso valore per la durata di un solo round-trip, alchè invertono il valore. Il risultato è dunque una serie di pulsazioni di 0 e 1 utile al monitoraggio.

Questo tipo di misurazione è efficace solo se il peer che invia non è limitato a livello di applicazione o di controllo di flusso, in assenza di riordino di pacchetti (causato dalla rete stessa).

# User-space (spazio-utente)

Implementare un protocollo di trasporto all'interno dello user-space permette di poter manipolare/programmare il protocollo facilmente, senza necessita di aggiornare il kernel o il sistema operativo ad ogni evoluzione delle librerie client/server.

Non vi sono tuttavia ostacoli fattuali alla implementazione di QUIC all'interno del kernel; qualcuno in futuro se ne occuperà, se veramente ve ne sarà la necessità.

## Molte implementazioni

L'implementazione di un nuovo protocollo di trasporto all'interno dello user-space ha per effetto la nascita di molteplici implementazioni.

In un futuro prossimo, è certo che le diverse applicazioni conterranno (o si baseranno) su implementazioni eterogenee di HTTP/3 e QUIC.

# API

Uno dei fattori del successo del classico TCP e dei programmi su di esso basati è stata la disponibilita di API socket standardizzazate. Poter offrire funzionalità ben definite usando tali API permette di "portare" programmi fra diversi sistemi operativi dato che TCP funziona in maniera identica indipendentemente dalla piattaforma.

QUIC non è ancora arrivato a quel punto. Non esistono API standard in QUIC.

Con QUIC, ci si deve basare su una delle librerie esistenti e fare affidamento alle API fornite dalla libreria scelta. Ciò fa si che le applicazioni siano "abbinate" ad una singola libreria, almeno in teoria. Cambiare ed adottare un'altra libreria signigica nuove API e potrebbe richiedere molto lavoro di adattamento.

Inoltre, dato che QUIC è spesso implementato in user-space, non è adatto ad estendere le API socket o offrire funzionalità in qualche modo simili agli attuali TCP e UDP. Usare QUIC significa utilizzare un'API esterna al socket.

# HTTP/3

Come sottolineato in precedenza, il primo e principale protocollo trasportato via QUIC è HTTP.

In maniera simile a quanto successe per HTTP/2 -introdotto per trasportare HTTP in binario- HTTP/3 introduce una nuova modalità di trasporto per inviare sintassi HTTP sulla rete.

HTTP mantiene ancora gli stessi paradigmi e concetti che gia conteneva in precedenza. Vi sono intestazioni e corpo del messaggio, vi sono richieste e risposte. Esistono verbi, cookies e meccanismi di ritenzione del contenuto (cache). Ciò che in sostanza cambia in HTTP/3 è la modalità di trattamento (trasferimento) riservata alle informazioni da trasferire all'altro capo.

Per fruire di HTTP via QUIC sono stati necessari alcuni cambiamenti, il risultato dei quali prende il nome di HTTP/3. Questi cambiamenti sono legati alla diversa natura di QUIC rispetto a TCP. Le modifiche includono:

- In QUIC i flussi sono forniti dal meccanismo di trasporto stesso, mentre in HTTP/2 i flussi sono gestiti all'interno dello strato HTTP.
- Per via del fatto che i flussi in QUIC sono independenti l'uno dall'altro, non è stato possibile utilizzare la stessa identica modalità di compressione delle intestazioni gia impiegata in HTTP/2 poichè essa avrebbe provocato "bloccaggi di inizio riga".
- Gli streams QUIC sono lievemente diversi da quelli HTTP/2. La sezione dedicata a HTTP/3 tratta più in dettaglio l'argomento.

# URLs HTTPS://

HTTP/3 si servirà di URLs del tipo `HTTPS://`. Il mondo è pieno di questo tipo di URL ed è stato osservato che sarebbe impraticabile ed illogico proporre un ulteriore schema di URL per il nuovo protocollo. Similmente a come HTTP/2 non avesse bisogno di un nuovo schema, cosi fu per HTTP/3.

La complessità aggiunta da HTTP/3 è evidente se guardiamo al fatto che HTTP/2 -seppur rivoluzionario nel trasporto di HTTP a livello di trama- era ancora basato su TLS e TCP come nel caso del predecessore HTTP/1. Il fatto che HTTP/3 lavori su QUIC fa vacillare un certo numero di assunzioni.

Il caro vecchio URL "clear-text" `HTTP://` verrà lasciato intatto e sarà via via abbandonato nel momento in cui si passerà a modalità di trasporto più sicure. Richieste verso questo tipo di URL non riceveranno alcun upgrade a HTTP/3. Nella realtà dei fatti, ben poche di queste connessioni ricevono upgrade a HTTP/2, ma per altri motivi.

## Connessione iniziale

La prima connessione ad una risorsa o host non ancora visitato via URL HTTPS:// sarà probabilmente stabilita via TCP (eventualmente in parallelo con un tentativo di connessione QUIC). L'host distante potrebbe essere un vecchio server che non fornisce supporto a QUIC, o potrebero esservi dispositivi sul percorso di rete che impediscono il successo della connessione via QUIC.

Un client moderno negozierebbe probabilment HTTP/2 in prima istanza. Una volta che tale connessione fosse stata effettuata, e che il server avesse risposto ad una richiesta HTTP, il server potrebbe comunicare al client la disponibilità del supporto e la preferenza per HTTP/3.

# Alt-svc

L'intestazione di servizio alternativo (Alt-svc:) e il corrispettivo frame HTTP/2 `ALT-SVC` non sono specificamente creati per QUIC o HTTP/3. Tali header sono parte di un meccanismo gia ben rodato per cui un server può segnalare ad un client *"guarda, io erogo questo stesso servizio ANCHE su QUESTO HOST, utilizzando QUESTO PROTOCOLLO, su QUESTA PORTA"*. Vedi i dettagli nella RFC 7838.

In caso lo supportasse e lo desiderasse, ad un client che ricevesse tale header Alt-Svc in una risposta verrebbe consigliato di connettersi in parallelo ed in background all'host suggerito -utilizzando il protocollo definito- ed in seguito completare il resto delle operazioni utilizzando questo nuovo canale, al posto della connessione iniziale.

In caso la connessione iniziale stesse utilizzando HTTP/2 o addirittura HTTP/1, il server può decidere di rispondere al client di riprovare tale connessione via HTTP/3. Tale connessione potrebbe essere diretta allo stesso host, o ad un altro host distante, capace di servire tale contenuto sul protocollo prescelto. L'informazione fornita all'interno dell'header Alt-Svc è dotata di una data di scadenza, che permette la redirezione dei clients verso host alternativi entro un certo lasso temporale.

# Esempio

Un server HTTP include nella propria risposta un header di tipo `Alt-Svc:` :

```
Alt-Svc: h3=":50781"
```

Questo indica che HTTP/3 è disponibile via UDP sulla porta 50781, sullo stesso host utilizzato in prima istanza.

Un client può a quel punto tentare di stabilire una connessione QUIC verso la destinazione indicata, ed in caso di successo continuerebbe a comunicare con l'origine attraverso il nuovo canale, non attraverso l'originale in HTTP.

# Flussi QUIC e HTTP/3

HTTP/2 è stato costretto a concepire il proprio schema di flussi e multiplexing basandosi su TCP, mentre HTTP/3 è studiato per lavorare con QUIC e quindi trae massimo vantaggio dall'utilizzo degli streams.

Le richieste HTTP veicolate su HTTP/3 sfruttano un set di flussi specifico.

# Frames HTTP/3

Dialogare in HTTP/3 implica l'attivazione di un flusso QUIC e l'invio di una serie di frames all'altro capo della connessione. Esistono al momento (alle 9 del 18 Dicembre 2018) una serie abbastanza limitata di frames predefiniti in HTTP/3. I più rilevanti sono seza dubbio:

- HEADERS, si occupano di comprimere e spedire le intestazioni
- DATA, tratta l'invio di dati binarii
- GOAWAY, pregasi terminare la connessione

# Richiesta HTTP

Il client invia la propria richiesta HTTP su un flusso QUIC *bidirezionale* iniziato dal client.

Una richiesta consiste di un singolo frame HEADERS e può essere seguita da uno o due frames aggiuntivi: una serie di frames DATA ed eventualmente un frame finale contenente gli HEADERS di coda.

Dopo aver spedito la propria richiesta, il client termina il flusso in uscita.

# Risposta HTTP

Il server replica spedendo la risposta HTTP sullo stream bidirezionale. Un frame HEADERS, una serie di DATA ed eventualmente un frame HEADERS di coda.

# Intestazioni QPACK

I frames HEADERS contengono le intestazioni HTTP compresse con l'algoritmo QPACK. QPACK risulta simile al vecchio HPACK utilizzato in HTTP/2 (RFC 7541 ), benchè modificato per consegnare i differenti flussi in modalità "disordinata" (out-of-order).

QPACK si avvale di due streams QUIC unidirezionali fra i due estremi della connessione. Tali streams sono utilizzati per trasportare la tabella dinamica di compressione nelle due direzioni opposte.

# Prioritizzazione in HTTP/3

Uno dei frame facenti parte di ogni flusso HTTP/3 è il campo `PRIORITY`. E' usato per segnalare la priorità e l'ordine di dipendenza fra flussi diversi molto similmente a quanto gia avveniva in HTTP/2.

Il frame può essere impostato in modo che un flusso dipenda da un altro e può definire un "peso specifico" su ogni determinato flusso.

Un flusso dipendente avrà diritto ad allocare risorse solamente se tutti gli altri flussi dai quali esso dipende fossero gia in fase di chiusura, o non fosse possibile avanzare ulteriormente su tali suddetti flussi.

Uno stream assume un valore compreso fra 1 e 256. La regola vuole che un flusso con uno stesso antenato **dovrebbe** allocare risorse in maniera proporzionale basandosi sul peso specifico di ognuna.

# HTTP/3 Server push

La modalita "server push" di HTTP/3 è tutto sommato simile a quanto descritto per HTTP/2 nella (RFC 7540 ) pur usando meccanismi diversi.

Una "server push" è a tutti gli effetti la risposta ad una richiesta che il client in realtà non ha mai inviato !

Le "server push" sono difatti autorizzate solamente in caso il client abbia deciso di accettarle. In HTTP/3 il client imposta addirittura un limite al numero massimo di risposte push che è disposto ad accettare, comunicando al server l'ID di fluso più alto che sarà disposto a trattare. Se il server dovesse tentare di inviare flussi al di sopra dell'ID indicato, ciò provocherebbe un errore di connessione.

Se il server dovesse pensare che il client possa aver bisogno di una determinata risorsa -risorsa comunque ritenuta necessaria- potrebbe volergli inviare un frame detto `PUSH_PROMISE` (all'interno dello stesso flusso della richiesta) contenente le coordinate della richiesta per la risorsa in questione ed in seguito inviare la risposta all'interno di un nuovo flusso.

Anche nel momento in cui le "server push" dovessero essere ritenute credibili da parte del client, ognuna delle singole "push" può essere annullata ad ogni momento se il client dovesse in qualche modo ritenerlo necessario. In tal caso invierebbe un frame `CANCEL_PUSH` al server.

# Problematiche

Questa caratteristica è stata messa in discussione, criticata e rigirata sin dal primo momento, sia durante lo sviluppo di HTTP/2 sia dopo la sua pubblicazione e impiego su larga scala, tentando di renderla utilizzabile.

Un invio in "push" non è mai gratis, benchè utilizzi solo metà del tempo di round-trip, occupa banda passante. E' spesso difficile -impossibile- dal punto di vista del server stabilire con certezza se una determinata risorsa possa necessitare (beneficiare) del "push" o meno.

# HTTP/3 paragonato a HTTP/2

HTTP/3 è stato disegnato per QUIC, che è un protocollo di trasporto autonomo nel gestire i proprii flussi.

HTTP/2 è stato concepito su TCP, quindi gestisce i sotto-flussi all'interno dello strato HTTP.

## Somiglianze

I due protocolli hanno virtualmente un insieme di caratteristiche identico.

- Entrambi offrono streams (flussi)
- Entrambi supportano la modalità "server push"
- Entrambi utilizzano la compressione degli header, QPACK e HPACK risultano molto simili nel design funzionale
- Entrambi sfruttano il multiplexing via flussi, su una singola connessione
- Entrambi i protocolli contengono la nozione di priorità di flusso

## Differenze

Le differenze riguardano i dettagli, soprattutto in relazione al fatto che HTTP/3 usi QUIC:

- HTTP/3 funziona meglio con i "dati anticipati" grazie al supporto per la negoziazione a 0-RTT, mentre sappiamo che il TCP Fast Open accoppiato a TLS spesso incontra problemi, inviando comunque minor quantità di dati
- Le negoziazioni (handshakes) sono molto più rapide in HTTP/3 grazie a QUIC
- Non esistono versioni non crittate o non sicure di HTTP/3. HTTP/2 può ancora essere implementato ed utilizzato senza HTTPS - benchè molto raro su Internet
- HTTP/2 può essere negoziato direttamente all'interno di una negoziazione TLS grazie all'estensione ALPN. Diversamente, in HTTP/3 (essendo su QUIC) si dovrà a priori informare il client con una risposta contenente l'intestazione `Alt-Svc:` prima che il client stesso possa avere conoscenza di tale risorsa.

# Critiche comuni

## UDP non funzionerà mai

Una marea di aziende, operatori ed organizzazioni hanno l'abitudine di bloccare o limitare tutto il traffico UDP non afferente alla porta 53 (DNS) dato che [UDP] è stato spesso sfruttato in attacchi DDoS. Nello specifico alcuni dei protocolli UDP esistenti e le rispettive implementazioni server, sono stati recentemente coinvolti in attacchi detti "di amplificazione", dove a partire da un piccolo volume di traffico si arriva a generare una vera e propria onda di traffico, onde "mirate" verso un obiettivo innocente.

QUIC contiene una tecnica di mitigazione contro gli attacchi "amplificatori" esigendo che il pacchetto iniziale consti almeno di 1200 bytes ed impiegando una restrizione a livello di protocollo che IMPEDISCE al server di inviare una risposta più grande di tre volte il volume della richiesta originale, senza prima aver ricevuto un pacchetto di conferma dal client.

## UDP è lento nel kernel

Sembra essere vero, almeno ad oggi nel 2018. Non possiamo ovviamente predire in che direzione andranno gli sviluppi, o quanto questo sia il risultato dei lunghi anni di indifferenza da parte degli sviluppatori verso le prestazioni di UDP stesso.

La maggior parte dei client non percepisce affatto questa lentezza.

## QUIC prende troppa CPU

Analogamente a quanto detto sopra -la critica "UDP è lento"- questa situazione deriva dal fatto che TCP e TLS hanno goduto di molto più tempo per maturare, per evolvere, ed hanno conseguentemente beneficiato di un supporto hardware e software più cospicuo.

Abbiamo ragione di credere che questa situazione evolverà in futuro. Una domanda rimane: fino a che punto questi problemi di CPU determineranno un ostacolo per i responsabili della rete ?

## C'è solo Google

Non è vero. Google è reponsabile per aver presentato la prima specifica allo IETF, dopo aver dimostrato la potenziale efficacia nell'adozione di un tale protocollo, costruito sul gia noto UDP.

A partire da allora, una molteplicità di persone di compagnie ed entità diverse ha lavorato all'interno dell'organizzazione neturale IETF, al fine di creare un protocollo di trasporto standardizzato. Ovviamente, gli impiegati di Google hanno participato al lavoro di gruppo, ma non dobbiamo trascurare un gran numero di esperti che hanno interesse a contribuire all'evoluzione del trasporto Internet, quali Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook e Apple.

## Non rappresenta un gran miglioramento

Questa non è una vera critica ma piuttosto un'opinione. Magari è il caso, magari il miglioramento non è davvero così netto considerando che HTTP/2 è ancora molto recente.

Si suppone che HTTP/3 sarà più performante in reti con alto tasso di perdita di pacchetti, oltre ad essere in grado di attuare una negoziazione più veloce; la latenza percepita e la latenza effettiva ne beneficieranno sicuramente. Saranno questi benefici sufficienti a motivare gli utenti e gli editori a migrare verso un supporto per HTTP/3 a livello di servizi e software? Solo il tempo ed un buon assessment delle prestazioni potranno rispondere!

# Le specifiche

Qui troviamo una collezione delle ultimissimee bozze ufficiali per tutti i vari pezzi e componenti di QUIC e HTTP/3.

## Invarianti (Costanti)

Proprietà di QUIC indipendenti dalla versione

## Trasporto

QUIC: un trasporto "multiplexato" e sicuro basato su UDP

## Recupero ("recovery")

Rilevamento delle perdite e controllo della congestione in QUIC

## TLS

Usare TLS (sicurezza a livello di trasporto) per securizzare QUIC

## HTTP

Protocollo di trasferimento di ipertesto (HTTP) via QUIC

## QPACK

QPACK: Compressione degli Header per HTTP via QUIC

# QUIC v2

Per potersi concentrare il più possibile sul cuore del protocollo QUIC ed essere in grado di consegnarlo per tempo, diverse opzioni che inizialmente avrebbero dovuto essere incluse come base del procollo hanno purtroppo dovuto essere posticipate fino al punto da essere state pianificate in una versione successiva di QUIC. Appunto, QUIC versione 2 o maggiore.

L'autore di questo documento dispone di una sfera di cristallo non proprio perfetta, quindi non può predire esattamente quali di queste caratteristiche saranno o meno integrate nella versione 2. Possiamo tuttavia citare alcune delle opzioni esplicitamente rimosse dalla v1 -per essere sviluppate in un secondo momento- opzioni che potrebbero perciò essere presenti nella v2.

## Correzione dell'errore anticipata (Forward Error Correction)

La FEC (Forward Error Correction) è un metodo di controllo d'errore relativo alle trasmissioni dati in cui un emissario (sender) spedisca una quantità di dati ridondante e in cui il ricevente (receiver) riconosca solamente la porzione di dati che non contenga alcun errore apparente.

Google ha sperimentato questa tecnica nella sua implementazione originaria di QUIC ma fu successivamente rimossa in quanto gli esperimenti non diedero risultati soddisfacenti. Questa opzione è soggetta a discussione per entrare nella seconda versione di QUIC, ma necessita di argomenti a proprio favore nel mostrare che non penalizzi o influenzi negativamente le prestazioni.

## Multipath

Il "multipath" è una tecnica di trasporto che per definizione utilizza percorsi di rete complementari (multipli ed alternativi) al fine di massimizzare l'utilizzo delle risorse ed aumentarne l'affidabilità attraverso la ridondanza.

I fautori e sostenitore di SCTP tengono a sottolineare come SCTP contempli gia il multipath, così come il TCP più moderno.

## Dati inaffidabili

E' stato proposto di offrire streams "non-affidabili" come opzione, il che permetterebbe a QUIC di rimpiazzare qualsiasi applicazione che utilizzi UDP.

## Adattamenti che non riguardano HTTP

DNS over QUIC è stato uno dei protocolli non-HTTP di cui si è parlato per primo, e al quale probabilmente sarà dedicata molta attenzione una volta che QUIC v1 e HTTP/3 saranno consegnati al pubblico. Ovviamente, con la nascita di un nuovo tipo di trasporto come questo, sono sicuro che il campo di applicazione non si limiterà al solo DoQ.