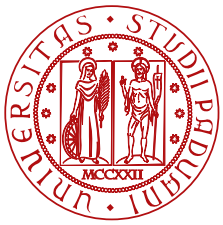

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Impatto del Protocollo HTTP/3 sull'Architettura e sulla Sicurezza dei Sistemi

Relatore
Prof. Zingirian Nicola

Laureando
Rotondo Alessandro

ANNO ACCADEMICO 2023-2024

Data di laurea 12/11/2024

”The Web does not just connect machines, it connects people”

~ Tim Berners-Lee

Sommario

HTTP/3 rappresenta un punto di svolta, non solo tecnico, ma anche strategico, nel panorama delle comunicazioni web, portando con sé una sostituzione progressiva del tradizionale protocollo TCP con QUIC, un sistema pensato, sviluppato e spinto da Google. Questo nuovo standard allontana i sistemi operativi da una gestione centralizzata delle connessioni di rete, fino a oggi garantita dall'implementazione nativa di TCP in Linux e Windows, rendendo sempre più autonomo e dominante lo stack di rete proprietario di Google, presente in tutte le principali piattaforme di navigazione (Chrome, YouTube, Google Suite, ecc.).

Questa tesi esplora a fondo le trasformazioni architetturali indotte dall'adozione di HTTP/3 nei moderni sistemi informatici, analizzando come i modelli tradizionali di trasporto dati e sicurezza vengano progressivamente bypassati e, in alcuni casi, impoveriti. Con l'integrazione nativa di TLS direttamente in QUIC, il protocollo HTTP/3 rivendica il primato della sicurezza e dell'efficienza, ma lo fa riducendo il ruolo dei sistemi operativi tradizionali a semplici "passanti" che trasmettono i dati preparati da librerie esterne.

Attraverso un'analisi tecnica e una valutazione delle implicazioni per gli sviluppatori e gli amministratori di rete, questa tesi analizza i vantaggi, ma anche i potenziali rischi, di una rete in cui le logiche di comunicazione sono plasmate dalle concentrazioni rappresentate dalle "Big Tech". La progressiva diffusione di HTTP/3, promossa dalle risorse e dall'influenza di Google, solleva interrogativi su chi realmente controlli l'infrastruttura di Internet e sui possibili impatti a lungo termine per la privacy, la sicurezza e la governance della rete.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La costante evoluzione e l'espansione di Internet sono state storicamente il motore dello sviluppo di protocolli di rete sempre più efficienti, in grado di offrire connessioni rapide, affidabili e sicure. Negli ultimi anni, con il crescente successo delle applicazioni "web-based", l'efficienza dei protocolli di trasporto è diventata un elemento cardine per ottimizzare l'esperienza d'uso. In questo contesto di trasformazione continua, l'*Hypertext Transfer Protocol version 3* (HTTP/3) rappresenta la più recente evoluzione nel campo delle comunicazioni di rete, introducendo notevoli miglioramenti in termini di velocità, sicurezza e riduzione della latenza.

Uno degli aspetti chiave di HTTP/3 è il passaggio dal tradizionale protocollo di trasporto TCP (Transmission Control Protocol) a QUIC (Quick UDP Internet Connections), basato su UDP (User Datagram Protocol). Questa scelta non è casuale: QUIC, sviluppato originariamente da Google nel 2012 come alternativa sperimentale al modello TCP/IP, rispondeva già allora a molte delle limitazioni tipiche dei protocolli preesistenti. Nel 2015, l'Internet Engineering Task Force (IETF) ha avviato il processo di standardizzazione di QUIC, concludendolo nel maggio 2021 con il rilascio dell'RFC 9000 [1].

I protocolli HTTP/3 e QUIC sono stati ideati per superare le limitazioni delle versioni precedenti di HTTP, tra cui la scarsa flessibilità nella gestione di connessioni parallele, l'inefficienza dell'handshake e il problema dell'*Head of Line Blocking*. Queste criticità penalizzano le performance, in particolare in condizioni di elevata latenza, larghezza di banda ridotta o alto tasso di perdita dei pacchetti. HTTP/3, sfruttando le caratteristiche avanzate di QUIC, riesce a migliorare notevolmente l'esperienza utente, ottimizzando i tempi di connessione, la sicurezza dei dati e le prestazioni di rete.

Questi miglioramenti sono il risultato di innovazioni significative nel protocollo QUIC, che include funzionalità come la ripresa delle connessioni con 0-RTT handshake, la capacità di migrare connessioni senza interruzioni e una gestione avanzata delle connessioni parallele tramite un sistema di multiplexing complesso. Un'altra novità importante è l'integrazione del proto-

collo TLS 1.3 (Transport Layer Security) per garantire la sicurezza di tutte le comunicazioni effettuate tramite QUIC.

Questa tesi esplora l'impatto di HTTP/3 e QUIC sull'architettura e sulla sicurezza dei sistemi informatici moderni. Verranno analizzate le differenze rispetto alle precedenti versioni di HTTP e i benefici in termini di prestazioni e sicurezza offerti da queste nuove tecnologie. Saranno inoltre discussi nel dettaglio i meccanismi di connessione QUIC e i cambiamenti nelle responsabilità dei sistemi operativi con l'introduzione delle connessioni basate su UDP. Per semplificare l'analisi, tutte le connessioni esaminate saranno strutturate secondo il modello *Client-Server*, dato che le altre topologie di rete possono essere ricondotte a questo modello di base, riducendo così la complessità. Si specifica inoltre che tutti i dettagli implementativi di QUIC trattati si riferiscono alla versione 1 del protocollo, escludendo i cambiamenti minori introdotti nel RFC 9369 [2].

Capitolo 2

I Protocolli di Rete

2.1 Lo Stack di Rete: Divisione in Strati e Protocolli associati ad ognuno di essi

Quando si parla di ”stack di rete” ci si riferisce all’insieme dei protocolli che cooperano per permettere lo scambio di dati tra due o più sistemi. Si fa riferimento ad una pila, *stack* in inglese, perché i dati che devono essere trasmessi vengono elaborati in modo sequenziale dai diversi protocolli che compongono lo stack stesso.

2.1.1 Il Modello ISO/OSI

La struttura di riferimento per tutti i moderni stack di rete è sicuramente l’Open Systems Interconnection creato dall’International Organization for Standardization, in breve ”ISO/OSI”. Questo modello di tipo concettuale descrive uno standard articolato in 7 livelli che tutti i sistemi possono applicare in modo da poter comunicare tra loro. Ogni livello ha il compito di gestire un particolare aspetto della connessione e può comunicare solamente con i livelli superiormente ed inferiormente adiacenti.

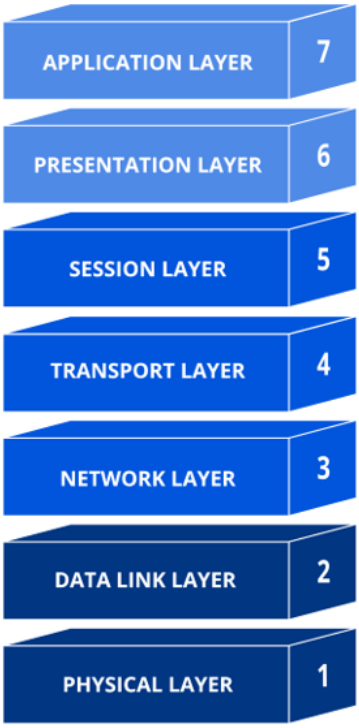

Figura 2.1: Rappresentazione del modello ISO/OSI

Di seguito viene riportata in modo schematico la suddivisione dei diversi aspetti di una connessione tra i livelli del modello OSI:

1. **Physical Layer** : Si occupa del controllo della rete dal punto di vista dei segnali fisici che attraversano i vari mezzi di trasmissione: esso gestisce, ad esempio, aspetti come la forma e la tensione degli impulsi che attraversano un cavo di rame.

2. **Data Link Layer** : Costruisce i frame inserendo i dati in strutture che meglio si adattano al mezzo trasmissivo designato e li invia al livello fisico. Questo livello aggiunge indirizzi fisici, controlla l'accesso al mezzo trasmissivo e corregge gli eventuali errori registrati a livello fisico.

3. **Network Layer** : Questo livello si occupa principalmente dell'instradamento e dell'indirizzamento IP dei pacchetti verso la loro destinazione; questo processo è noto come "routing". Fornisce anche una forma di astrazione dei flussi di rete rispetto a quelli che sono i processi di trasmissione fisica delle informazioni.

4. **Transport Layer** : Implementa i sistemi di segmentazione dei dati prima di passarli al livello inferiore e, allo stesso tempo, permette la ricostruzione degli stessi durante la ricezione. Sempre a questo livello vengono gestiti i flussi dei dati scambiati e il controllo

di eventuali errori registrati durante la comunicazione tramite ”checksum”. Nel caso in cui venissero rilevate delle incongruenze nei dati ricevuti alcuni protocolli che lavorano a questo livello, tra cui TCP, possono richiedere la ritrasmissione dei dati corrotti.

5. **Session Layer** : Ha come obiettivo il controllo delle comunicazioni tra le applicazioni e la gestione delle diverse connessioni che vengono create. È compito del livello di sessione garantire anche la sincronia di invio e ricezione dei messaggi.

6. **Presentation Layer** : Questo livello implementa i meccanismi di compressione, traduzione e crittografia sui dati che verranno utilizzati dal livello applicativo.

7. **Application Layer** : Occupa il livello più alto dell’intero stack e, pertanto, rappresenta l’interfaccia con la rete con cui le applicazioni dell’utente possono interagire. Si precisa che nelle diverse declinazioni del modello OSI le interfacce degli stack non sono limitate al solo livello applicativo ma possono includere alcune funzioni legate, ad esempio, al livello di trasporto.

All’atto pratico, però, non sono mai esistite implementazioni di stack di rete che ricalcano in modo preciso i concetti descritti dal protocollo ISO/OSI. Nella quasi totalità dei casi si è preferito, infatti, implementare soluzioni con un minor numero di livelli per via di alcuni accorpamenti effettuati tra gli stessi: un chiaro esempio di ciò è identificabile nello stack del protocollo TCP/IP, caratterizzato da quattro livelli.

2.1.2 Gli Stack utilizzati dai Protocolli HTTP

Andando ad analizzare le implementazioni degli stack di rete associati ai protocolli HTTP, è possibile osservare che i livelli individuabili hanno degli obiettivi differenti rispetto al modello OSI sopra descritto e, generalmente, si ha una struttura più snella e compatta. Nella figura 2.2 è possibile osservare come il livello inferiore sia comune a tutte le versioni del protocollo HTTP. Questo accade perché il protocollo IP, declinato poi nelle diverse versioni IPv4 ed IPv6, costituisce un’interfaccia univoca tra tutte le apparecchiature connesse ad Internet. Nel diagramma presentato sono stati omessi gli strati inferiori, ovvero quelli riguardanti il livello fisico e il livello data link, perché la loro implementazione dipende strettamente dal tipo di rete in uso e sono meno rilevanti per un’analisi del protocollo HTTP. La parte rilevante per questa analisi è costituita da tutto ciò che si trova sopra il protocollo IP in quanto quest’ultimo può essere considerato come un’interfaccia comune a tutti gli stack di rete moderni.

Salendo di un livello si incontra già la prima differenza introdotta nello stack del protocollo HTTP/3. Se tutte le precedenti versioni utilizzavano il protocollo TCP per il trasporto dei dati, la versione 3.0 di HTTP è caratterizzata dall’utilizzo del protocollo UDP. Questo sostanziale

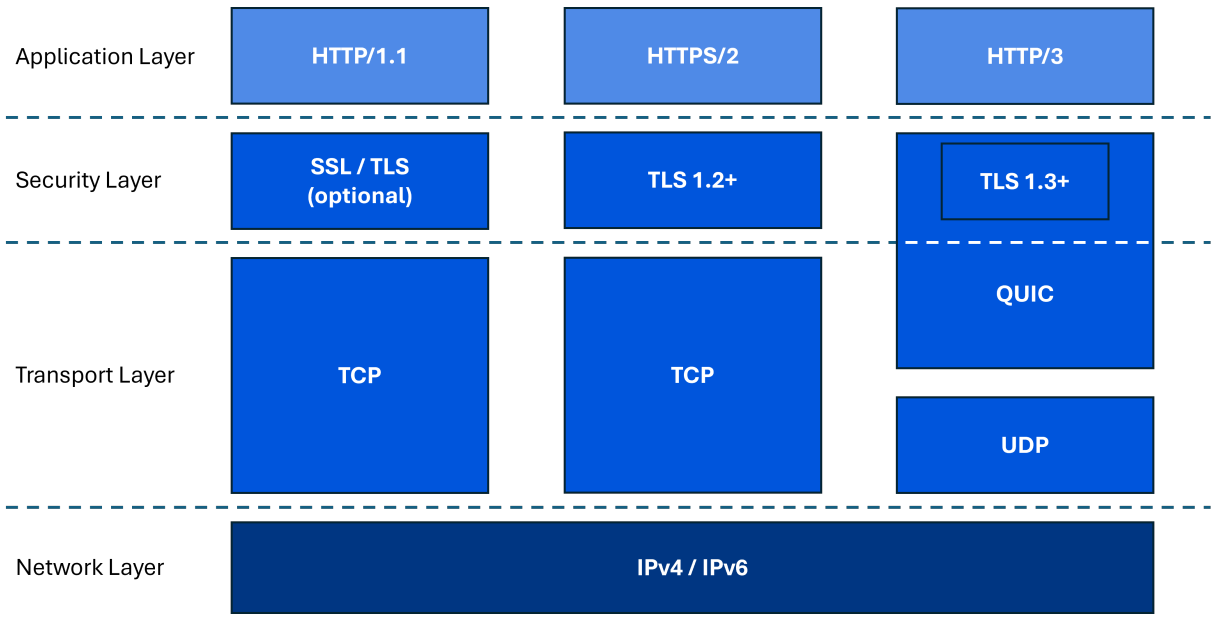

Figura 2.2: Confronto tra gli stack dei protocolli HTTP/1.1, HTTPS/2 e HTTP/3

cambiamento, in favore di un protocollo meno complesso, ha generato la necessità di aumentare gli aspetti che devono essere controllati dal protocollo posizionato superiormente. Nel caso specifico, si ha che il protocollo QUIC coesiste nel livello di trasporto insieme a UDP. In questo modo possono essere garantite una serie di funzioni di gestione dei pacchetti stessi, i cui dettagli sono esposti al capitolo 3. Al contrario del protocollo UDP, quando si utilizza uno stack di rete basato su TCP si ha che tutte le funzioni legate all'ambito del trasporto sono gestite in modo autonomo quest'ultimo.

Salendo ad uno strato superiore si trovano i protocolli che controllano la sicurezza delle connessioni. Negli stack usati dalle precedenti versioni di HTTP, il protocollo legato alla sicurezza veniva inteso come un blocco autonomo: questo ambito viene controllato dal protocollo TLS (Transport Layer Security) oppure, nel caso in cui si usi una versione storica di HTTP, SSL (Secure Sockets Layer). Le principali funzioni legate a questo strato sono: l'autenticazione di un sistema all'interno di una rete, la negoziazione di chiavi crittografiche, la cifratura e la decifratura dei messaggi scambiati. Osservando lo stack di rete utilizzato da HTTP/3 è facile notare come la sicurezza venga gestita in un modo differente: ancora una volta, infatti, è il protocollo QUIC a governare la situazione.

Durante l'implementazione dell'aspetto crittografico all'interno del protocollo QUIC si è preferito utilizzare una nuova architettura: questa prevede, infatti, che tutte le informazioni legate al protocollo TLS vengano comunque incapsulate in pacchetti QUIC e spedite utilizzando la stessa forma di tutti gli altri dati applicativi. Questa scelta ha comportato numerosi vantaggi dal punto di vista della velocità senza però compromettere la sicurezza dei dati: grazie a questo

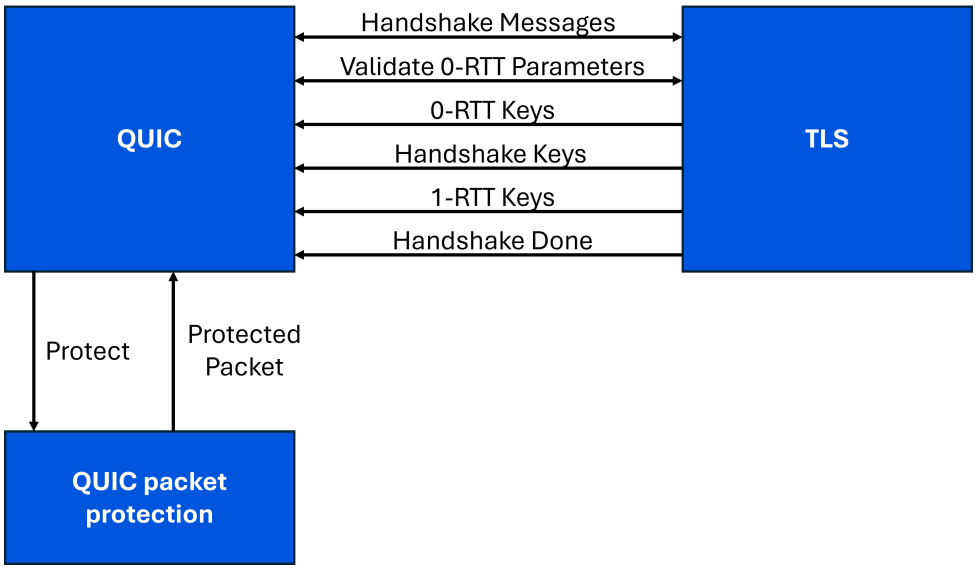

Figura 2.3: Interazioni tra i protocolli QUIC e TLS

cambiamento è stato possibile implementare un processo di handshake che, nel caso peggiore, necessita solamente di un RTT. Nei casi in cui si voglia riaprire una connessione con un server precedentemente contattato il protocollo QUIC mette a disposizione uno specifico tipo di handshake che non richiede alcun tipo di *overhead* temporale (0-RTT handshake). Questi miglioramenti sono da considerarsi estremamente rilevanti in quanto precedentemente, utilizzando uno stack basato su TCP/IP e TLS, erano richiesti tre round-trip per aprire una connessione sicura.

Nel nuovo stack di rete introdotto da HTTP/3 il protocollo TLS può essere interpretato non come un livello superiore rispetto a QUIC ma come una componente che opera nell'ambito del trasporto crittografico. Questo particolare tipo di configurazione si distacca un po' dai paradigmi descritti dal modello OSI ma per una giusta ragione: precedentemente, quando il protocollo di sicurezza occupava uno strato indipendente all'interno della pila di rete, era necessario occupare del tempo prezioso per effettuare un handshake specifico. Grazie alla novità introdotta da QUIC è stato possibile ridurre notevolmente il numero di "round-trip" necessari migliorando in modo sensibile le prestazioni del protocollo stesso. Un secondo vantaggio portato da questo tipo di organizzazione dei protocolli riguarda le strutture dati utilizzate per il trasporto: il pacchetto QUIC rimane l'unità fondamentale per la trasmissione dei dati in modo sicuro. Il protocollo TLS non avvolge ulteriormente i dati ma si limita a fornire le chiavi di cifratura e a gestire le operazioni di negoziazione della sicurezza.

Infine, in cima a tutti gli stack di rete finora descritti, è collocato il protocollo HTTP nelle sue diverse versioni. È importante sottolineare che, nonostante esse forniscano un'interfaccia comune a tutte le applicazioni che ne fanno uso, la scelta della versione di HTTP influenza in modo diretto la scelta del protocollo di trasporto che si andrà ad impiegare. Questo accade

perché', nonostante le più recenti revisioni mantengano la retrocompatibilità con le versioni precedenti, non tutte sono progettate per interagire allo stesso modo con i protocolli di rete sottostanti. Un chiaro esempio di ciò è rappresentato dalla gestione del multiplexing: mentre il protocollo HTTP/2 era direttamente responsabile della gestione di questa funzionalità al livello applicativo utilizzando una singola connessione TCP per più flussi, nella versione successiva il multiplexing è integrato all'interno del protocollo di trasporto QUIC. Questo esempio aiuta a comprendere l'importanza dell'intero ecosistema di protocolli nell'ambito delle connessioni di rete: anche se ad alto livello tutte le versioni di HTTP condividono un'interfaccia simile, i meccanismi di gestione della connessione e controllo dei dati possono differire notevolmente a seconda della revisione che si decide di utilizzare.

2.2 Differenze e Somiglianze tra i protocolli HTTP/2 e HTTP/3

L'introduzione della versione numero 3 per il protocollo HTTP ha portato molte novità in diversi ambiti che riguardano le connessioni di rete. Di seguito sono riportate tutte le maggiori differenze rispetto al protocollo HTTP/2.

2.2.1 Protocollo di Trasporto

Come già accennato al capitolo 2.1, con il protocollo HTTP/3 è stato introdotto un nuovo stack di rete che si basa su connessioni UDP. Questa novità rappresenta un cambio di rotta molto importante in un contesto in cui la grande maggioranza delle connessioni *general purpose* è basata sul protocollo TCP. L'utilizzo di quest'ultimo tipo di connessione comporta diversi vantaggi per quanto riguarda la gestione dei flussi di dati. Il protocollo TCP, infatti, garantisce che i dati trasportati da uno stream siano consegnati in modo continuo ed ordinato, garantendo anche un controllo di integrità tramite *checksum* e un servizio di ritrasmissione dei pacchetti persi. Nonostante i diversi vantaggi, l'uso del protocollo TCP comporta anche alcuni problemi nell'ambito della gestione di connessioni parallele. Rispetto ai primi anni di Internet, quando la maggior parte dei contenuti veniva servito in modo monolitico, nell'ultimo periodo sta diventando sempre più rilevante la capacità di gestire molteplici richieste in modo concorrente. Le limitate potenzialità del protocollo TCP nella creazione e la gestione di molteplici flussi di dati tra due sistemi hanno accentuato una problematica nota come "Head of Line Blocking".

In generale, quando ci si riferisce a questo genere di problematica si parla di un rallentamento della trasmissione di più pacchetti in un singolo stream dovuto a ritardi di diverso genere. L'Head of Line Blocking, o "HoL-blocking" in breve, è stato uno dei problemi principali che si è cercato di mitigare lavorando sulle nuove versioni del protocollo HTTP. Nelle revisioni iniziali del protocollo, tra cui la 1.0 e 1.1, una delle cause scatenanti dell'HoL-blocking poteva essere il raggiungimento del numero massimo di richieste parallele che una determinata applicazione era in grado di gestire. Di conseguenza, le successive richieste erano costrette ad aspettare che le precedenti venissero servite e completate e, di conseguenza, che si liberasse dello spazio nei flussi di trasmissione. In particolare, con l'introduzione del protocollo HTTP/1.1, si era già provato a ridurre questa problematica introducendo la funzione *keep-alive*. Grazie a questo meccanismo è stato possibile mantenere aperta una connessione in modo da poterla sfruttare per più di una richiesta. Prima dell'introduzione del *keep-alive*, infatti, ogni connessione veniva aperta appositamente per servire una singola richiesta ed al termine di essa la connessione veniva chiusa. Grazie a questa novità il protocollo HTTP/1.1 è riuscito ad ottenere prestazioni migliori rispetto alle versioni precedenti limitando l'utilizzo di nuove connessioni che, com'è noto, sono

caratterizzate da una velocità di trasferimento dei dati inferiore negli istanti che seguono la loro creazione.

Nonostante i miglioramenti delle prestazioni introdotti a livello logico, il protocollo HTTP/1.1 continua a non poter gestire richieste multiple simultaneamente senza attendere a conclusione della precedente: ciò è dovuto all'assenza di un sistema di multiplexing. Di conseguenza, non è difficile registrare rallentamenti quando si ha un numero richieste da processare che supera il numero massimo di connessioni gestibili in modo concorrente. Solitamente questo limite si attesta tra le 6 e le 7 connessioni tra un medesimo server e client.

Con l'introduzione del protocollo HTTP/2 viene risolto solo parzialmente il problema dell'HoL-blocking introducendo una prima forma di multiplexing. A partire da questa versione, infatti, una singola connessione TCP viene sfruttata per la trasmissione di molteplici richieste e la ricezione delle relative risposte. Questo traguardo viene raggiunto implementando una distinzione tra i flussi associati ad una richiesta e quelli legati al canale logico di comunicazione con il server. L'introduzione di questo nuovo meccanismo per il multiplexing mitiga ancora una volta il persistente problema di HoL-blocking: questo non può considerarsi ancora risolto in quanto il protocollo TCP continua ad usare per ogni comunicazione un numero esiguo di connessioni e l'invio di nuovi pacchetti rimane subordinato alla ritrasmissione dei pacchetti persi di un precedente invio. Questa limitazione assume un maggior grado di criticità quando si considera lo scarso utilizzo della banda a disposizione del server: nella maggior parte dei casi, infatti, i dati che devono essere inviati non sono a priori disponibili in modo integrale e necessitano di tempo per essere generati. Per questa ragione risulterebbe più efficiente suddividere la banda disponibile in tanti piccoli flussi autonomi anziché' dedicarne grandi porzioni a poche connessioni che non riescono a sfruttarli a pieno.

Un ulteriore motivo per cui si presenta la problematica dell'Head of Line Blocking è identificabile all'interno degli algoritmi di protezione delle comunicazioni utilizzati da TCP. Ogni qual volta vengano rilevate delle perdite consistenti di dati all'interno di un determinato flusso, TCP blocca completamente la trasmissione di nuovi dati rimanendo in attesa di una corretta ritrasmissione. Così facendo però le nuove informazioni, pronte per essere instradate verso il destinatario, sono costrette ad aspettare: anche in questo caso l'attesa è dovuta ad un HoL-blocking.

Uno dei punti chiave che contraddistinguono il protocollo HTTP/3, ed in particolare il livello di trasporto gestito da QUIC, è identificabile nella capacità di gestire un numero estremamente elevato di connessioni in modo concorrente. Grazie a questa caratteristica il problema dell'HoL-blocking viene risolto in quanto nessun pacchetto rimarrà in attesa che un flusso di rete si liberi considerata l'alta disponibilità di essi l'indipendenza tra i flussi stessi. Nel modello proposto da QUIC, a differenza di ciò che accade con TCP, i diversi stream non sono soggetti ad interruzioni nell'invio dei dati a livello di trasporto. Maggiori dettagli riguardanti il meccanismo di

multiplexing implementato da QUIC sono disponibili nella sezione successiva.

2.2.2 Multiplexing

Con il termine ”multiplexing” si intende la tecnica di trasmissione per cui diversi segnali analogici o flussi digitali vengono combinati per essere trasmessi simultaneamente sul medesimo mezzo. Nello specifico, quando si fa rifermento a questo processo nell'ambito delle connessioni di rete, si intende la capacità di un protocollo di combinare diversi flussi di dati del livello applicativo in un singolo ”stream” a livello di trasporto. Come già accennato in precedenza, questa tecnica è stata implementata già dal protocollo HTTP/2 in una prima forma. Questa versione del protocollo HTTP è in grado di instradare molte richieste differenti utilizzando un numero piuttosto limitato di flussi TCP tra client e server; per questo motivo è possibile affermare che HTTP/2 utilizza un sistema di multiplexing a livello applicativo.

Nell'ambito del multiplexing La differenza principale che contraddistingue il protocollo HTTP/3 è individuabile nel numero di connessioni parallele che possono essere gestite. Il protocollo QUIC mette a disposizione fino a 8 byte all'interno degli header dei pacchetti per identificare a quale connessione appartenga un determinato dato. In questo modo diventa possibile indicizzare un numero di connessioni pari a $2^{64-1} \sim 1.8 \cdot 10^{19}$. Avendo a disposizione un numero così elevato di connessioni viene eliminato, all'atto pratico, il problema dell'Head of Line Blocking descritto nel paragrafo precedente. In aggiunta a ciò, all'interno di un pacchetto del protocollo QUIC è possibile inserire unità di dato, dette ”frame”, che possono appartenere a flussi distinti ma con un destinatario comune. Si rimanda al capitolo 3.2 per una descrizione approfondita dei pacchetti utilizzati da QUIC e l'organizzazione dei dati al loro interno.

La gestione di un numero potenzialmente molto elevato di connessioni e di flussi di dati deve essere necessariamente accompagnata da un robusto sistema di gestione delle priorità per tutti i dati in entrata ed in uscita. La gestione delle priorità a livello applicativo è descritta al capitolo 3.3 mentre, per quanto riguarda il livello di trasporto, i dettagli implementativi possono variare a seconda della libreria in uso. Tra i diversi sistemi in uso per la gestione della *Stream Prioritization* è possibile individuare:

- **Round Robin** : le richieste vengono processate con un ordine ”circolare” dedicando ad ogniuna un intervallo di tempo fissato. Questo algoritmo può essere calibrato agendo sulla lunghezza della finestra temporale in cui ogni richiesta viene servita.
- **First Come First Serve** : le richieste vengono servite in modo completo seguendo l'ordine in cui esse sono state generate.

Nella maggior parte delle librerie, a prescindere dall'algoritmo di gestione delle priorità in uso, i dati che non raggiungono la destinazione e necessitano di una ritrasmissione vengono

trattati in modo prioritario aggirando, di fatto, i sistemi di gestione dei flussi multipli sopra citati.

Le capacità di multiplexing migliorate hanno permesso al protocollo QUIC di accentuare anche l'utilizzo del meccanismo *server push*. Grazie a questo sistema, infatti, un server è in grado di inviare risorse ad un client prima che esso le richieda esplicitamente: i maggiori ambiti di applicazione riguardano, ad esempio, i file `.css` che definiscono gli stili oppure risorse di carattere multimediale contenute all'interno della pagina richiesta. Il meccanismo di *server push* non rappresenta una novità introdotta con HTTP/3 in quanto esso risulta già presente tra le funzioni supportate dalla versione 2 dello stesso protocollo. Grazie ai miglioramenti dal punto di vista del multiplexing, uniti all'utilizzo di flussi con bassa priorità, il protocollo QUIC riesce a sfruttare il *server push* in modo ottimale e concreto.

2.2.3 Creazione delle Connessioni

Come già descritto al paragrafo precedente, uno dei grandi vantaggi riscontrabili durante l'utilizzo del protocollo HTTP/3 riguarda la velocità in fase di handshake. Questi miglioramenti sono dovuti per la maggior parte all'introduzione di un nuovo protocollo che gestisce il livello di trasporto, noto come QUIC, in grado di integrare anche le funzioni legate alla sicurezza. Grazie alla nuova configurazione dello stack di rete introdotta con il protocollo HTTP/3, infatti, è stato possibile combinare i processi di configurazione di trasporto e sicurezza in un unico passaggio.

Quando si utilizza il protocollo HTTP/2 il processo di handshake necessario per aprire una connessione sicura è suddiviso nei seguenti passaggi:

1. Il protocollo TCP inizializza lo scambio di dati tra due sistemi grazie ad un "3-way handshake": sono necessari, infatti, 3 messaggi per aprire una connessione in modo corretto. All'interno di questo processo si ha una prima comunicazione di inizio handshake proveniente dal client, SYN, seguita dalla risposta del server, nota come SYN + ACK. L'ultimo passaggio include un ulteriore risposta dal client, costituita da un ulteriore ACK. Con l'acronimo SYN si indica un Synchronize sequence Number, utilizzato per condividere informazioni legate alla connessione che si desidera creare tra i due sistemi.

2. Il protocollo di sicurezza in uso, solitamente TLS 1.2 o seguenti, impegna la connessione per i successivi 2 RTT. Questo tempo viene utilizzato per lo scambio di segreti tra gli estremi della connessione creata in modo da poter criptare le comunicazioni successive.

Sommando i tempi necessari per l'apertura di una connessione si ottiene che, nel caso migliore, sono necessari 3.5 RTT per completare un processo di handshake utilizzando i protocolli dello stack HTTP/2.

Come descritto al capitolo 3.1, per instaurare una connessione tramite il protocollo QUIC è sufficiente il tempo di un singolo RTT. Le applicazioni possono iniziare a scambiare dati in modo protetto dopo questo breve periodo di tempo, nonostante il processo di handshake non si sia ancora stato portato a termine. In aggiunta a ciò, nel caso in cui un client provi a connettersi ad un server precedentemente utilizzato, i dati necessari alla creazione di una connessione possono essere inserti all'interno dei pacchetti che trasportano la prima richiesta ottenendo uno 0-RTT handshake. Si precisa che, sebbene questo tipo di connessione riduca sensibilmente i tempi di attesa prima dell'invio di dati applicativi, questo sistema introduce alcuni rischi dal punto di vista della sicurezza. Per questo motivo viene utilizzato principalmente per connessioni ripetute, dove questi rischi possono essere gestiti nel modo migliore.

La diminuzione dei tempi di apertura di una connessione possono portare benefici in tutti gli ambienti di utilizzo del protocollo HTTP/3. Quando si lavora con connessioni caratterizzate da un'elevata larghezza di banda risulta molto importante ridurre al minimo i tempi in cui non è possibile inviare dati in modo da non sprecare risorse di rete potenzialmente importanti. Al contrario, invece, quando si lavora con connessioni caratterizzate da una latenza particolarmente elevata, la riduzione del numero di RTT necessari per creare un nuovo flusso di rete permette di anticipare di molto la trasmissione di dati applicativi.

Per ragioni di natura fisica non è possibile diminuire il tempo di viaggio di un dato sul mezzo di trasmissione in uso. Di conseguenza, in un contesto in cui il valore della latenza è fissato risulta particolarmente importante tentare di ridurre il fattore moltiplicativo legato a questo intervallo di tempo. È per questo motivo che la riduzione del numero di "round trip" utilizzati nella fase di apertura di una connessione assume un ruolo centrale per i nuovi protocolli di comunicazione. Si precisa inoltre che, nonostante il nuovo sistema di creazione delle connessioni aiuti a diminuire i tempi di handshake, i problemi di congestione e latenza di una rete vengono mitigati anche da altri aspetti del funzionamento di QUIC come il *congestion control* e il multiplexing su UDP.

2.2.4 Sicurezza e Crittografia

Il protocollo TLS, incaricato della gestione della sicurezza nel trasporto dei dati, occupa un ruolo centrale all'interno dello stack di rete di HTTP/3. Una delle novità introdotte dal protocollo QUIC riguarda l'utilizzo della crittografia per tutti i flussi di rete in uso. A livello protocollare si è deciso che ogni singolo dato, prima di essere immesso nella rete, deve essere necessariamente criptato utilizzando uno dei sistemi di sicurezza previsti all'interno delle specifiche di QUIC. Questo punto rappresenta un'importante differenza rispetto a tutte le precedenti versioni di HTTP in quanto esse comprendevano alcune funzioni di crittografia ma il loro utilizzo non era imposto in nessun modo. Si precisa, inoltre, che non esistono ad oggi protocolli di si-

curezza proprietari per le connessioni HTTP/3. È infatti possibile implementare un livello di sicurezza basato su TLS 1.3 anche durante l'utilizzo di versioni precedenti dello stesso protocollo applicativo in modo da poter fare affidamento sugli algoritmi di crittografia più recenti e aggiornati.

Come già presentato in precedenza, all'interno del protocollo QUIC gli aspetti della sicurezza e della crittografia occupano un ruolo centrale: tutte le comunicazioni, compresa la fase di handshake, devono essere necessariamente protette tramite la crittografia dei dati. Questo è un particolare punto di distacco dal protocollo HTTP/2 il quale, invece, poteva prevedere anche delle comunicazioni non criptate. Il funzionamento dello strato di sicurezza all'interno dello stack di rete del protocollo HTTP/3 è descritto in modo approfondito al capitolo 3.1.

2.2.5 Performance

Da sempre le prestazioni di un protocollo rappresentano un aspetto cruciale da tenere in considerazione durante le fasi sia di creazione che di adozione. Per quanto riguarda HTTP/3 è possibile concludere che, utilizzando una rete con caratteristiche ottimali, le performance non si distacchino di molto rispetto alle precedenti versioni del protocollo (HTTP/2 e HTTP/1.1). È possibile osservare, invece, un miglioramento significativo nelle velocità di scambio dei dati anche quando ci si trova in condizioni di alta latenza o ridotta larghezza di banda. In questi casi la nuova architettura di rete utilizzata dal protocollo QUIC riesce a sfruttare in modo più efficiente le risorse a disposizione e, di conseguenza, ad ottenere prestazioni migliori. È stato possibile ottenere questi risultati grazie all'introduzione dei sistemi di 0-RTT handshake e dell'ottimizzazione del processo di recupero rapido delle perdite (loss recovery).

Al contrario, invece, quando ci si trova a lavorare con connessioni di rete dotate di un basso livello di latenza ed una elevata velocità di trasferimento, si potrebbero registrare performance migliori utilizzando la consolidata architettura basata su TCP/IP. Ovviamente, non è possibile generalizzare questo argomento facendo delle distinzioni solamente riguardo alcune caratteristiche delle connessioni di rete in uso: molto spesso le prestazioni di un flusso di dati dipendono anche dalla struttura dei dati stessi. Per le motivazioni esposte ai paragrafi precedenti è possibile affermare che le connessioni basate sul protocollo QUIC riescano a adattarsi in modo generalmente migliore in base al contesto in cui vengono impiegate.

Un ulteriore fattore di variabilità che può impattare notevolmente sulle prestazioni di una connessione che utilizza QUIC riguarda l'implementazione stessa del protocollo. Quest'ultimo è definito ad alto livello all'interno dei documenti redatti dall'IETF (Internet Engineering Task Force) ma i dettagli implementativi sono stati lasciati agli sviluppatori. Da questa grande libertà deriva un'ottimizzazione solamente parziale e non uniforme degli algoritmi necessari al funzionamento del protocollo stesso; tra questi troviamo: multiplexing, scheduling, flow control,

congestion control, packetization. Si osserva inoltre che, essendo le librerie che implementano QUIC spesso sviluppate da grandi aziende informatiche, ogni implementazione di QUIC potrebbe essere ottimizzata per gestire lo specifico tipo di traffico che interessa i servizi offerti dall'implementatore stesso.

Quando si parla di performance, però, non è corretto riferirsi solamente all'aspetto delle velocità di trasferimento e delle latenze. Un altro elemento molto importate per tutti i "content provider" riguarda l'impatto che un protocollo ha sui sistemi a livello fisico. È stato registrato [3] che l'utilizzo del protocollo QUIC in condizioni di elevato traffico ha comportato ad un sensibile aumento delle risorse fisiche all'interno di un server. Questo fenomeno è perfettamente in accordo con le caratteristiche esposte fino ad ora: tutte le novità introdotte nei protocolli HTTP/3 e QUIC hanno portato ad un generale aumento della complessità dei protocolli stessi. Se è sempre vero che tutto ha un costo, in questo caso i miglioramenti introdotti hanno portato ad un aumento del carico sulle CPU dei sistemi che controllano le reti.

Nonostante l'impiego del protocollo HTTP/3 possa risultare più "costoso" ai gestori delle reti, sicuramente i vantaggi introdotti superano di gran lunga gli inconvenienti legati all'aumento del carico di lavoro sui server. Questo risulta evidente quando si considera che due tra i maggiori provider di servizi online, Google e Cloudflare, stanno continuando a investire molto su questa nuova tecnologia e già da alcuni anni hanno integrato la compatibilità con il protocollo HTTP/3 all'interno di tutti i loro sistemi.

2.2.6 Loss Recovery

Durante il processo di trasporto delle informazioni su di un mezzo fisico è possibile che alcuni pacchetti non raggiungano la destinazione oppure che qualche dato venga corrotto e, di conseguenza, che non possa essere utilizzato correttamente da chi lo riceve. In questi casi è necessario procedere ad informare il mittente della necessità di ritrasmettere le informazioni perse e attendere nuovamente il tempo di viaggio del dato stesso. La funzione di recupero dei pacchetti persi è solitamente affidata al protocollo che si occupa del livello di trasporto all'interno di una connessione di rete.

Per quanto riguarda il protocollo HTTP/2 e, in particolare, le connessioni basate su TCP/IP, i pacchetti vengono trasmessi in modo ordinato e progressivo: al destinatario dei dati viene assegnato il compito di notificarne la corretta ricezione tramite un messaggio di "acknowledgement" (ACK in breve). Questo sistema, seppur semplice, può talvolta incepparsi a causa di problematiche legate alla latenza della connessione. In condizioni di alto RTT è possibile che venga ritardata la trasmissione di alcuni pacchetti in modo da limitare il livello di congestione all'interno della rete. Questo sistema produce come effetto collaterale una momentanea dimi-

nuzione del throughput della rete nel tentativo di ridurre il numero dei pacchetti che necessitano di ritrasmissione.

Quando si utilizza il protocollo UDP per il trasporto delle informazioni non si ha a disposizione alcun sistema di controllo delle perdite: questo è uno degli aspetti critici a cui provvede il protocollo QUIC introducendo un sistema di *loss recovery* personalizzato. All'interno dell'header di ogni pacchetto spedito viene inserito un numero identificativo univoco e monotonamente crescente, denominato "packet number". Tramite una semplice analisi il destinatario dei pacchetti riesce a rilevare eventuali incongruenze nella numerazione ed ha la facoltà di richiedere la ritrasmissione dei dati necessari. Si osserva che la numerazione dei pacchetti rimane crescente in ogni momento: anche nel caso di un secondo invio di un pacchetto, nonostante le informazioni in esso contenute non siano cambiate, questo sarà identificato da un nuovo numero. Grazie a questa rigida regola viene anche risolto un problema tipico delle trasmissioni basate su TCP: utilizzando questo protocollo, infatti, non è possibile sapere se un pacchetto ricevuto è frutto di una seconda trasmissione oppure se esso abbia solo impiegato un tempo elevato per raggiungere la destinazione. Questa situazione di ambiguità potrebbe portare ad un'errata gestione della rete dal punto di vista della congestione in quanto le stime sul traffico e la latenza potrebbero diventare imprecise.

La numerazione dei pacchetti risolve anche un secondo problema legato all'utilizzo del protocollo UDP per il trasporto dei dati. Questo sistema non garantisce nativamente la consegna dei diversi Datagram nel medesimo ordine in cui sono stati spediti. Potendo però sfruttare una numerazione sempre crescente, però, il problema dell'ordine viene risolto disponendo i pacchetti secondo il loro identificatore.

2.2.7 Migrazione delle Connessioni

Ancora una volta, uno degli svantaggi riscontrabili durante l'utilizzo del protocollo di trasporto UDP viene sfruttato da QUIC per implementare funzioni aggiuntive. Nel caso specifico si fa riferimento alla caratteristica delle connessioni create di essere "stateless". È possibile utilizzare questa espressione quando si fa riferimento ad una connessione che non porta con sé informazioni riguardo allo stato delle comunicazioni che avvengono attraverso essa. In questi casi l'aspetto di gestione viene affidato ad un protocollo posizionato ad uno strato superiore rispetto al trasporto e spesso queste funzioni sono affidate al livello applicativo. Durante l'utilizzo del protocollo HTTP/3 tutte le connessioni sono stateless e la gestione dei flussi di rete che le attraversano è affidata a QUIC. Si ha che le connessioni stabilite attraverso questo stack di rete vengono identificate tramite un "Connection ID", un numero intero che può occupare fino a 160 bit all'interno dell'header di ogni pacchetto. Questo valore viene condiviso da tutti i pacchetti che appartengono alla stessa connessione e viene dismesso nel caso in cui la comunicazione

venga terminata dal protocollo stesso. La presenza di questo valore che accompagna tutti i dati trasmessi all'interno dello stesso canale rende possibile la migrazione di un flusso di rete nel caso in cui, ad esempio, un dispositivo mobile passi dall'utilizzo di una rete senza fili domestica (connessione Wi-Fi) ad una connessione Internet cellulare (4G o 5G). Si precisa che, nonostante il *Connection ID* rimanga generalmente costante per tutto il periodo di utilizzo di una connessione, il protocollo QUIC prevede dei meccanismi per rinnovare questo valore senza terminare una comunicazione. In questo modo è possibile garantire una maggiore stabilità anche durante le fasi di migrazione di una connessione.

Questo nuovo meccanismo si distacca completamente dai paradigmi imposti nelle connessioni TCP: quando ci si affida a questo protocollo, infatti, le connessioni sono strettamente dipendenti dalla coppia di indirizzi IP del client e del server. Nel caso in cui uno di essi dovesse cambiare il livello di trasporto richiederebbe di effettuare un nuovo handshake. In questo modo si genererebbero ulteriori ritardi nello scambio dei dati dovuti alla riapertura di una nuova connessione e potrebbero essere persi i progressi nelle operazioni di invio e ricezione fino a quel momento effettuate.

Grazie al sistema di migrazione delle connessioni introdotto dal protocollo QUIC è possibile ridurre notevolmente le interruzioni dovute al cambio della rete in uso. Questa caratteristica è particolarmente importante in contesti in cui devono essere trasferite grandi quantità di dati, alcune delle applicazioni più rilevanti riguardano: lo streaming video, le videochiamate o il trasferimento di file di grandi dimensioni tra il proprio dispositivo ed il cloud. Questo sistema, unito alla gestione avanzata degli stream paralleli ed alla capacità di creare nuove connessioni senza impiegare tempo aggiuntivo, permette di ottenere buone prestazioni anche durante l'utilizzo di connessioni senza fili che non eccellono nella stabilità.

Capitolo 3

Funzionamento del Protocollo QUIC

3.1 L'inizializzazione di una Connessione basata su QUIC

L'inizializzazione di una connessione basata sul protocollo HTTP/3 è un processo complesso e caratterizzato da molti passaggi, ognuno con uno scopo differente. Grazie a questo processo vengono scambiati i diversi parametri necessari per la configurazione della connessione stessa.

Il primo passaggio, fondamentale per la comunicazione con un qualsiasi server tramite l'Internet Protocol (IP), riguarda la risoluzione dell'indirizzo del server stesso. Nella maggior parte dei casi, infatti, un client web non ha a disposizione il vero indirizzo IP del server con il quale deve essere instaurata la connessione ma soltanto il nome del sottodominio ad esso associato. In queste situazioni si ricorre l'utilizzo dei sistemi DNS (Domain Name System): effettuando una richiesta ad uno dei server che ospita questo tipo di servizio è possibile ottenere il corretto indirizzo della macchina con cui si vuole comunicare.

Una volta concluso questo primo passaggio perliminare e ottenuto l'indirizzo del server con il quale si vuole comunicare, il passo successivo per aprire una connessione riguarda l'inizializzazione del protocollo di trasporto che si intende impiegare: per quanto riguarda HTTP/3, il protocollo su cui si basa il trasporto di tutte le informazioni è l'UDP. Per propria natura l'User Datagram Protocol non necessita di stabilire una vera e propria connessione tra client e server mediante un processo di handshake. Questo tipo di connessione si occupa soltanto dell'invio e della ricezione di particolari strutture, chiamate "Datagram", all'interno delle quali vengono collocati i dati da trasportare. Il protocollo UDP, a differenza del TCP, non possiede un meccanismo di "acknowledgement" grazie al quale il mittente di un pacchetto ha la certezza che i dati siano stati consegnati correttamente al destinatario. In aggiunta a ciò, quando i pacchetti vengono ricevuti non è garantito che l'ordine delle informazioni in arrivo sia il medesimo utilizzato dal mittente per ragioni dovute al routing IP. Si precisa, inoltre, che UDP non prevede un sistema di ordinamento dei pacchetti e che questa funzione viene solitamente lasciata ai protocolli

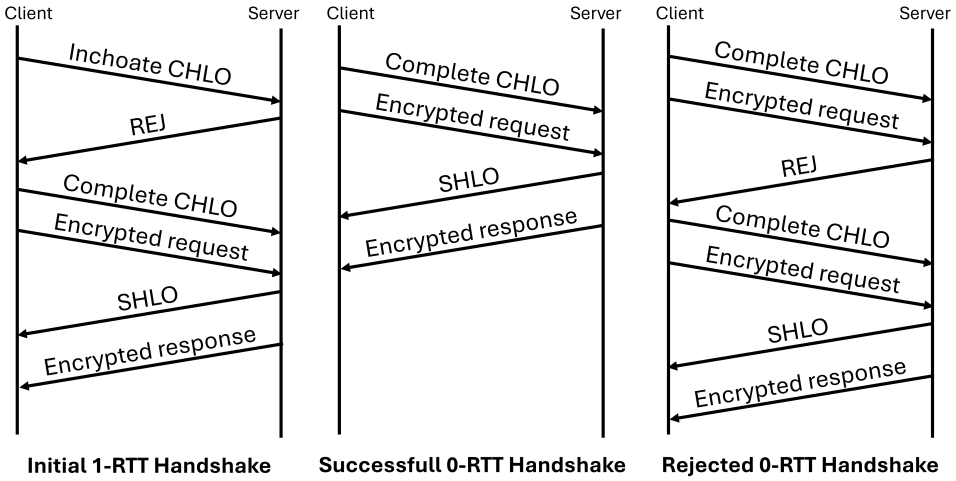

Figura 3.1: Visualizzazione dei processi di handshake previsti nel protocollo QUIC

superiori, come QUIC. Queste problematiche emergono per via dei principi di funzionamento del protocollo UDP: per riuscire a garantire una maggiore velocità e allo stesso tempo ridurre la complessità da un punto di vista computazionale UDP utilizza delle connessioni cosiddette ”stateless”. Queste, a differenza di quanto accade utilizzando il protocollo TCP, sono paragonabili ad un ”tubo vuoto” in quanto non viene mantenuto uno stato delle connessioni stesse durante la trasmissione dei dati.

Le limitazioni del protocollo UDP citate precedentemente possono essere accettabili in determinati ambiti come la trasmissione di video in diretta, ma non sono sostenibili in un protocollo ”general purpose” come HTTP/3. Per questa ragione QUIC, il protocollo che funziona tramite le connessioni UDP, è tenuto a compensare le mancanze dovute al protocollo di trasporto implementando dei nuovi meccanismi per l’ordinamento dei pacchetti e il controllo sia dei dati scambiati che degli errori.

Il passaggio successivo prevede l’inizio del vero e proprio processo di handshake tra client e server. Per garantire un elevato livello di sicurezza e rapidità, lo scambio di dati iniziale del protocollo QUIC combina sia i dati utili all’apertura di una comunicazione dal punto di vista del trasporto sia le informazioni crittografiche relative alla connessione stessa. Nello specifico, nel primo round-trip si ha che il server condivide con il client che sta cercando di aprire una nuova connessione diverse informazioni di base, tra queste si trovano: i dati di configurazione del server, la chiave effimera del server relativa al protocollo Diffie-Hellman, i certificati che autenticano il server stesso e tutte le informazioni ad essi relative, il *source-address token*. Quest’ultimo ”gettone” è costituito da una stringa che contiene in modo criptato l’indirizzo IP del client ed un riferimento temporale; quest’informazione tornerà utile al client per velocizzare i successivi processi di handshake con il medesimo server. Tutte le informazioni presentate in precedenza sono contenute all’interno di un particolare messaggio generato dal server e indicato

con l'etichetta "REJ", abbreviazione di *Reject* (*rifiuto* in italiano). Questo tipo di messaggio è stato inviato in seguito alla ricezione di un "inchoate CHLO", ovvero *inchoate client hello message* (*messaggio di saluto incipiente* in italiano): questo specifico messaggio viene usato da un client quando esso non possiede alcuna informazione riguardo il server a cui si sta provando a connettere. Una volta ottenute le informazioni relative alla configurazione del server designato, il client web può procedere alla verifica dei certificati ottenuti e, solo in seguito, all'invio di un *complete CHLO*. Il contenuto di questo tipo di messaggio differisce dall'*inchoate CHLO* principalmente per una ragione: esso contiene la chiave pubblica effimera associata al client relativa al protocollo di negoziazione di chiavi effimere tramite TLS 1.3. In questo momento, però, il client ha già ricevuto le informazioni necessarie per poter calcolare le chiavi iniziali con cui criptare le prime richieste per il server. Anche se i primi dati dell'applicazione possono già essere stati inviati il processo di handshake non può ancora ritenersi concluso. Se i passaggi precedentemente descritti sono andati a buon fine il server presenterà un ultimo messaggio di tipo "server hello", *SHLO* in breve. All'interno di quest'ultimo è contenuta la chiave effimera associata al server relativa al protocollo Diffie-Hellman. Il messaggio *SHLO* è l'ultima comunicazione tra client e server che viene cifrata utilizzando le chiavi iniziali. Questo accade perché, in seguito alla ricezione del *server hello*, entrambe le parti possiedono le informazioni necessarie per poter calcolare le chiavi effimere definitive del protocollo Diffie-Hellman e tali chiavi saranno usate per criptare tutte le successive trasmissioni di dati. Si precisa che, come riportato anche nella figura 3.1, i messaggi *CHLO* e *SHLO* non costituiscono l'unico contenuto dei pacchetti ad essi associati: si può osservare che, come accennato già in precedenza, il protocollo prevede l'inserimento di richieste e risposte anche in questa fase in modo da minimizzare i tempi di attesa a livello applicativo. I suddetti passaggi sono riassunti anche nell'immagine 3.2. Il significato delle sigle utilizzate è spiegato nella tabella sottostante.

Message	Annotation
CEPri	Client's ephemeral Diffie-Hellman private value
LPri	Server's long-term DH private value
SEPri	Server's ephemeral Diffie-Hellman private value
LPub	Server's long-term DH public value
g	Primitive Root
CEPub	Client's ephemeral Diffie-Hellman public value
SEPub	Server's ephemeral Diffie-Hellman public value
InitKC	Initial key of Client
InitKS	Initial key of Server
FSKC	Forward-secure key of Client
FSKS	Forward-secure key of Server
pkS	Public signature key of the server
skS	Private signature key of the server
{.}skS	{.} is signed using the private signature key of the server
{.}InitKC	{.} is encrypted using the initial key of client
{.}InitKS	{.} is encrypted using the initial key of server
{.}FSKS	{.} is encrypted using the forward-secure key of server

Tabella 3.1: Simboli utilizzati nella figura 3.2

Risulta importante osservare che la crittografia implementata nel protocollo QUIC prevede l'utilizzo di due coppie di chiavi per cifrare i messaggi trasmessi: inizialmente si sono utilizzate delle chiavi basate su valori pubblici legati al server e i valori effimeri proprietari del client. Successivamente, in seguito al completamento del processo di handshake, le chiavi iniziali vengono rimpiazzate con una nuova coppia di chiavi: queste sono state generate a partire dai valori effimeri posseduti da entrambe le parti e pertanto possono essere considerate più sicure rispetto alla coppia iniziale.

Uno dei punti di forza del protocollo QUIC è rappresentato dalla possibilità stabilire una connessione completa utilizzando solo alcuni dei passaggi presentati precedentemente. Esiste infatti un secondo processo di handshake, più rapido rispetto al principale, che permette al client di inviare richieste al server già durante la prima trasmissione di dati: in questo caso si parla di 0-RTT handshake. Questo nome è stato assegnato per mettere in evidenza il tempo necessario per avviare una connessione prima che essa possa essere utilizzata da un'applicazione. Se normalmente è necessario aspettare il tempo di un *round trip*, ovvero il tempo che intercorre tra la trasmissione di un messaggio e la ricezione della relativa risposta, grazie allo 0-RTT handshake è possibile omettere il primo *inchoate CHLO* ed il messaggio *REJ* ad esso associato. Questa riduzione dei messaggi scambiati è applicabile solamente nel caso in cui il client sia già stato connesso in precedenza al server in questione. È facoltà dei client che utilizzano il protocollo QUIC salvare in modo permanente le informazioni contenute nei messaggi *REJ* ricevuti in fase di handshake in modo da poterli riutilizzare qualora si presentasse l'opportunità. Il processo di

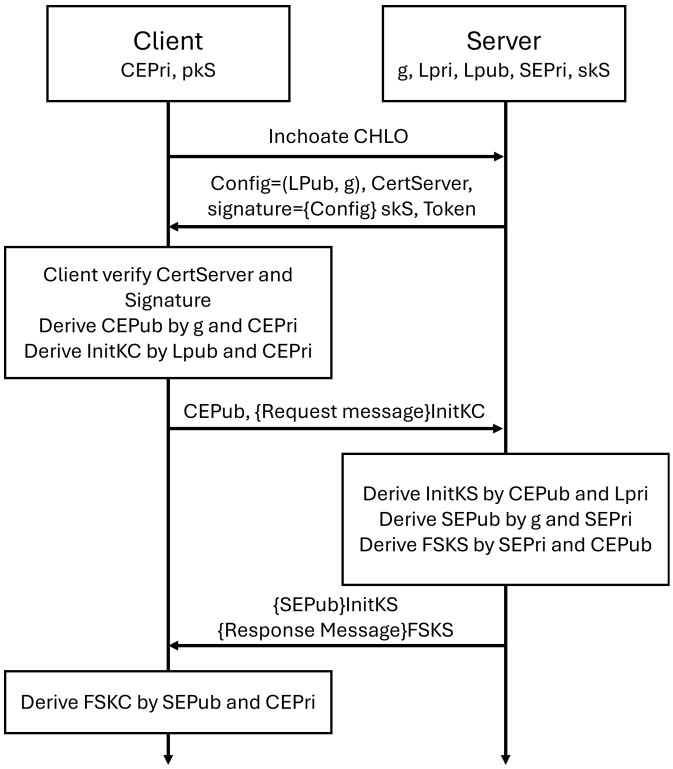

Figura 3.2: Dettaglio delle informazioni scambiate nella fase di handshake

0-RTT handshake può avere due esisti: il server potrebbe accettare la connessione rispondendo con un messaggio *SHLO* seguito dalle risposte alle richieste effettuate oppure la connessione potrebbe essere respinta con un messaggio *REJ*. Quest'ultimo caso si può verificare quando, ad esempio, la configurazione del server è cambiata oppure i certificati di sicurezza ad esso associati sono scaduti. In generale si ha che le informazioni in possesso di un client rimangono valide per una settimana nel caso medio. Si precisa che la stima precedentemente presentata potrebbe subire delle variazioni a seconda della configurazione del server e dalle politiche di sicurezza applicate. Possono esistere contesti in cui il processo di 0-RTT handshake viene completamente disattivato e le informazioni in possesso del client devono essere rinnovate ad ogni connessione.

Qualora una connessione venisse respinta il client può procedere direttamente presentando un nuovo messaggio *CHLO* completo in quanto le nuove informazioni necessarie per comporlo sono già state ricevute con il messaggio *REJ* del server. Si precisa che, come già accennato in precedenza, il processo di 0-RTT handshake non è disponibile in tutte le situazioni: esso può essere compiuto solamente nei casi in cui si sta cercando di aprire una nuova connessione con un server a cui il client era già stato connesso in precedenza e di cui aveva già acquisito le informazioni di configurazione. Questo metodo di connessione deve essere utilizzato con una certa cautela poiché esso rimane vulnerabile a diversi tipi di attacco informatico, tra cui i replay attack.

3.1.1 Taglia Massima di una Risposta e QUIC flood DDos attack

La disponibilità di svolgere uno 0-RTT handshake solleva però delle preoccupazioni dal punto di vista della sicurezza. Per eliminare la possibilità che possano essere messi in atto attacchi del tipo *flood DDos* (Distributed Denial of Service) semplicemente falsificando la propria identità ed impersonando un client noto al server sono state introdotte delle limitazioni. La più rilevante tra queste riguarda la dimensione massima che le risposte possono assumere prima che il processo di handshake venga finalizzato. Questa restrizione, nota come ”three times amplification limit”, è specifica per il periodo pre-handshake ed è stata inserita per prevenire gli attacchi amplificati, ovvero i casi in cui un attaccante potrebbe usare un server per amplificare il traffico verso un target.

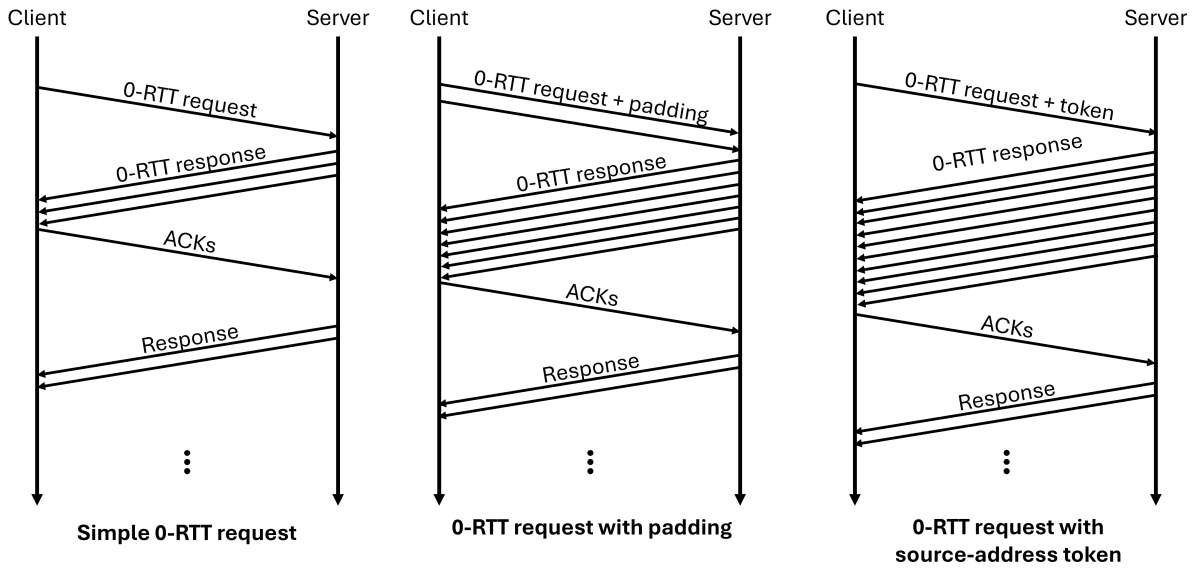

Figura 3.3: Strategie per aumentare la taglia della risposta ad una richiesta con handshake 0-RTT

Nonostante la limitazione imposta dal protocollo esistono dei metodi per sorpassare la soglia dei dati inviabili in modo da rendere ancora più efficaci le connessioni create tramite 0-RTT handshake. Il più semplice da mettere in atto prevede l'invio di dati addizionali in aggiunta alla richiesta in modo da aumentarne la dimensione: così facendo si sfrutta la proporzionalità diretta tra la taglia di richiesta e risposta, di fatto elevando la soglia imposta dal protocollo. Una seconda soluzione prevista dal protocollo QUIC prevede l'inserimento del *source-address token* tra le informazioni fornite nel messaggio *CHLO*. Così facendo il server è in grado di validare l'indirizzo utilizzato dal client e, di conseguenza, può procedere all'invio dei dati ignorando la limitazione imposta dal protocollo stesso. Questo processo non sostituisce un'autenticazione completa ma viene utilizzato per mitigare gli attacchi di *spoofing* dell'indirizzo IP. Entrambe le

soluzioni presentate sono perfettamente legali da un punto di vista protocollare e, pertanto, la decisione di quale approccio applicare è lasciata agli sviluppatori della libreria HTTP/3 in uso.

3.2 Organizzazione dei Dati all'interno dei Pacchetti di QUIC

La suddivisione dei dati in pacchetti è una delle responsabilità principali del protocollo di trasporto in tutti gli stack di rete. Durante il processo di pacchettizzazione i dati vengono suddivisi in parti di ridotte dimensioni, dette per l'appunto "pacchetti", in modo da garantire una gestione più agevole delle operazioni di trasporto sia a livello logico che a livello fisico. Grazie alla suddivisione in pacchetti, infatti, è stato possibile semplificare processi come il controllo degli errori e della congestione all'interno dei flussi di rete.

Analizzando il percorso dei dati attraverso la pila dei protocolli di rete utilizzata dalle connessioni HTTP/3 sono identificabili quattro passaggi fondamentali, ognuno associato ad uno specifico protocollo:

- **QUIC** : In questa prima fase le informazioni subiscono l'alterazione più significativa. Durante questo processo, infatti, i dati vengono criptati e suddivisi in pacchetti. La struttura di quest'ultimi è analizzata nel dettaglio al nei paragrafi 3.2.1 e 3.2.2 . Come è possibile intuire, da questo momento in poi le informazioni possono essere solo considerate come una sequenza ordinata di bit e tutte le operazioni successive vengono effettuate ignorando il contenuto dei dati stessi perchè gli altri protocolli non sono in grado di interpretarli.

- **UDP** : In questo secondo passaggio le informazioni provenienti da QUIC vengono inserite all'interno di una nuova struttura chiamata "Datagram". All'interno di un Datagram possono essere inseriti diversi pacchetti QUIC, accompagnati da un header molto semplice. Questo contiene solamente informazioni riguardanti i numeri delle porte UDP di sorgente e di destinazione ed un dato che indica la lunghezza del payload stesso. La presenza di questi valori è utile per il corretto funzionamento dello stack di rete: questi valori rappresentano un identificatore per i servizi utilizzati negli strati successivi. Si osserva che le connessioni legate ai protocolli HTTP/3 e QUIC utilizzano solitamente la porta UDP numero 443: nonostante questo valore sia utilizzato in molti casi esso non è fissato. Il protocollo QUIC può essere configurato per utilizzare altri numeri di porta.

- **IP** : I Datagram creati al livello di trasporto vengono ulteriormente "avvolti" in un pacchetto controllato dal protocollo IP. Durante questo processo vengono aggiunte ai dati che devono essere instradati tutte le informazioni necessarie per portare a termine le operazioni di trasporto. Tra le numerose informazioni contenute all'interno dell'header del protocollo IP si segnala la presenza di un flag che indica allo strato di rete e ai sistemi sottostanti di non frammentare ulteriormente i pacchetti. La presenza di questa direttiva all'interno di tutte le comunicazioni di QUIC deriva da una scelta effettuata a livello di protocollo: una

prima operazione di frammentazione delle informazioni è già stata effettuata precedentemente ed un'ulteriore suddivisione potrebbe compromettere il funzionamento di alcuni sistemi di controllo del protocollo QUIC. Si precisa che la direttiva "don't fragment" potrebbe essere ignorata da alcuni dispositivi di rete in caso di incompatibilità dei pacchetti con il MTU (Maximum Transmission Unit).

- **Ethernet** : Per l'ultima volta i dati ricevuti dal protocollo sovrastante vengono inseriti all'interno di una nuova unità di trasporto. La struttura dati utilizzata a livello fisico, detta "frame Ethernet", è caratterizzata da un header contenente informazioni utili al trasporto su un mezzo fisico: tra queste si trovano l'indirizzo fisico sia del mittente che del destinatario oltre ad un'indicazione sul protocollo usato al livello superiore, i cui dati sono incapsulati nel payload. Nella parte finale del frame Ethernet vengono anche inseriti 4 byte che contengono le informazioni relative al controllo di parità CRC (Cyclic Redundancy Check).

All'interno dei pacchetti utilizzati dal protocollo QUIC si possono distinguere due sezioni principali: "header" e "frames". Di seguito si analizza in dettaglio struttura e contenuti di entrambe le parti.

3.2.1 Header in un pacchetto QUIC

I pacchetti generati dal protocollo QUIC possono essere suddivisi in due macrocategorie, distinguendo il tipo di header di cui sono dotati: essi si dividono in *long header* e *short header*

3.2.1.1 Long Header

I pacchetti che contengono un long header sono utilizzati esclusivamente durante le fasi di apertura delle connessioni. Come suggerisce il nome, all'interno della sezione iniziale di questi pacchetti sono contenute un gran numero di informazioni.

```
Long Header Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2),
  Type-Specific Bits (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Type-Specific Payload (..),
}
```

Figura 3.4: Struttura di un long header

Di seguito sono riportate le descrizioni di tutti i campi presenti all'interno di un long header:

- **Header Form** : indica al sistema quale tipo di header si sta leggendo. Assume il valore `0b1` all'interno di tutti i long header.
- **Fixed Bit** : bit che assume il valore `0b1` in tutti i pacchetti di QUIC. L'unica eccezione riguarda i pacchetti utilizzati per la negoziazione della versione del protocollo QUIC all'interno dei quali il primo byte ha sempre vaore pari a `0x80`. La presenza di questo valore fisso permette a QUIC di coesistere con altri protocolli come, ad esempio, SRTP (Secure Real-time Transport Protocol).
- **Long Packet Type** : il valore di questo campo permette al sistema di comprendere quale tipo di pacchetto ha ricevuto. Una descrizione dei valori di questi bit e del loro significato è presente al paragrafo 3.2.1.2.
- **Type-specific Bits** : il significato di questi bit dipende direttamente dal tipo di pacchetto ricevuto e, pertanto, non può essere descritto a priori.
- **Version** : il valore di questi 4 byte è utilizzato per indicare, come suggerito dal nome, la versione in uso del protocollo QUIC. Il codice associato alla prima versione di QUIC è `0x00000001`; le successive versioni utilizzeranno un codice successivo.
- **Destination Connection ID Length** : Il valore assunto da questi 8 bit rappresenta la lunghezza del campo successivo. Quando questo byte assume il valore `0b0` si assume che la successiva sezione dell'header sia stata omessa.
- **Destination Connection ID** : questo valore è utilizzato per instradare il pacchetto verso il destinatario. Questo valore rimane costante per tutti i pacchetti associati ad una medesima connessione.

- **Source Connection ID Length** : anche in questo caso si ha un dato che indica la lunghezza del campo che segue. Come per il Destination Connection ID Length, anche in questo caso il valore `0b0` è usato per indicare l'assenza del campo successivo.

- **Source Connection ID** : in questo campo è contenuto il valore che il destinatario del pacchetto utilizzerà come Destination Connection ID per le successive risposte al proprio interlocutore.

- **Type-Specific Payload** : quest'ultima sezione viene usata per inserire tutti i dati provenienti dal livello applicativo. Le informazioni contenute all'interno del payload sono organizzate in strutture chiamate "frame": per una completa descrizione si faccia riferimento al paragrafo 3.2.2.

Si osserva che la struttura sopra descritta potrebbe subire delle variazioni in base alla versione del protocollo QUIC in uso. Sono definiti come invarianti del protocollo QUIC i seguenti campi del long header: *Header Form, Version, DCID Length, DCID, SCID Length, SCID*. Si ha, di conseguenza, che i 7 bit meno significativi contenuti nel primo byte dell'header potrebbero subire variazioni nel loro significato a seconda della alla versione di QUIC in uso.

La diretta conseguenza di questo è riscontrabile all'interno del pacchetto utilizzato per la negoziazione della versione per il protocollo QUIC. Essendo questo un pacchetto per natura indipendente dalla versione in uso, all'interno della propria struttura presenta alcune particolarità. Come detto in precedenza, il primo byte assumerà sempre il valore `0x80` in quanto si ha il primo bit, associato al tipo di header in uso, impostato ad `0b1` ed i rimanenti bit assumono sempre il valore `0b0` perché' non utilizzati. Una seconda particolarità di questo speciale pacchetto è individuabile nel numero di versione del protocollo: anche in questo caso tutto deve rimanere indipendente da ogni versione e, per convenzione, si è deciso di utilizzare il valore `0x00000000`. Infine, all'interno del payload è presente una lista di interi lunghi 32 bit che rappresentano le versioni supportate del protocollo QUIC.

3.2.1.2 Tipi di pacchetti che utilizzano un Long Header

Come accennato in precedenza un long header viene utilizzato in tutti i pacchetti che costituiscono le operazioni di apertura di una connessione. All'interno del protocollo QUIC sono previsti 4 specifici tipi di pacchetto, ognuno con una funzione ben specifica. Di seguito sono riportati in forma tabellare i valori che il campo "Long Packet Type" può assumere.

Ognuno dei suddetti tipi svolge un preciso compito nel contesto di una connessione QUIC, di seguito si presentano per ciascuno i casi di utilizzo:

Long Packet Type Hex	Long Packet Type Bin	Packet Name
0x00	0b00	Initial
0x01	0b01	0-RTT
0x02	0b10	Handshake
0x03	0b11	Retry

Tabella 3.2: Tipi di pacchetto che utilizzano un long header

- **Initial Packet** : viene utilizzato per trasportare i primi *CRYPTO frame* inviati dal client per effettuare lo scambio di chiavi con il server. All'interno di un initial packet è consentito anche inserire frame appartenenti alle seguenti categorie: ACK, PING, PADDING, CONNECTION_CLOSE. Si precisa che questo tipo di pacchetto può essere utilizzato sia dal client che dal server per inviare informazioni crittografiche e non è in alcun modo legato alla sola prima trasmissione di una connessione.

- **0-RTT Packet** : viene utilizzato per effettuare le operazioni omonime di handshake. Si distingue dagli altri tipi di pacchetto perché' permette di inviare al server dati provenienti dal livello applicativo prima della conclusione del processo di handshake. È riservata al server la decisione di accettare o meno i dati contenuti all'interno di questi pacchetti a seconda dell'esito dell'handshake.

- **Handshake Packet** : viene utilizzato per trasportare ulteriori informazioni crittografiche e i relativi messaggi di acknowledgement. Quando un client invia e un server riceve il primo handshake packet tutti i successivi initial packet ricevuti smettono di essere processati e le relative chiavi di cifratura vengono scartate. Da questo momento i CRYPTO frame necessari per terminare il processo di handshake saranno necessariamente allegati ad un Handshake packet.

- **Retry Packet** : viene utilizzato da un server per convalidare l'indirizzo di un client che sta tentando di stabilire una connessione. Il processo di *address validation* può essere applicato anche duranrte la migrazione di una connessione. Questo processo viene utilizzato dai server come difesa contro gli *amplification attack*. All'interno dell'header di questo pacchetto il server inserisce uno specifico "Retry Token" che il client è tenuto ad allegare alla risposta per poter essere validato correttamente. In aggiunta a ciò, all'interno di questo tipo di pacchetto il server inserisce uno specifico Source Connection ID che il client sarà tenuto ad usare come Destination Connection ID durante l'invio della sua risposta. Si osserva, inoltre, che questo specifico tipo di pacchetto è l'unico che non contiene alcun payload aggiuntivo. Per questa ragione il Retry Packet non verrà considerato nella discussione sui diversi frame supportati dal protocollo QUIC.

3.2.1.3 Short Header

La prima versione di QUIC prevede un singolo tipo di pacchetto che utilizza lo short header; questa tipologia è nota come ”1-RTT Packet”. Come è possibile intuire dal nome, questo specifico tipo di struttura viene utilizzato in qualsiasi fase in cui la comunicazione può essere cifrata tramite una chiave consolidata o quando si utilizza una connessione già stabilita. In questa situazione, infatti, non è più necessario scambiare informazioni legate alle chiavi crittografiche o alla versione del protocollo in quanto esse sono state precedentemente definite.

```
1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),
  Key Phase (1),
  Packet Number Length (2),
  Destination Connection ID (0..160),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

Figura 3.5: Struttura di un short header

Come conseguenza si ha che la struttura dell’header risulta più semplice. Di seguito se ne fornisce una descrizione approfondita:

- **Header Form** : indica al sistema quale tipo di header si sta leggendo. Assume il valore `0b0` all’interno di tutti gli short header.

- **Fixed Bit** : bit che assume il valore `0b1` in tutti i pacchetti di QUIC. L’unica eccezione riguarda i pacchetti utilizzati per la negoziazione della versione del protocollo QUIC all’interno dei quali il primo byte ha sempre vaore pari a `0x80`. La presenza di questo valore fisso permette a QUIC di coesistere con altri protocolli come, ad esempio, SRTP (Secure Real-time Transport Protocol).

- **Spin Bit** : bit utilizzato per la misurazione delle prestazioni della rete e, in particolare, del tempo impiegato da un pacchetto a compiere un ”Round Trip”. Per questa ragione il server è tenuto a riflettere il valore dello spin bit di ogni pacchetto all’interno della relativa risposta. Il client, sapendo in che momento è avvenuto l’invio del pacchetto marchiato dallo spin bit, è in grado di misurare l’intervallo di tempo che precede la ricezione di suddetta risposta.

- **Reserved Bits** : questi bit sono riservati per usi futuri del protocollo QUIC. Attualmente questi bit assumono sempre il valore `0b00` e, nel caso in cui un sistema rilevi valore differente in un pacchetto ricevuto, quest'ultimo è tenuto a generare un errore con tipo PROTOCOL_VIOLATION.

- **Key Phase** : bit utilizzato per determinare quale chiave utilizzare per la decriptazione del pacchetto stesso nel caso in cui sia stato generata una nuova coppia di segreti.

- **Packet Number Length** : valore che indica la lunghezza in byte del numero del pacchetto inserito nell'ultima parte dell'header. La lunghezza del numero del pacchetto può variare per ragioni di sicurezza e compressione.

- **Destination Connection ID** : questo campo contiene l'identificatore del destinatario del pacchetto. Come già discusso in precedenza, questo valore è utilizzato per istradare le informazioni in direzione del destinatario e rimane costante in tutte le comunicazioni di una determinata connessione.

- **Packet Number** : contiene il numero identificatore del singolo pacchetto. Come già discusso al paragrafo 2.2.7, questi numeri sono monotonamente crescenti e, anche nel caso di una ritrasmissione, non vengono mai ripetuti.

- **Packet Payload** : la sezione finale del pacchetto contiene i dati provenienti dal livello applicativo che devono essere trasportati. Tutte le informazioni in questa parte del pacchetto sono state precedentemente criptate utilizzando le chiavi negoziate in fase di handshake.

Si osserva che all'interno della struttura sopra presentata tutti i campi, fatta esclusione dell'*Header Form* e del *DCID*, possono variare a seconda della versione del protocollo QUIC in uso.

3.2.2 Frames in un pacchetto QUIC

Come già accennato in precedenza, il contenuto principale di un pacchetto QUIC consiste in una sequenza di strutture dati chiamate "frame". Tutti i pacchetti, fatta eccezione per quelli dedicati alla negoziazione della versione di QUIC e i *Retry Packet*, contengono sempre almeno un frame. Spesso all'interno di un singolo pacchetto sono contenuti molti frame distinti e, nella maggior parte dei casi, questi assumono funzioni differenti. Si osserva che il contenuto di un singolo frame non viene mai frammentato in pacchetti differenti. Nel caso in cui la dimensione dei dati da trasferire superi il limite di capacità di un singolo pacchetto i dati verranno suddivisi in diversi frame trasmessi all'interno di pacchetti differenti. All'interno della tabella 3.3 si presentano i tipi di frame disponibili nella prima versione del protocollo QUIC.

Osservando la suddetta tabella è possibile effettuare immediatamente alcune osservazioni. In primo luogo, si osserva che i pacchetti 1-RTT supportano tutti i tipi di frame previsti all'interno dello standard QUIC: questo comportamento è auspicabile poichè questo suddetto tipo di pacchetto viene utilizzato per la quasi totalità delle comunicazioni. Un comportamento simile è osservabile anche per i pacchetti 0-RTT: anche in questo caso la maggior parte delle tipologie di frame sono supportati. A differenza degli *1-RTT Packet*, però, sono esclusi tipi come l'ACK che non hanno alcuno scopo nel contesto di una prima comunicazione oppure il CRYPTO il cui utilizzo è mutualmente escluso da un handshake 0-RTT. Al contrario, invece, si osserva che i pacchetti Initial e Handshake supportano un sottoinsieme ridotto dei frame disponibili in quanto la maggior parte di essi non assume alcun significato nel contesto in cui questi pacchetti sono impiegati.

Type Value	**Frame Type Name**	**Suported Packet Type** [1]
0x00	PADDING	I H 0 1
0x01	PING	I H 0 1
0x02 - 0x03	ACK	I H _ 1
0x04	RESET_STREAM	_ _ 0 1
0x05	STOP_SENDING	_ _ 0 1
0x06	CRYPTO	I H _ 1
0x07	NEW_TOKEN	_ _ _ 1
0x08 $\longrightarrow$ 0x0F	STREAM	_ _ 0 1
0x10	MAX_DATA	_ _ 0 1
0x11	MAX_STREAM_DATA	_ _ 0 1
0x12 - 0x13	MAX_STREAMS	_ _ 0 1
0x14	DATA_BLOCKED	_ _ 0 1
0x15	STREAM_DATA_BLOCKED	_ _ 0 1
0x16 - 0x17	STREAMS_BLOCLED	_ _ 0 1
0x18	NEW_CONNECTION_ID	_ _ 0 1
0x19	RETIRE_CONNECTION_ID	_ _ 0 1
0x1A	PATH_CHALLENGE	_ _ 0 1
0x1B	PATH_RESPONSE	_ _ _ 1
0x1C - 0x1D	CONNECTION_CLOSE	I H 0 1
0x1E	HANDSHAKE_DONE	_ _ _ 1

Tabella 3.3: Frame supportati dal protocollo QUIC

Di seguito si riporta una breve descrizione di ogniuno dei tipi di frame elencati precedentemente:

[1] I simboli presenti alla colonna Supported Packet Type indicano: I = Initial Packet, H = Handshake Packet, 0 = 0-RTT Packet, 1 = 1-RTT Packet

PADDING Frame Questo tipo di frame non contiene alcun tipo di informazione in quanto tutti i byte contenuti al proprio interno assumono il valore `0x00`. Può essere utilizzato per aumentare la dimensione del pacchetto in modo da raggiungere la taglia desiderata prima della trasmissione.

PING Frame Questo tipo di frame può essere usato per controllare se il sistema con cui si sta comunicando è ancora raggiungibile. Chi riceve questo tipo di messaggio, infatti, è tenuto a rispondere con un acknowledgement del pacchetto appena ricevuto. All'interno di questa struttura non sono presenti dati applicativi ma può essere sfruttato per mantenere aperta una connessione in assenza di informazioni da trasmettere posticipando, di fatto, il timeout della connessione stessa.

ACK Frame Questo tipo di frame è utilizzato per informare il mittente di un pacchetto della corretta ricezione dello stesso. All'interno di questi frame è possibile specificare uno o più intervalli di pacchetti di cui si vuole effettuare l'acknowledgement: grazie a questa funzione è possibile ridurre drasticamente il numero necessario di ACK Frame poiché' non viene richiesta la corrispondenza uno ad uno con i pacchetti ricevuti. Il codice identificativo `0x03` viene utilizzato per indicare la presenza di un campo aggiuntivo all'interno del frame stesso. In questo spazio è indicato il numero di pacchetti QUIC ricevuti sulla corrente connessione che contengono i cosiddetti "ECN marks" (Explicit Congestion Notification). Questo protocollo viene usato dal livello di rete per regolare i parametri di trasmissione dei dati in modo da adattarsi meglio allo stato corrente della rete.

RESET_STREAM Frame Questo tipo di frame viene utilizzato per interrompere le comunicazioni associate ad uno specifico flusso di rete. Il mittente di questo messaggio interrompe le operazioni di trasmissione di frame STREAM associati ad uno specifico *Stream ID*. Il destinatario di questa informazione può reagire scartando tutte le informazioni possedute sul flusso specificato in quanto esso non verrà più utilizzato.

STOP_SENDING Frame Questo tipo di frame viene utilizzato per comunicare che i successivi dati ricevuti attraverso un determinato flusso, identificato dallo *Stream ID*, verranno scartati. Quando un sistema riceve questo tipo di messaggio è tenuto ad interrompere le comunicazioni relative al flusso indicato.

CRYPTO Frame Questo tipo di frame viene utilizzato per trasportare informazioni di carattere crittografico. Pur non avendo particolari caratteristiche, questi pacchetti sono gestiti in

modo differente rispetto ai normali STREAM Frame: essi non appartengono a nessun flusso di dati e sono ”trasparenti” rispetto ai sistemi di controllo della rete (*flow* e *congestion control*).

NEW_TOKEN Frame Questo tipo di frame viene utilizzato dai server per consegnare ad un client un nuovo *token*. Questo ”gettone” potrà essere inserito dal client nell'intestazione di un futuro *Initial Packet* in modo da essere identificato più velocemente da un server a cui è stato precedentemente connesso. Il suddetto *token* può essere anche impiegato in caso di handshake ripetuti come forma di autenticazione del client.

STREAM Frame Questo tipo di frame viene utilizzato per trasportare i dati appartenenti ai flussi creati dal protocollo QUIC. Questo specifico tipo di frame può essere identificato da un grande spettro di valori poiché i tre bit meno significativi di questo campo sono utilizzati come marcatori booleani per diverse funzioni. Nell'ordine essi indicano:

- la presenza del campo ”offset” all'interno del frame (OFF bit). Questo valore viene inserito per indicare come posizionare i dati trasportati dal frame rispetto alle altre informazioni ricevute nello stesso flusso.

- la presenza del campo ”length” all'interno del frame (LEN bit). Questo valore indica la lunghezza dei dati in byte contenuti all'interno del frame stesso.

- la presenza di ulteriori ”STREAM Frame” associati al medesimo flusso (FIN bit). Se questo bit assume il valore `0b1` allora il corrente frame può essere considerato come l'ultimo del relativo flusso di rete.

MAX_DATA Frame Questo tipo di frame viene utilizzato per comunicare al sistema di gestione dei flussi del proprio interlocutore la quantità massima di dati che possono essere inviati per ogni connessione. Se questo limite massimo dovesse essere superato il mittente di suddetto frame sarebbe tenuto a terminare la connessione utilizzando il codice di errore FLOW_CONTROL_ERROR.

MAX_STREAM_DATA Frame Questo tipo di frame, molto simile al precedente, viene utilizzato per indicare la quantità massima di dati che possono essere inviati attraverso un determinato flusso, identificato dallo *Stream ID*. Anche in questo caso, se il limite imposto verrà superato, la connessione dovrà essere terminata segnalando l'errore con il codice FLOW_CONTROL_ERROR.

MAX_STREAMS Frame Questo tipo di frame viene utilizzato per comunicare ad un altro sistema quanti flussi di rete possono essere aperti. Quando il tipo di frame assume il valore `0x12` si sta facendo riferimento ai flussi bidirezionali mentre, quando si utilizza il codice `0x13` per identificare il frame, si vuole indicare il numero massimo di stream unidirezionali. Il conteggio dei flussi comprende anche tutti quelli che sono stati precedentemente terminati e, se la soglia dovesse essere superata, il sistema mittente di questo tipo di messaggio sarebbe tenuto a terminare la connessione utilizzando il codice di errore STREAM_LIMIT_ERROR.

DATA_BLOCKED Frame Questo tipo di frame viene utilizzato quando il mittente intende manifestare la volontà di inviare una maggior quantità di dati rispetto a quella consentita dal sistema di *flow control* che governa la connessione in uso. All'interno di questo frame il mittente indica anche il limite di trasmissione riscontrato durante l'uso di suddetta connessione.

STREAM_DATA_BLOCKED Frame Questo tipo di frame, simile al precedente, viene utilizzato da un sistema per segnalare l'impossibilità di inviare dati in uno specifico flusso dovuta all'azione del sistema di "flow control". In questo caso viene allegato al messaggio anche l'identificatore dello specifico flusso a cui si sta facendo riferimento.

STREAMS_BLOCKED Frame Questo tipo di frame viene utilizzato per segnalare l'incapacità di un sistema di inizializzare nuovi flussi di rete. Questa problematica può sorgere quando, ad esempio, si raggiunge il limite imposto utilizzando i frame di tipo MAX_STREAMS. Anche in questo caso lo standard QUIC fornisce due codici differenti per indicare un frame STREAMS_BLOCKED. Il tipo `0x16` viene utilizzato dopo aver raggiunto il limite per i flussi bidirezionali e, al contrario, per segnalare che non è stato possibile aprire un flusso unidirezionale si utilizza il codice `0x17`.

NEW_CONNECTION_ID Frame Questo tipo di frame viene utilizzato per indicare al proprio interlocutore il valore di un nuovo *Connection ID*. Questo identificatore potrà essere utilizzato nel caso in cui la connessione in uso dovrà essere migrata in seguito, ad esempio, di un cambio di rete. All'interno di questo frame viene anche inserito uno speciale gettone, noto come "Stateless Reset Token". L'invio di questo token in un successivo pacchetto comporterà il ripristino di tutti i parametri associati alla connessione. Questo meccanismo viene utilizzato come ultima risorsa nel caso in cui uno dei due sistemi subisca disservizi temporanei e non sia più in grado di recuperare i dati persi. Si specifica che il token contenuto in questo frame è associato alla connessione identificata dal nuovo codice trasmesso. Si precisa che il meccanismo di "stateless reset" viene utilizzato solamente in casi limite mentre il nuovo *Connection ID* può essere utilizzato nelle nuove connessioni create in condizioni ordinarie.

RETIRE_CONNECTION_ID Frame Questo tipo di frame viene utilizzato per notificare al proprio interlocutore che un *Connection ID* sarà dismesso con effetto immediato. L'identificatore della connessione che deve essere chiusa è indicato come *Sequence Number* all'interno dei dati del frame stesso. Si precisa, inoltre, che una volta terminata una connessione anche lo *Stateless Reset Token* associato verrà invalidato. Questo messaggio svolge anche una funzione secondaria: chiudendo una connessione viene sottointesa la richiesta al proprio interlocutore di nuovi *Connection ID* da utilizzare successivamente. I nuovi identificatori verranno recapitati mediante l'uso di frame con tipo NEW_CONNECTION_ID.

PATH_CHALLENGE Frame Questo tipo di frame viene utilizzato per convalidare un nuovo percorso di rete verso il proprio interlocutore. È necessario svolgere questo tipo di operazione quando, ad esempio, è richiesto di migrare una specifica connessione su una nuova rete. Il sistema che invia questo specifico tipo di frame rimane in attesa di una riposta, contenuta in un frame di tipo PATH_RESPONSE. All'interno di questa struttura sono contenuti 8 byte che contengono dati arbitrari: lo scopo di questi dati è riportato alla sezione seguente.

PATH_RESPONSE Frame Questo tipo di frame viene utilizzato per soddisfare un messaggio avente tipo PATH_CHALLENGE. La sola ricezione di questo tipo di messaggio è sufficiente al mittente di un frame con tipo PATH_CHALLENGE per sapere che il proprio destinatario è raggiungibile. All'interno di questa struttura vengono riportati anche i dati ricevuti in precedenza per indicare a quale *challenge* la corrente *response* è associata. Nel caso in cui non si registri una completa corrispondenza negli 8 byte di payload, il destinatario potrebbe generare un errore di tipo PROTOCOL_VIOLATION per segnalare l'incongruenza.

CONNECTION_CLOSE Frame Questo tipo di frame viene utilizzato per segnalare la chiusura della corrente connessione. All'interno di questa struttura il mittente del messaggio indica il codice di errore, ove presente, che ha portato alla terminazione della comunicazione e un messaggio di tipo testuale utilizzato per descrivere in modo esteso tale decisione. Sono associati a questo frame due distinti codici: quando viene utilizzato il valore `0x1C` si indica la presenza di un errore del protocollo di trasporto QUIC. Al contrario, quando viene impiegato il codice `0x1D`, si segnala un errore registrato a livello applicativo. Per ovvie ragioni, la seconda specializzazione può essere inserita solamente all'interno di pacchetti aventi tipo 0-RTT o 1-RTT.

HANDSHAKE_DONE Frame Questo tipo di frame viene utilizzato da un server che vuole segnalare la corretta conclusione di un processo di handshake. Questo tipo di frame non prevede alcun tipo di dato allegato al messaggio stesso perché può essere utilizzato unicamente per la comunicazione di un evento semplice.

Sono stati presentati tutti i frame compatibili con la prima versione del protocollo QUIC. Trasmettendo e ricevendo una sequenza di queste unità, unite ai diversi header presentati precedentemente, è possibile creare, utilizzare, gestire e chiudere una connessione di rete in tutti i suoi aspetti.

Si osserva che tutti i tipi di frame sono noti a priori da entrambi i sistemi connessi tramite QUIC: questo protocollo attualmente non supporta frame che possono descrivere in modo autonomo il proprio contenuto ne' strutture che descrivono il contenuto di successivi messaggi. Questo sistema permette di ottenere ottime prestazioni in fase di codifica e decodifica dei messaggi sacrificando, però, la flessibilità delle comunicazioni stesse.

Nonostante non possano essere introdotti nuovi tipi di frame in modo dinamico, si precisa che l'uso delle strutture dati previste da QUIC per il trasporto di informazioni non è limitato al solo protocollo HTTP. Grazie all'utilizzo di un sistema di ALPN (Application-Layer Protocol Negotiation) nella fase di handshake è possibile concordare con il proprio interlocutore quale debba essere il protocollo a livello applicativo da impiegare per una determinata connessione. Solitamente questa decisione viene presa da parte del server scegliendo il protocollo che si intende utilizzare nella lista di quelli supportati dal client.

3.3 La gestione delle priorità in presenza di connessioni parallele in HTTP/3

All'interno degli header delle richieste inviate da HTTP/3 sono contenute molte informazioni che il client invia al server, una delle componenti più rilevanti è associata alla chiave "Priority". I valori associati, descritti all'interno del RFC 9218 [4], sono utilizzati dai server web per comprendere in quale ordine processare le richieste ricevute : questo campo viene considerato come un suggerimento proveniente dal client e, a seconda delle politiche di gestione in uso, potrebbe essere ignorato. Nel caso molteplici richieste siano caratterizzate dallo stesso livello di priorità questi indicatori sono usati per stabilire se processare più di una richiesta in modo concorrente oppure se sfruttare un algoritmo di scheduling sequenziale. Questi dati sono rappresentati sotto forma di dizionario nel quale ogni coppia chiave-valore rappresenta un parametro specifico per la richiesta alla quale è associata. Per quanto riguarda i parametri di priorità, essi sono etichettati con la lettera "u" (iniziale della parola inglese "urgency") e possono assumere valori numerici nell'intervallo 0-7: si ottiene un livello di urgenza maggiore utilizzando un valore basso e, nel caso non sia indicato nulla, i server dovrebbero considerare una priorità standard pari a 3. Si precisa che, a seconda delle scelte implementative effettuate durante lo sviluppo del server, l'urgenza di una determinata richiesta potrebbe assumere valori al di fuori dell'intervallo presentato sopra. Allo stesso modo, si potrebbe avere un valore di default assegnato alle richieste che non lo specificano pari a 2 o 4. Queste leggere variazioni possono essere effettuate per calibrare il comportamento di un server in modo da gestire meglio un particolare tipo di traffico.

Questo meccanismo è particolarmente utile quando i dati che devono essere ricevuti non sono tutti dello stesso tipo: se in una pagina web sono presenti diversi contenuti multimediali è possibile posticiparne l'invio da parte del server assegnando alle specifiche richieste un livello di priorità più basso, utilizzando dei valori compresi tra 4 e 6. In questo modo il client ha la possibilità di effettuare molteplici richieste nello stesso istante e, allo stesso tempo, garantire che il contenuto più rilevante sia consegnato nel minor tempo possibile. Si precisa anche che, secondo quanto scritto all'interno dello standard sopracitato, il livello di urgenza 7 è generalmente associato ad attività che non interessano l'utente in modo diretto e generalmente svolte in background come, ad esempio, il download di aggiornamenti software.

Un secondo parametro utile per gestire in modo migliore la banda a disposizione quando si processano richieste concorrenti riguarda il marcatore booleano "i" (iniziale della parola inglese "incremental"). Esso, infatti, può essere inserito all'interno dell'header *priority* nei casi in cui esiste un beneficio se il server web processa in modo parallelo le richieste con pari priorità provenienti dal client. In questi casi si ha che la banda a disposizione per il trasferimento dei dati viene suddivisa tra le diverse risposte generando, di conseguenza, un generale rallentamento

nella consegna delle informazioni. A meno che non sia diversamente specificato, tutte le richieste vengono servite in modo incrementale e non parallelo. Anche in questo caso, come descritto per i valori di urgenza, il marcatore ”i” potrebbe essere ignorato dal server.

I parametri di priorità descritti in precedenza non sono necessariamente costanti nel periodo in cui la richiesta viene processata: all’interno di un messaggio HTTP/3 è possibile inserire una specifica sezione, chiamata ”PRIORITY_UPDATE”, che permette di alterare parametri di una richiesta precedente. Questo meccanismo è particolarmente utile quando, ad esempio, una determinata risorsa viene richiesta in modo preventivo dal client (viene assegnata un livello di priorità pari a 7) ed improvvisamente essa viene richiesta dall’utente. Di conseguenza, nel caso in cui fossero disponibili ulteriori risorse da dedicare alla specifica riposta, i rimanenti dati della risorsa potrebbero essere processati in modo più rapido migliorando così l’esperienza percepita dall’utente finale. Si precisa che il sistema di aggiornamento delle priorità in tempo reale potrebbe non essere sempre disponibile oppure, quando si lavora con connessioni molto veloci e con bassi valori di latenza, potrebbe essere totalmente inefficace.

Nel caso in cui il server non riceva alcun tipo di indicazione riguardo l’ordine in cui servire le richieste ricevute esso ricorrerà all’utilizzo di un algoritmo di scheduling, i più diffusi sono: *Round Robin* (RR), *First Come First Serve* (FCFS) e *Weighted Fair Queuing* (WFQ). Gli standard che descrivono il funzionamento dei protocolli HTTP/3 e QUIC non precisano quale tra gli algoritmi citati sia necessario applicare e offrono la libertà agli sviluppatori di scegliere il sistema che meglio soddisfa le esigenze di ogni specifica applicazione. All’interno delle specifiche protocollari viene solamente indicato che, a prescindere dall’algoritmo scelto, si tenga conto delle informazioni sulla priorità presentate in precedenza.

Capitolo 4

Implementazioni dei protocolli HTTP/3 e QUIC

4.1 Il Ruolo del Sistema Operativo nell'Utilizzo di Protocolli Proprietari

Storicamente lo scambio di dati della maggior parte delle applicazioni che utilizzano la rete è stato gestito dalle diverse implementazioni dello stack TCP/IP, protocolli su cui, ad esempio, sono basate le prime implementazioni di HTTP (versioni 0.9, 1.0, 1.1 e 2.0). Nonostante siano disponibili librerie sviluppate da terzi, la quasi totalità del traffico è generato e gestito grazie alle implementazioni incluse all'interno dei sistemi operativi. Questo livello di integrazione permette a tutte le applicazioni di sfruttare dei servizi di rete altamente ottimizzati rispetto a quella che è la struttura del sistema in uso. La gestione centralizzata da parte del sistema operativo di tutti i flussi di rete che utilizzano il protocollo TCP ha portato diversi vantaggi: in questa configurazione il sistema stesso ricopre un ruolo di amministratore delle interfacce di rete e ha un ampio controllo su tutto il traffico scambiato attraverso esse. Allo stesso tempo, le diverse applicazioni che fanno utilizzo della rete non devono interessarsi di quali siano tutti i processi intermedi per ottenere la consegna di un dato su un altro computer in quanto possono sfruttare i servizi offerti dal sistema operativo utilizzando un approccio "Black Box".

Questo paradigma, però, è stato modificato con l'introduzione del protocollo QUIC: utilizzando per tutti i tipi di applicazione una connessione basata sul protocollo UDP, il controllo su tutto lo scambio dei dati è stato indirettamente spostato ad un livello superiore dello stack di rete. Questo evento si è verificato a causa della natura del protocollo QUIC e delle sue molteplici implementazioni: nonostante esistano alcune librerie create direttamente dagli sviluppatori dei maggiori sistemi operativi, MsQuic nel caso di Microsoft, la maggior parte del traffico HTTP/3

sfrutta codice scritto da aziende terze. Sicuramente, la maggior forza in questo campo è rappresentata da Google, azienda ideatrice del protocollo QUIC nell'anno 2012 ed a oggi maggior utilizzatrice dello stesso. Oltre alle implementazioni sopracitate, attualmente sono disponibili agli sviluppatori circa una dozzina di librerie differenti che consentono di utilizzare il nuovo stack di rete basato sul protocollo HTTP/3. Nonostante quest'ultimo sia stato regolamentato attraverso alcuni standard prodotti dall'Internet Engineering Task Force (es RFC 9114 [5], 9000 [1] e seguenti), esistono diverse aree grigie che hanno permesso di differenziare le diverse implementazioni del protocollo stesso utilizzando algoritmi differenti per gestire alcuni tipi di problematiche come: *stream scheduling, flow control, congestion control e packetization*. Per quanto i problemi sopra elencati siano stati risolti impiegando sempre soluzioni valide dal punto di vista tecnico, ogni gruppo di sviluppatori ha lavorato per ottimizzare principalmente le proprie applicazioni. Questo fenomeno ha portato un notevole vantaggio alle grandi aziende che possiedono le risorse per derivare uno stack di rete basato su HTTP/3 ma, allo stesso tempo, ottimizzato per il particolare traffico prodotto dai propri servizi.

Riprendendo quanto accennato in precedenza, un'altra preoccupazione derivata dalla diversità tra le varie implementazioni del protocollo HTTTP/3 riguarda la perdita di centralità del sistema operativo nella gestione dei flussi di rete. Se tutti gli incarichi di gestione delle comunicazioni vengono spostati a livello applicativo non esiste più nessuna entità in grado di assicurare una corretta suddivisone delle risorse fisiche tra le diverse applicazioni.

Estendendo questo concetto ulteriormente è possibile figurare una situazione limite in cui una singola applicazione, la quale utilizza delle policy estremamente aggressive per la gestione del proprio traffico di rete, possa occupare quasi completamente le risorse a disposizione nell'intero computer rallentando il funzionamento di servizi potenzialmente più importanti. Nella migliore delle ipotesi ciò è causato da errori commessi durante la fase di sviluppo dell'applicazione stessa ma al giorno d'oggi non è difficile pensare che questo fenomeno possa essere sfruttato per scopi malevoli.

Un ulteriore problema annesso a quanto detto precedentemente è rappresentato dall'aumento del carico di lavoro dal punto di vista computazionale per processare le richieste del protocollo HTTP/3: è stato misurato che l'utilizzo questo nuovo tipo di connessione impatta in modo considerevole il carico di lavoro sulle CPU, specialmente quando vengono impiegate librerie non altamente ottimizzate per il tipo di applicazione in uso. Ancora una volta, quindi, le grandi aziende che hanno a disposizione una considerevole quantità di risorse da dedicare allo sviluppo di soluzioni specifiche sono avvantaggiate rispetto ai piccoli programmatori. Questi ultimi, non avendo la possibilità di dedicare le proprie energie ai protocolli di rete sottostanti le proprie applicazioni, sono costretti ad utilizzare soluzioni generiche e perciò peggiori dal punto di vista della performance.

La perdita di controllo da parte del sistema operativo citata precedentemente è derivata dal modo in cui viene utilizzato il protocollo UDP nello stack di rete HTTP/3 e QUIC: le connessioni UDP vengono ridotte ai minimi termini dal punto di vista delle responsabilità e possono essere figurate come una sorta di ”tubo vuoto”. L'unico compito rimasto al sistema operativo, oltre alla creazione di questi canali generici, è prendere i pacchetti preformati da QUIC e trasmetterli in modo passivo sulla rete.

Al termine di questa breve analisi risulta evidente che, nonostante l'introduzione del protocollo QUIC offra innumerevoli vantaggi, essa solleva diversi dubbi dal punto di vista etico. In uno scenario in cui tutto il traffico di rete deve essere processato attraverso implementazioni proprietarie del protocollo QUIC potrebbe affermarsi un vero e proprio monopolio tecnologico dal punto di vista delle comunicazioni di rete. In questa situazione potrebbe essere registrato un generale scompenso nella spartizione delle risorse di un sistema tra le diverse applicazioni che ne richiedono l'uso e, come spesso accade, sarebbero le applicazioni minori a trovarsi in una situazione di generale svantaggio.

Questo fenomeno potrebbe essere mitigato introducendo all'interno dei sistemi operativi di nuovi meccanismi di gestione delle risorse di rete. Aggiornando le implementazioni delle connessioni UDP incluse nelle librerie standard dei diversi sistemi, si potrebbe potenziare il meccanismo di gestione e assegnazione delle risorse di rete. In questo modo si poterebbe aumentare l'autorità esercitata dal sistema operativo sui canali di trasmissione UDP rispetto a quelle che sono le ridotte responsabilità attuali.

4.2 Le Librerie che implementano una Comunicazione basata su HTTP/3 e QUIC

Nonostante la recente pubblicazione dello standard HTTP/3, sono già disponibili per tutti gli utenti molte librerie che lo implementano. Nella tabella sottostante sono riportate le principali:

Nome	**Linguaggio**	**Sviluppatore**	**Repository**
MsQuic	C	Microsoft	github.com/microsoft/msquic
ProxyGen	C++	Facebook	github.com/facebook/proxygen
Quiche	Rust	Cloudflare	github.com/cloudflare/quiche
Neqo	Rust	Mozilla	github.com/mozilla/neqo
s2n-quic	Rust	Amazon WS	github.com/aws/s2n-quic

Tabella 4.1: Librerie che implementano i protocolli HTTP/3 e QUIC

Oltre agli esempi riportati sopra sono disponibili implementazioni anche per i seguenti linguaggi: C#, Go, Haskell, Java e Python. Si osserva che tutte le librerie citate possono essere utilizzate sia durante l'implementazione di un client web che di un server web.

Per quanto riguarda la compatibilità dei browser Internet, sebbene tutti i maggiori programmi in uso a livello globale supportino il protocollo HTTP/3 già a partire dall'anno 2020, il livello i supporto e ottimizzazione può variare a seconda del programma in uso e alcuni potrebbero ancora necessitare di miglioramenti prima di raggiungere il completo supporto. Tra i principali browser che attualmente supportano connessioni basate su QUIC si trovano: Google Chrome, Mozilla Firefox ed Apple Safari. In questo elenco si possono aggiungere anche tutti i browser basati su Chromium - progetto open source di Google su cui è fondato anche Chrome - come, ad esempio, Microsoft Edge, Opera o Brave.

Per quanto riguarda l'aspetto del funzionamento delle suddette libreire, è importante segnalare come esse siano caratterizzate da interfacce pubbliche simili a quelle che i diversi sistemi operativi offrono per la creazione di connessioni tramite il protocollo TCP. Spesso, infatti, è possibile trovare concetti con nomi familiari, ad esempio *Socket*, e funzioni come *accept()* o *connect()*. Queste entità, pur riprendendo la nomenclatura delle controparti TCP, implementano i nuovi meccanismi legati ai protocolli HTTP/3 e QUIC. Anche in questo ambito si osserva l'utilizzo di un approccio *Black Box*: al programmatore che decide di usare una delle librerie citate non viene richiesto di conoscere in modo approfondito tutti i dettagli implementativi ma soltanto di avere una generale familiarità con i protocolli che si stanno utilizzando.

Un dato interessante che è possibile identificare all'interno della repository di Microsoft su GitHub riguarda il futuro dei protocolli di rete secondo questa azienda: si può leggere che, almeno per il momento, HTTP/3 non è destinato a rimpiazzare in modo completo le versioni precedenti del protocollo stesso ma a coesistere con esse. A sostegno di ciò, è esplicitamente

indicato che lo sviluppo degli stack di rete basati su TCP e su UDP continuerà ad essere portato avanti in modo parallelo. Al giorno d'oggi si ha che il traffico HTTP di Internet è suddiviso in modo quasi equo tra tre revisioni del protocollo: 1.1, 2.0 e 3.0. Nonostante ci sia una generale tendenza all'adozione delle versioni più recenti, molte piattaforme online mantengono il supporto anche con le precedenti revisioni di HTTP. In questi casi è compito del browser scegliere in modo autonomo la miglior versione del protocollo HTTP da utilizzare: si precisa che questa decisione viene presa in modo totalmente trasparente rispetto all'utente finale il quale, a meno di indagini approfondite, non è a conoscenza del tipo di connessione in uso.

Altro aspetto importante da sottolineare riguarda l'integrazione di queste nuove librerie all'interno dei sistemi operativi: al momento non è stato pianificato da parte di Microsoft l'inserimento delle funzionalità legate a HTTP/3 all'interno del kernel del sistema Windows. Questa scelta tecnica è stata fatta per consentire, almeno al momento, una gestione più agile delle costanti migliorie e ottimizzazioni applicate all'intero stack di rete. È molto più semplice, anche per Microsoft, gestire per il momento a livello applicativo le ottimizzazioni frequenti di HTTP/3 e QUIC, in aggiunta alla loro interazione con il trasporto basato su UDP. Di conseguenza, viene lasciata agli sviluppatori delle diverse applicazioni la scelta di utilizzare connessioni di rete basate su HTTP/3. Ciò può avvenire implementando una delle librerie riportate nella tabella 4.1 le quali, a loro volta, si appoggiano sulle API di rete del protocollo UDP fornite dal sistema operativo.

Capitolo 5

Conclusioni

L'introduzione dei protocolli HTTP/3 e QUIC rappresenta un notevole passo avanti nel campo delle connessioni di rete e delle comunicazioni su Internet, segnando un cambiamento significativo rispetto al tradizionale stack TCP/IP. Questi nuovi protocolli affrontano alcuni dei limiti delle connessioni TCP, come la latenza, l'Head of Line Blocking e la difficoltà di gestire in parallelo più connessioni, offrendo un nuovo stack basato su QUIC che garantisce connessioni più veloci, stabili e sicure. Questa tesi ha esplorato i meccanismi interni di HTTP/3 e QUIC, concentrandosi su sistemi di multiplexing, sicurezza e sull'innovativo approccio di QUIC alla gestione efficiente e rapida delle connessioni.

Un aspetto centrale del successo di questi protocolli risiede nella scelta del trasporto su UDP. Sfruttando questo protocollo, QUIC ha potuto introdurre soluzioni più efficienti per la gestione della perdita di pacchetti, il controllo di numerosi flussi paralleli e l'eliminazione di tempi di attesa per la trasmissione dei dati, migliorando significativamente le prestazioni, specialmente su reti instabili o lente. Inoltre, il supporto alla migrazione delle connessioni introdotto da QUIC consente agli utenti di dispositivi mobili, spesso soggetti a cambi di rete, di continuare a utilizzare i servizi online senza interruzioni, un vantaggio notevole per l'accesso a contenuti multimediali o per applicazioni in tempo reale come le videochiamate.

In questo contesto, il ruolo di Google come principale promotore e utilizzatore del protocollo ha facilitato notevolmente la sua adozione. Introdotto da Google nel 2012, QUIC è diventato essenziale in molte delle applicazioni dell'azienda, tra cui Google Chrome, YouTube e Google Suite. Questa presenza nei principali servizi online e browser ha favorito la diffusione di QUIC, rendendo rapidamente evidente il valore di questo protocollo nell'intero ecosistema di Internet.

Tra i pro, l'integrazione di TLS direttamente nel protocollo QUIC rappresenta un notevole passo avanti per la sicurezza. Con QUIC, la crittografia è una componente obbligatoria: tutte le informazioni sono automaticamente protette durante la trasmissione. Se in passato la sicurezza era un'opzione aggiuntiva sovrapposta al protocollo di trasporto, ora la cifratura delle comuni-

cazioni è parte integrante del sistema, in linea con le esigenze di sicurezza e privacy del web contemporaneo.

Tuttavia, l'adozione di HTTP/3 e QUIC porta anche con sé alcune sfide e potenziali criticità. La decentralizzazione della gestione delle connessioni, ora spostata a livello applicativo, riduce il controllo che il sistema operativo ha sui flussi di rete, compromettendo il coordinamento dell'uso delle risorse tra le varie applicazioni. Questa perdita di controllo potrebbe portare a situazioni in cui singole applicazioni, con politiche di gestione del traffico aggressive, monopolizzano le risorse del sistema, rallentando altre applicazioni e servizi più critici.

Inoltre, l'implementazione di QUIC richiede un maggior carico computazionale rispetto al tradizionale stack TCP, poiché il protocollo viene gestito in gran parte dal software dell'applicazione e non dal sistema operativo. Questo aumento del carico di lavoro sulle CPU è particolarmente evidente quando si utilizzano librerie non ottimizzate, mettendo in difficoltà gli sviluppatori con meno risorse rispetto alle grandi aziende. Per queste ultime, infatti, è più facile investire nello sviluppo di librerie specificamente adattate al loro traffico, lasciando gli sviluppatori minori con implementazioni generiche meno performanti.

In ultima analisi, i protocolli HTTP/3 e QUIC rappresentano un nuovo standard per le comunicazioni di rete rapide, sicure e a bassa latenza. Attualmente, si stima che circa un terzo del traffico globale utilizzi già QUIC, e l'adozione continua a crescere. Questo trend suggerisce che, con il tempo, la velocità e l'affidabilità delle applicazioni web che utilizzano questi protocolli continueranno a migliorare. Tuttavia, sarà essenziale affrontare le sfide della gestione delle risorse e del carico computazionale, soprattutto per garantire che l'impatto positivo di HTTP/3 e QUIC sull'efficienza e la sicurezza delle connessioni non venga compromesso. La diffusione di questi protocolli segna dunque l'inizio di una nuova fase nell'evoluzione di Internet, una fase che promette significativi miglioramenti ma che richiede una gestione attenta dei suoi aspetti critici.

Bibliografia

[1] J. Iyengar e M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, RFC 9000, mag. 2021. doi: 10.17487/RFC9000. indirizzo: https://www.rfc-editor.org/info/rfc9000.

[2] M. Duke, *QUIC Version 2*, RFC 9369, mag. 2023. doi: 10.17487/RFC9369. indirizzo: https://www.rfc-editor.org/info/rfc9369.

[3] A. Langley, A. Riddoch, A. Wilk et al., «The QUIC Transport Protocol: Design and Internet-Scale Deployment,» in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 183–196, isbn: 9781450346535. doi: 10.1145/3098822.3098842. indirizzo: https://doi.org/10.1145/3098822.3098842.

[4] K. Oku e L. Pardue, *Extensible Prioritization Scheme for HTTP*, RFC 9218, giu. 2022. doi: 10.17487/RFC9218. indirizzo: https://www.rfc-editor.org/info/rfc9218.

[5] M. Bishop, *HTTP/3*, RFC 9114, giu. 2022. doi: 10.17487/RFC9114. indirizzo: https://www.rfc-editor.org/info/rfc9114.

[6] S. Floyd, D. K. K. Ramakrishnan e D. L. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168, set. 2001. doi: 10.17487/RFC3168. indirizzo: https://www.rfc-editor.org/info/rfc3168.

[7] S. Friedl, A. Popov, A. Langley e E. Stephan, *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension*, RFC 7301, lug. 2014. doi: 10.17487/RFC7301. indirizzo: https://www.rfc-editor.org/info/rfc7301.

[8] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, ago. 2018. doi: 10.17487/RFC8446. indirizzo: https://www.rfc-editor.org/info/rfc8446.

[9] M. Thomson e S. Turner, *Using TLS to Secure QUIC*, RFC 9001, mag. 2021. doi: 10.17487/RFC9001. indirizzo: https://www.rfc-editor.org/info/rfc9001.

[10] J. Iyengar e I. Swett, *QUIC Loss Detection and Congestion Control*, RFC 9002, mag. 2021. doi: 10.17487/RFC9002. indirizzo: https://www.rfc-editor.org/info/rfc9002.

[11] M. Thomson e C. Benfield, *HTTP/2*, RFC 9113, giu. 2022. doi: 10.17487/RFC9113. indirizzo: https://www.rfc-editor.org/info/rfc9113.

[12] M. Kühlewind e B. Trammell, *Manageability of the QUIC Transport Protocol*, RFC 9312, set. 2022. doi: 10.17487/RFC9312. indirizzo: https://www.rfc-editor.org/info/rfc9312.

[13] M. Trevisan, D. Giordano, I. Drago e A. S. Khatouni, «Measuring HTTP/3: Adoption and Performance,» in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2021, pp. 1–8. doi: 10.1109/MedComNet52149.2021.9501274.

[14] J. Koch, O. Falowo e N. Elrod, «What We Know About HTTP/3 and Its Implementation: A Literature Review,» in *2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI)*, 2024, pp. 1–7. doi: 10.1109/ICMI60790.2024.10585883.

[15] J. Dizdarević e A. Jukan, «Experimental Benchmarking of HTTP/QUIC Protocol in IoT Cloud/Edge Continuum,» in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6. doi: 10.1109/ICC42927.2021.9500675.

[16] R. Marx, J. Herbots, W. Lamotte e P. Quax, «Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity,» in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 14–20, isbn: 9781450380478. doi: 10.1145/3405796.3405828. indirizzo: https://doi.org/10.1145/3405796.3405828.

[17] G. Carlucci, L. De Cicco e S. Mascolo, «HTTP over UDP: an experimental investigation of QUIC,» in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15, Salamanca, Spain: Association for Computing Machinery, 2015, pp. 609–614, isbn: 9781450331968. doi: 10.1145/2695664.2695706. indirizzo: https://doi.org/10.1145/2695664.2695706.

[18] M. Kosek, H. Cech, V. Bajpai e J. Ott, «Exploring Proxying QUIC and HTTP/3 for Satellite Communication,» in *2022 IFIP Networking Conference (IFIP Networking)*, IEEE, giu. 2022. doi: 10.23919/ifipnetworking55013.2022.9829773. indirizzo: http://dx.doi.org/10.23919/IFIPNetworking55013.2022.9829773.

[19] P. Megyesi, Z. Krämer e S. Molnár, «How quick is QUIC?» In *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6. doi: 10.1109/ICC.2016.7510788.

[20] J. Zhang, L. Yang, X. Gao, G. Tang, J. Zhang e Q. Wang, «Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking,» *IEEE Access*, vol. 9, pp. 14 836–14 848, 2021. doi: 10.1109/ACCESS.2021.3052578.