

PROTOCOLLO DI TRASPORTO

QUIC

Architettura, funzionamento e analisi del protocollo di nuova generazione

 **Andrea Cazzato**

Internet, Reti e Sicurezza · Esame Integrativo

Panoramica

Cos'è QUIC?

Protocollo di trasporto moderno sviluppato da Google e standardizzato dall'IETF nel 2021 (RFC 9000). Sostituisce TCP per il trasporto HTTP/3.

Obiettivo

Risolvere i limiti strutturali di TCP nel web moderno: latenza elevata, head-of-line blocking e impossibilità di aggiornamento del protocollo.

Impatto reale

HTTP/3 è QUIC. Usato da Google, YouTube, Cloudflare e Meta per miliardi di richieste al giorno su scala globale.

Limiti di TCP

TCP nasce negli anni '70 per reti inaffidabili e lente: il contesto odierno è radicalmente cambiato

Head-of-Line Blocking

Se un pacchetto viene perso, tutti i pacchetti successivi devono aspettare prima di essere consegnati all'applicazione, anche se appartengono a flussi diversi e la loro ricezione è già avvenuta.

Handshake Lento

TCP richiede un round-trip solo per stabilire la connessione, a cui si aggiunge il round-trip di TLS. Fino a 3 RTT (Round-Trip Time) prima di trasmettere i primi dati utili.

Ossificazione del Protocollo

Firewall, NAT e router ispezionano e modificano i campi degli header TCP. Questo rende praticamente impossibile aggiornare il protocollo: qualsiasi modifica rischia di essere bloccata o alterata dai middlebox.

HTTP/2 e i suoi limiti

✓ Cosa ha risolto

Multiplexing logico

Più richieste HTTP su una singola connessione TCP, senza aprire più socket.

Compressione degli header (HPACK)

Riduce la dimensione degli header HTTP, risparmiando banda in modo significativo.

Server Push

Il server può inviare risorse proattivamente prima che il client le richieda esplicitamente.

✗ Problemi rimasti

Head-of-Line Blocking persiste

Il multiplexing è logico ma TCP non lo conosce: la perdita di un pacchetto blocca comunque tutti gli stream contemporaneamente.

Handshake ancora lento

TCP + TLS richiedono fino a 3 round-trip prima di poter scambiare dati effettivi.

Ossificazione invariata

HTTP/2 lavora ancora su TCP: tutti i limiti di aggiornabilità del protocollo rimangono irrisolti.

Nascita e standardizzazione di QUIC

2012

Google inizia la sperimentazione interna del protocollo QUIC nei propri data center, con l'obiettivo di ridurre la latenza.

2015

Google presenta QUIC pubblicamente e propone la standardizzazione all'IETF. Il lavoro diventa collaborativo e aperto.

2018

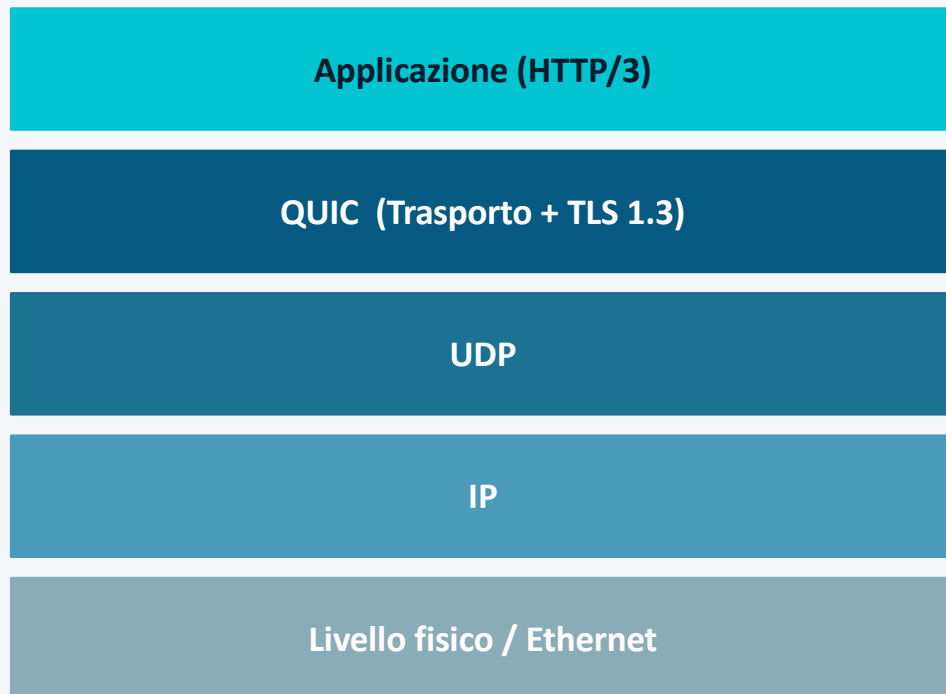
L'IETF avvia i lavori formali su QUIC v1; la specifica si distanzia dalla versione Google originale per maggiore generalità.

2021

Pubblicazione della RFC 9000: QUIC diventa standard ufficiale. HTTP/3 (RFC 9114) è definito come HTTP su QUIC.

QUIC su UDP

Una scelta deliberata, non una limitazione



Aggiornabilità

QUIC gira in user space: può essere aggiornato e modificato senza toccare il kernel del sistema operativo, rendendo l'evoluzione del protocollo molto più rapida.

Bypass dei middlebox

I router non possono ispezionare o modificare i meccanismi interni: quasi tutto il traffico QUIC è cifrato, inclusi header e metadati di trasporto.

Affidabilità su misura

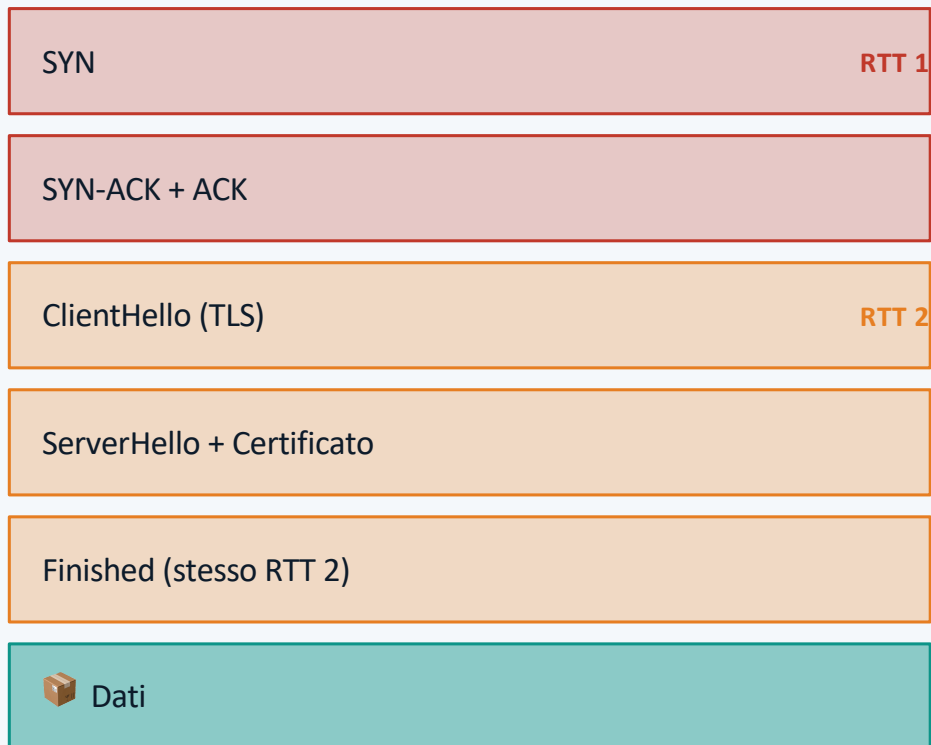
UDP non garantisce affidabilità, ma QUIC la implementa a livello applicativo con maggiore flessibilità rispetto ai meccanismi fissi di TCP.

Privacy del traffico

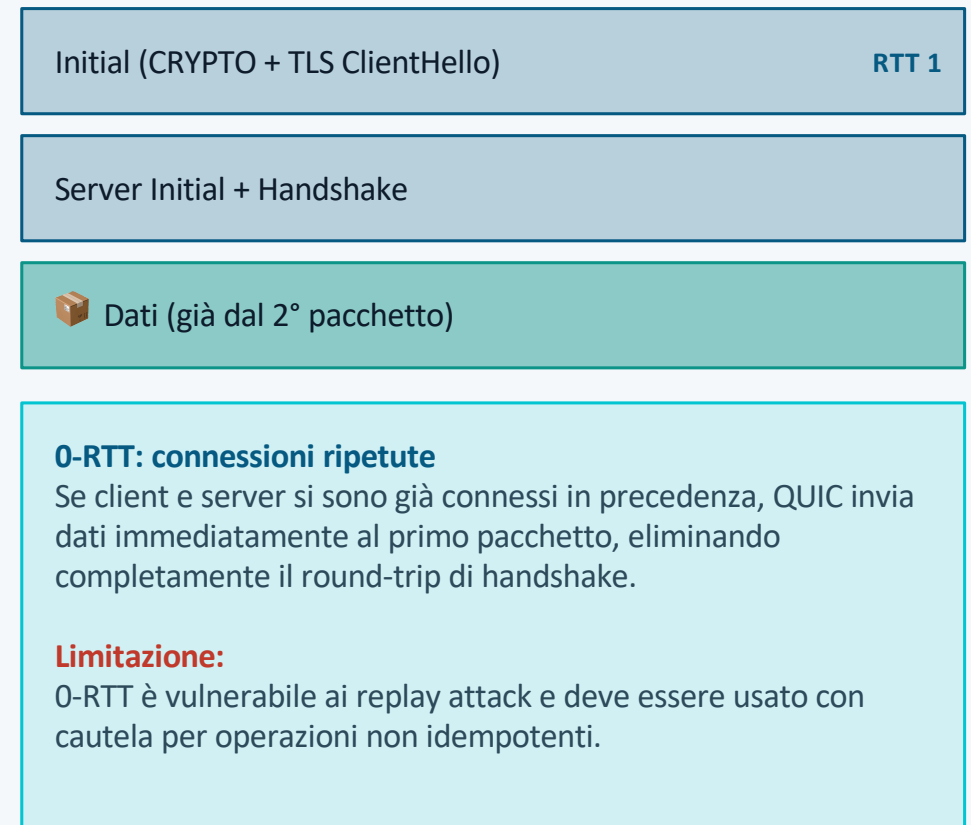
La cifratura pervasiva impedisce l'analisi del traffico da parte di terzi, inclusi ISP, firewall aziendali e attori malintenzionati sulla rete.

Handshake: TCP vs QUIC

TCP + TLS 1.3

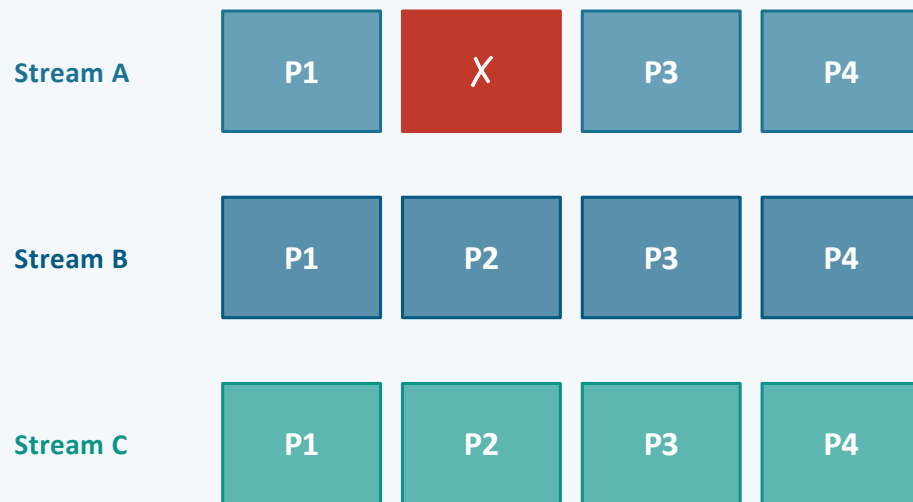


QUIC (1-RTT)



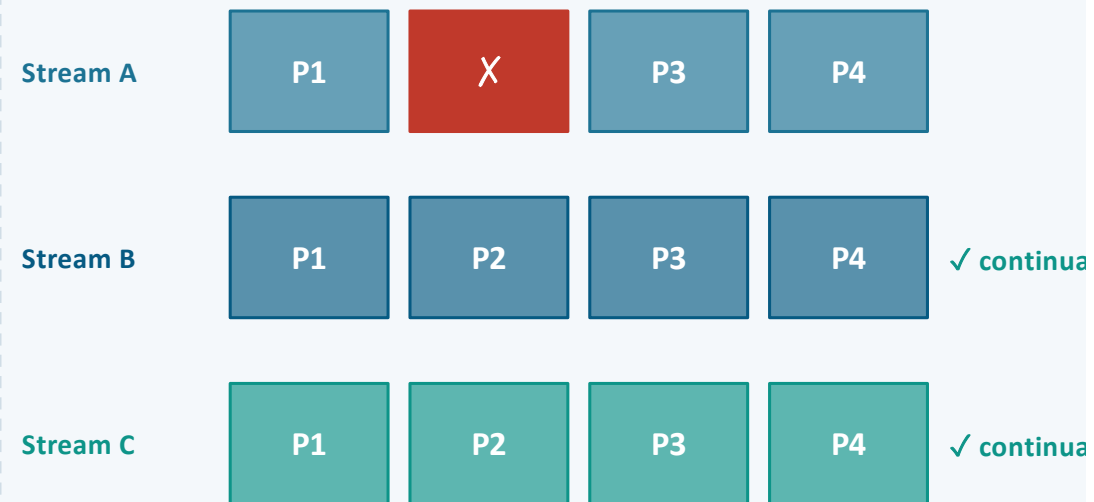
Stream Multiplexing

TCP: Head-of-Line Blocking



P2 perso su Stream A → tutti e tre gli stream sono bloccati

QUIC: Stream Indipendenti



P2 perso su Stream A → Stream B e C continuano normalmente

In QUIC ogni stream è gestito in modo del tutto indipendente. La perdita di un pacchetto su uno stream causa solo la ritrasmissione di quel pacchetto specifico, senza influenzare il flusso degli altri stream.

Controllo della Congestione e Affidabilità

Affidabilità su UDP

QUIC implementa i propri meccanismi di acknowledgment, ritrasmissione e rilevamento perdite, senza dipendere da TCP. UDP è solo il vettore di trasporto.

Numeri di Sequenza Monotoni

I packet number non vengono mai riutilizzati, nemmeno nelle ritrasmissioni. Questo elimina l'ambiguità degli ACK tipica di TCP (problema del Karn's Algorithm) e permette una stima precisa del RTT.

Algoritmi di Congestione Moderni

QUIC supporta algoritmi avanzati come CUBIC e BBR. TCP non può adottarli facilmente per i vincoli di retro-compatibilità del kernel. Con QUIC, l'algoritmo può essere aggiornato senza modificare l'OS.

ACK più informativi (Selective ACK)

Gli acknowledgment di QUIC possono riferirsi a blocchi non contigui di pacchetti ricevuti, simile a SACK ma più generalizzato. Il mittente identifica con precisione i pacchetti persi ed evita ritrasmissioni inutili.

Migrazione di Connessione



TCP

Identificazione connessione:

{ IP src, Porta src, IP dst, Porta dst }

Se uno qualsiasi di questi 4 valori cambia, ad esempio passando da una rete WiFi a una connessione 4G, la connessione TCP si interrompe irrimediabilmente.

L'applicazione deve riconnettersi da capo, perdendo completamente lo stato della sessione precedente. Per chiamate, streaming o download in corso, questo si traduce in interruzioni visibili all'utente.



QUIC

Identificazione connessione:

Connection ID (scelto arbitrariamente dal client)

Il Connection ID è completamente indipendente da indirizzi IP e porte. Anche cambiando rete, ad esempio da WiFi aziendale a 4G mentre si è in una videochiamata, la connessione QUIC sopravvive senza alcuna interruzione.

Questa funzionalità è fondamentale per applicazioni mobile e real-time, dove i cambi di rete sono frequenti.

Sicurezza Integrata

In TCP, TLS è opzionale e aggiunto sopra il trasporto. In QUIC, TLS 1.3 è parte integrante e obbligatoria del protocollo.

01

TLS 1.3 Nativo e Obbligatorio

Non è possibile usare QUIC senza crittografia. TLS 1.3 è integrato a livello di protocollo, non aggiunto come strato separato come avviene in TCP.

02

Header di Trasporto Cifrati

In TCP gli header sono in chiaro e possono essere letti da chiunque sulla rete. In QUIC, anche i metadati di trasporto (numeri di pacchetto, flag di stream) sono protetti dalla crittografia.

03

Resistenza alla Traffic Analysis

La cifratura pervasiva rende molto più difficile l'analisi del traffico da parte di terzi, inclusi ISP e firewall. Il contenuto dei pacchetti, così come la struttura degli stream, è opaco a osservatori esterni.

04

Protezione dai Middlebox

Non potendo leggere i campi interni, i middlebox non possono manomettere la connessione. Questo elimina una classe intera di vulnerabilità e attacchi presenti in TCP.

TCP vs QUIC: Confronto diretto

Caratteristica	TCP	QUIC
Trasporto base	TCP (kernel)	UDP (user space)
Handshake (nuova connessione)	2 – 3 RTT	1 RTT
Handshake (connessione ripetuta)	2 RTT	0 RTT
Multiplexing nativo	No	Sì
Head-of-Line Blocking	Sì	No
Migrazione di connessione	No	Sì, Connection ID
Sicurezza	TLS opzionale e separato	TLS 1.3 integrato e obbligatorio
Aggiornabilità del protocollo	Difficile (kernel)	Semplice (user space)

Performance nel Mondo Reale

~30%

riduzione latenza su reti mobili degradate

Fonte: Google / YouTube

0-RTT

handshake per connessioni ripetute

vs 2 RTT di TCP+TLS

↓15%

buffering video su reti con perdita pacchetti

Fonte: Google Research

Scenari comparativi

Scenario	TCP	QUIC	Osservazione
Rete stabile, bassa latenza	Ottimo	Ottimo	Performance comparabili
Rete con perdita pacchetti > 2%	Degradato	Robusto	QUIC significativamente meglio
Cambio rete (WiFi → 4G)	X si interrompe	✓ continua	Solo QUIC sopravvive
Handshake connessione ripetuta	2 RTT	0 RTT	QUIC molto più veloce
Molti stream paralleli	HoL Blocking	Nessun blocco	Nessun head-of-line blocking

Wireshark: Setup e Metodologia

Tre catture mirate per osservare in pratica i concetti teorici del protocollo QUIC

01 Analisi dell'Handshake

```
quic.long
```

Cattura del traffico verso un endpoint QUIC (YouTube/Google). Isola i pacchetti Initial e Handshake per osservare le fasi 0-RTT / 1-RTT.

02 Stream Multiplexing

```
quic && ip.addr == <IP>
```

Cattura isolata su una singola connessione QUIC per osservare più stream applicativi multiplexati sullo stesso canale cifrato.

03 Confronto con TCP

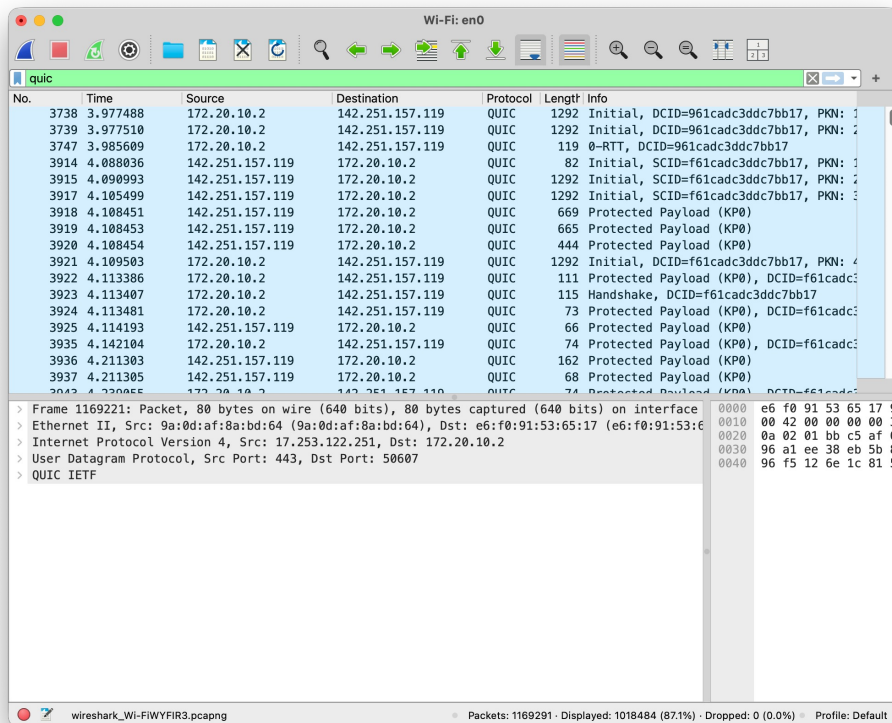
```
tcp.port == 443 && !quic
```

Cattura su un endpoint che non supporta ancora HTTP/3, per confrontare il three-way handshake TCP con l'handshake QUIC.

Strumenti: Wireshark 4.x su macOS · traffico generato con Chrome (YouTube / Google per QUIC)

Analisi dell'Handshake QUIC

Cattura 1: filtro quic, sequenza completa Initial → 0-RTT → Handshake → Protected Payload



No.	Time	Source	Destination	Protocol	Length	Info
3738	3.977488	172.20.10.2	142.251.157.119	QUIC	1292	Initial, DCID=961cad3ddc7bb17, PKN: 1
3739	3.977510	172.20.10.2	142.251.157.119	QUIC	1292	Initial, DCID=961cad3ddc7bb17, PKN: 2
3747	3.985609	172.20.10.2	142.251.157.119	QUIC	119	0-RTT, DCID=961cad3ddc7bb17
3914	4.088036	142.251.157.119	172.20.10.2	QUIC	82	Initial, SCID=f61cad3ddc7bb17, PKN: 1
3915	4.090993	142.251.157.119	172.20.10.2	QUIC	1292	Initial, SCID=f61cad3ddc7bb17, PKN: 2
3917	4.105499	142.251.157.119	172.20.10.2	QUIC	1292	Initial, SCID=f61cad3ddc7bb17, PKN: 3
3918	4.108451	142.251.157.119	172.20.10.2	QUIC	669	Protected Payload (KP0)
3919	4.108453	142.251.157.119	172.20.10.2	QUIC	665	Protected Payload (KP0)
3920	4.108454	142.251.157.119	172.20.10.2	QUIC	444	Protected Payload (KP0)
3921	4.109503	172.20.10.2	142.251.157.119	QUIC	1292	Initial, DCID=f61cad3ddc7bb17, PKN: 4
3922	4.113386	172.20.10.2	142.251.157.119	QUIC	111	Protected Payload (KP0), DCID=f61cad3
3923	4.113407	172.20.10.2	142.251.157.119	QUIC	115	Handshake, DCID=f61cad3ddc7bb17
3924	4.113481	172.20.10.2	142.251.157.119	QUIC	73	Protected Payload (KP0), DCID=f61cad3
3925	4.114193	142.251.157.119	172.20.10.2	QUIC	66	Protected Payload (KP0)
3935	4.142104	172.20.10.2	142.251.157.119	QUIC	74	Protected Payload (KP0), DCID=f61cad3
3936	4.211303	142.251.157.119	172.20.10.2	QUIC	162	Protected Payload (KP0)
3937	4.211305	142.251.157.119	172.20.10.2	QUIC	68	Protected Payload (KP0)

> Frame 1169221: Packet, 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface
> Ethernet II, Src: 9a:0d:af:8a:bd:64 (9a:0d:af:8a:bd:64), Dst: e6:f0:91:53:65:17 (e6:f0:91:53:65:17)
> Internet Protocol Version 4, Src: 172.253.122.251, Dst: 172.20.10.2
> User Datagram Protocol, Src Port: 443, Dst Port: 50607
> QUIC IETF

1 Initial (pkt 3738-3739)

Client → server: avvio della connessione, DCID 961cad3ddc7bb17.

2 0-RTT (pkt 3747)

Il client invia dati applicativi prima ancora del completamento dell'handshake.

3 Handshake (pkt 3923)

Completamento della negoziazione TLS 1.3 integrata nel protocollo.

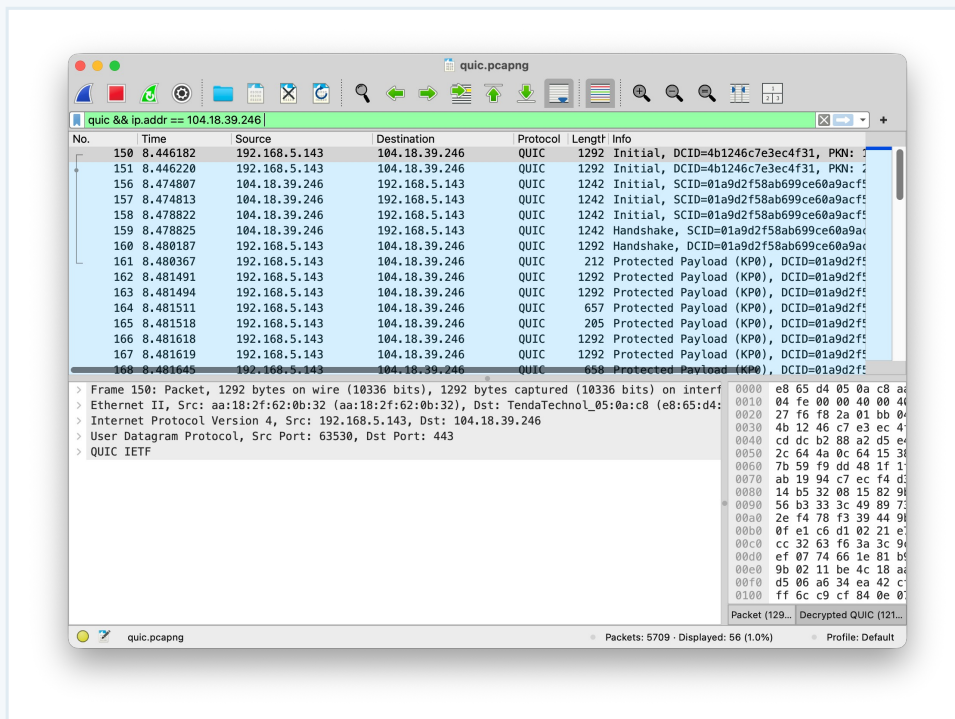
4 Protected Payload (KP0)

Da qui in poi il traffico è cifrato a regime: opaco a qualsiasi osservatore.

DCID 961cad3ddc7bb17: sequenza handshake catturata con filtro "quic"

Stream Multiplexing nella Cattura

Cattura 2: filtro quic && ip.addr == <IP>, connessione isolata verso un singolo endpoint



Stessa connessione (DCID 01a9d2f5...), pacchetti Protected Payload di lunghezza molto variabile

1 Lunghezze variabili

212, 1292, 657, 205 byte sullo stesso canale:
indizio visivo di stream diversi (dati, metadati)
multiplexati senza connessioni separate.

2 Payload cifrato

Il contenuto reale non è leggibile: Wireshark
segnala "Failed to create decryption context" senza
le chiavi di sessione (TLS 1.3).

*Estensione possibile: con SSLKEYLOGFILE impostato
lato browser, Wireshark può decifrare anche il
payload 1-RTT e mostrare gli header HTTP/3 reali.*

Confronto con una Cattura TCP

Cattura 3: filtro `tcp.port == 443 && !quic`, endpoint senza supporto HTTP/3

No.	Time	Source	Destination	Protocol	Length	Info
142	8.309463	192.168.5.143	52.22.168.36	TLSv1...	826	Application Data
143	8.429713	52.22.168.36	192.168.5.143	TCP	66	443 → 58496 [ACK] Seq=4382 Ack=1974 W...
144	8.429716	52.22.168.36	192.168.5.143	TLSv1...	245	Application Data
145	8.429718	52.22.168.36	192.168.5.143	TLSv1...	128	Application Data
146	8.429719	52.22.168.36	192.168.5.143	TLSv1...	97	Application Data
147	8.429877	192.168.5.143	52.22.168.36	TCP	66	58496 → 443 [ACK] Seq=2826 Ack=4623 W...
148	8.429927	192.168.5.143	52.22.168.36	TCP	66	58496 → 443 [ACK] Seq=2826 Ack=4654 W...
149	8.430095	192.168.5.143	52.22.168.36	TLSv1...	97	Application Data
152	8.454457	52.22.168.36	192.168.5.143	TLSv1...	490	Application Data
153	8.454459	52.22.168.36	192.168.5.143	TLSv1...	97	Application Data
154	8.454603	192.168.5.143	52.22.168.36	TCP	66	58496 → 443 [ACK] Seq=2857 Ack=5109 W...
175	8.508991	192.168.5.143	18.65.82.125	TCP	78	58497 → 443 [SYN, ECE, CWR] Seq=0 Win...
188	8.532584	18.65.82.125	192.168.5.143	TCP	74	443 → 58497 [SYN, ACK, ECE] Seq=0 Ack...
189	8.532811	192.168.5.143	18.65.82.125	TCP	66	58497 → 443 [ACK] Seq=1 Ack=1 Win=131...
190	8.533089	192.168.5.143	18.65.82.125	TLSv1...	583	Client Hello (SNI=mcp.figma.com)

> Frame 149: Packet, 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface en0, ...
> Ethernet II, Src: aa:18:2f:62:0b:32 (aa:18:2f:62:0b:32), Dst: TendaTechnol_05:0a:c8 (e8:65:d4:...)
> Internet Protocol Version 4, Src: 192.168.5.143, Dst: 52.22.168.36
> Transmission Control Protocol, Src Port: 58496, Dst Port: 443, Seq: 2826, Ack: 4654, Len: 31
> Transport Layer Security

Three-way handshake TCP seguito da Client Hello TLS separato (SNI=mcp.figma.com)

1 Three-way handshake

SYN → SYN-ACK → ACK, con negoziazione ECN (flag ECE/CWR) prima di qualunque dato.

2 TLS handshake separato

Il Client Hello parte solo dopo che il TCP handshake è completo: un round-trip in più.

3 RTT complessivo

TCP + TLS richiedono ~2 RTT minimo prima dei dati; QUIC li fonde nel proprio Initial.

2 RTT vs 1 RTT

TCP+TLS vs QUIC, dati osservati in cattura

Osservazioni Conclusive sulla Cattura

Sintesi critica dei dati raccolti nelle tre catture Wireshark

1

Divario di RTT confermato

Le catture mostrano concretamente la differenza tra i 2 RTT necessari a TCP+TLS e l'unico round-trip (o zero, in 0-RTT) di QUIC.

2

Payload sempre opaco

In assenza delle chiavi di sessione, ogni pacchetto Protected Payload resta illeggibile: conferma pratica della cifratura pervasiva vista in teoria.

3

Multiplexing dedotto per via indiretta

Senza decifrazione, la variabilità delle lunghezze dei pacchetti è la prova disponibile del multiplexing su singola connessione.

4

Estensione futura

Catturando con SSLKEYLOGFILE attivo si potrebbe decifrare anche il traffico 1-RTT e osservare direttamente gli header HTTP/3 per stream.

Stato Attuale di Adozione

~21%

del traffico web globale
su HTTP/3 (in plateau)

Cloudflare Radar, 2026

100%

dei principali browser
supportano HTTP/3

Chrome, Firefox, Safari, Edge

9 anni

di sviluppo
prima della standardizzazione

2012 → RFC 9000 (2021)

Chi lo usa in produzione

Google & YouTube

QUIC in produzione dal 2013; gran parte del traffico Google è QUIC.
HTTP/3 è il default per YouTube su Chrome.

Cloudflare

Storico sostenitore di HTTP/3, tra i maggiori contributori alla
standardizzazione; quota in calo dal 2023.

Meta (Facebook)

Usa MVFST, la propria implementazione open-source di QUIC, per
servire miliardi di utenti su scala globale.

Akamai & CDN

Le principali CDN supportano HTTP/3 per ridurre la latenza e
migliorare la UX su reti mobile.

Sfida attuale: firewall aziendali che bloccano UDP/443, e un plateau 2026 dell'adozione HTTP/3 a favore di HTTP/2.

Conclusioni

QUIC non è un semplice aggiornamento di TCP, ma una rearchitettura profonda del trasporto web moderno.

Risultati chiave

Eliminazione del head-of-line blocking, handshake 0-RTT, migrazione di connessione e TLS 1.3 integrato risolvono i limiti fondamentali di TCP nel web moderno.

Sfida principale

La sfida non è tecnica ma infrastrutturale: alcuni firewall bloccano UDP/443, rallentando l'adozione enterprise. La soluzione è in corso con policy di rete aggiornate.

Sviluppi futuri

WebTransport, DNS over QUIC (DoQ), QUIC multipath, applicazioni IoT e real-time. Il protocollo si sta espandendo ben oltre l'ambito HTTP.