ALMA MATER STUDIORUM

UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE

"GUGLIELMO MARCONI"

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA PER L'ENERGIA E L'INFROMAZIONE

"PROTOCOLLO QUIC ED EVOLUZIONE DI HTTP"

Elaborato in

Reti di Telecomunicazione

Relatore **Presentato da**

Prof. Walter Cerroni **Grisogani Luca**

SESSIONE I

ANNO ACCADEMICO 2021-22

INDICE

CAPITOLO 1

INTRODUZIONE

1.1 SOMMARIO E INTRODUZIONE

Nel documento qui presente verranno analizzati e confrontati i protocolli di rete per la trasmissione e lo scambio di informazioni sul web.

Un protocollo di rete può essere definito come un insieme di regole che vengono stabilite per instaurare una comunicazione corretta, inoltre specifica cosa deve essere comunicato, in che modo e quando. Si parla di protocollo di rete se le due macchine che prendono parte alla comunicazione sono remote; quindi, andando ad effettuare una connessione tra loro grazie ad internet.

Per prima cosa vengono presentati gli attuali protocolli utilizzati nella rete e ricercati i motivi che hanno portato a crearne e svilupparne di nuovi, effettuando un confronto tra vantaggi e svantaggi di ogni singolo protocollo, analizzandone teoricamente i funzionamenti.

In particolare, ci si soffermerà su due protocolli dello stesso genere, vi sarà un confronto di ciò che c'è sempre stato ed è ancora presente, protocollo HTTP tramite TCP, e ciò che si sta cercando di sviluppare ed implementare al meglio HTTP3 implementato con QUIC. Proprio quest'ultimo verrà approfondito ed esaminato nello specifico, andando a mettere a fuoco i principali motivi per cui si è arrivati al suo sviluppo e alla sua implementazione.

Attualmente la navigazione e lo scambio di informazioni digitali tramite l'utilizzo del web sono all'ordine del giorno. La stragrande maggioranza della popolazione mondiale ha accesso al web e a tutto ciò che ne concerne, il tutto in maniera rapida veloce e quasi istantanea. In pochi secondi chiunque, indistintamente da età e situazione economica, è in grado di reperire informazioni provenienti da ogni parte del mondo tramite qualsiasi dispositivo che permetta una connessione ad Internet.

In questo documento ci si soffermerà proprio su tutti i parametri che possono determinare una maggiore comodità e velocità all'accesso di questi contenuti presenti sul web.

CAPITOLO 2
HTTP, TCP E SPDY

2.1 PANORAMICA HTTP

In telecomunicazioni e informatica HTTP è un acronimo che sta per "HyperText Transfer Protocol" (protocollo di trasferimento di un ipertesto) ovvero un protocollo a livello applicativo utilizzato come sistema per la trasmissione di informazioni sul web.

La storia dell'HTTP risale a una trentina di anni fa, perché è del 1991 la prima versione documentata del protocollo, negli anni vi sono state varie ratifiche per migliorarne le prestazioni ma ancora oggi la base utilizzata è quella risalente al suo anno zero.

Se viene inserito un indirizzo Internet nel browser e poco dopo viene visualizzato un sito web, il browser ha comunicato con il server web attraverso HTTP. Metaforicamente parlando, l'HTTP è la lingua che utilizza il browser per parlare al server web e comunicargli ciò che viene richiesto.

Il protocollo funziona su un meccanismo richiesta/risposta (client/server): il client esegue una richiesta ed il server restituisce la risposta. Nell'uso comune il client corrisponde al browser ed il server al sito web.

HTTP utilizza TCP (anziché UDP) come protocollo di trasporto. Il client HTTP per prima cosa inizia una connessione TCP con il server. Una volta stabilita, i processi browser e server accedono a TCP attraverso le proprie socket. Dal lato client l'interfaccia socket è la porta tra il processo client e la connessione TCP; dal lato server la porta tra il processo server e la connessione TCP.

Per prima cosa, quindi, deve essere fatta una richiesta per una risorsa e poi avviene la risposta con il trasferimento dei dati.

Il client invia richieste e riceve risposte HTTP tramite la propria interfaccia socket. Allo stesso modo il server riceve richieste e invia messaggi di risposta attraverso la propria interfaccia socket.

Vediamo tramite un esempio come avviene questo procedimento immaginando che l'utente richieda accesso ad una pagina web che contiene testo HTML con riferimento a due immagini.

1. Il client richiede di aprire una connessione di tipo TCP verso il server che ospita la risorsa cercata sulla porta 80 che è la porta assegnata di default per l'HTTP.
2. Il server riceve la richiesta di connessione, se le condizioni lo permettono, accetta la richiesta di connessione e invia una notifica al client.
3. Il client allora formula una richiesta HTTP per la risorsa desiderata e passa la richiesta, tramite la socket, al protocollo TCP che si preoccuperà di gestire la connessione con il server.
4. Il server riceve la richiesta HTTP del client e se non ci sono errori di sintassi nella richiesta invia al client il testo HTML della pagina richiesta. A questo punto il Server notifica al TCP di chiudere la connessione con il client non appena arrivi la notifica di corretta ricezione.
5. Il client riceve correttamente i dati contenenti il testo HTML, lo analizza e nell'analisi trova il riferimento alle due immagini.
6. Per ogni immagine vengono ripetuti i passi dall'1-5.
7. Il client, una volta scaricato il corpo HTML della pagina e tutti gli oggetti referenziati dalla stessa, assembla il tutto mostrando a video la pagina web richiesta precedentemente dall'utente.

2.2 INEFFICIENZE HTTP

Il più grande problema per quanto riguarda il protocollo http è quello della latenza.

Infatti, per come è regolato il tutto, il tempo che occorre per effettuare un processo è lungo dato che, come visto nell'esempio sopra, il client deve fare tutto il processo ogni volta che deve fare una nuova richiesta e questo chiaramente crea problemi di performance e di latenza; il problema è che si perde un po' di efficienza perché bisogna rifare tante volte questa operazione.

Per migliorare le prestazioni, una delle prime cose che si è pensato di fare è stato riutilizzare la connessione stabilita e quindi il lavoro fatto per stabilirla, ovvero una

volta che è stata aperta la connessione TCP ed è stata già negoziata una sessione, magari la si tiene aperta e se occorre chiedere più cose al server le si chiede nella stessa connessione. Questo meccanismo nasceva in un'epoca in cui i siti web comunque erano ancora statici e le pagine web avevano bisogno di fare poco più di una sola richiesta, cioè oltre all'HTML, magari caricare qualche immagine GIF attorno.

Ora invece, anziché fare una richiesta per ogni singola immagine e aprire ogni volta una nuova connessione, le richieste si possono impilare; quindi, una cosa che si può fare è il pipelining, ossia piuttosto che fare una richiesta, attendere una risposta dal server e poi farne un'altra e attendere la risposta corrispondente; quindi, elaborare una procedura completa per ogni singola richiesta, è possibile inviare dal client le richieste tutte insieme in sequenza 1 2 3 e poi il server risponderà.

Questo di fatto è l'utilizzo del http che facciamo quotidianamente e che la maggior parte delle applicazioni, non soltanto i browser, utilizza. Oggigiorno si tende a modulare la profondità del pipelining, ossia una connessione stabilita viene mantenuta aperta con un keep-alive per un certo numero di secondi e in questo intervallo si elabora una serie di richieste.

Questa però non è una soluzione efficiente, almeno non secondo Google e gli IETF, perché resta il problema dell'head of line blocking: ciò significa di fatto che sebbene si possano inviare tante richieste una dopo l'altra, anche se le si può fare insieme sulla stessa connessione, il server deve rispondere in ordine, quindi fintanto che non è stata ricevuta la risposta alla prima richiesta non è possibile ricevere dati sulla seconda e via di seguito.

Per questo motivo, non è detto che il semplice pipelining sia una possibile soluzione, infatti la soluzione adottata dalla maggior parte dei browser è quella di aprire una batteria di connessioni; quindi, non vi è una sola connessione TCP ma molteplici, perché il client prova a inviare più richieste in parallelo.

Questo va bene, però rimane il problema di prima che per ogni connessione TCP e ogni handshake si perde del tempo prezioso; quindi, più sono le richieste di connessione aperte, più tempo occorre; insomma, si ripresenta il problema della latenza.

2.3 PANORAMICA TCP

Come visto nell'esempio sopra HTTP utilizza TCP come protocollo di trasporto.
In telecomunicazioni e informatica il Transmission Control Protocol (TCP) è un protocollo di rete a pacchetto di livello di trasporto, appartenente alla suite di protocolli Internet, che si occupa di controllo della trasmissione ovvero rendere affidabile la comunicazione dati in rete tra mittente e destinatario.
TCP permette a due punti terminali all'interno di una rete informatica comune di realizzare una connessione attraverso cui può avvenire una trasmissione bidirezionale dei dati. Non è altro che un protocollo di comunicazione utilizzato per connettere dispositivi su Internet.
Nell'ambito di questa connessione, le eventuali perdite di dati vengono riconosciute e corrette automaticamente; per questo, il TCP viene denominato anche protocollo affidabile.
Si tratta di un protocollo orientato alla connessione, prima di poter trasmettere dati deve stabilire la comunicazione, negoziando e instaurando una connessione tra mittente e destinatario, la quale resta attiva anche in assenza di scambio di dati e viene chiusa quando non più necessaria. Esso, quindi, possiede le funzionalità per creare, mantenere e chiudere una connessione. Il preambolo della connessione, avviene tramite un handshake, ossia i processi devono inviarsi reciprocamente alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento dati.
Un'altra specifica del protocollo è che esso va in esecuzione solo sui sistemi terminali e non negli elementi di rete intermedi (router e commutatori a livello di collegamento), questi ultimi non salvano lo stato della connessione TCP. Infatti, i router intermedi sono completamente ignari delle connessioni TCP; essi vedono datagrammi, non connessioni.
Per quanto riguarda l'atto pratico della connessione TCP, supponiamo che il processo di un host, il client, voglia inizializzare una connessione con il processo in un altro host, il server. Il processo applicativo client informa il livello di trasporto client di voler stabilire una connessione verso un processo nel server.
Vi sarà uno scambio di messaggi tra client e server, il client invia per primo uno speciale segmento TCP, il server a sua volta risponde con un secondo segmento speciale TCP e infine il client risponde nuovamente con un terzo segmento speciale.
I primi due non trasportano carico utile, ossia non hanno dati a livello applicativo; il

terzo può invece trasportare informazioni utili. Dato che i due host si scambiano tre segmenti, questa procedura viene detta handshake a tre vie (three-way handshake). Una volta instaurata una connessione TCP, i due processi applicativi si possono scambiare dati. Il primo manda un flusso di dati attraverso la socket. Quando questi hanno attraversato la porta, sono nelle mani di TCP in esecuzione nel client. TCP dirige i dati al buffer d'invio della connessione, uno dei buffer che viene riservato durante l'iniziale handshake a tre vie, da cui, di tanto in tanto, preleverà spezzoni di dati. Un aspetto interessante è che le specifiche TCP non si preoccupano di indicare quando TCP debba effettivamente inviare i dati nel buffer, affermando che TCP dovrebbe" spedire tali dati in segmenti quando è più conveniente".

2.4 INEFFICIENZE TCP

A differenza di HTTP, TCP non presenta delle inefficienze, anzi le specifiche presentate qui sotto sono proprio i suoi pregi, che però se messi a confronto col "rivale" UDP risultano appunto essere delle inefficienze in quanto lo rendono più lento. Riassumendo possiamo considerare le sue inefficienze come tempistiche più lunghe in termini di attesa ed esecuzione. Quindi il motivo per cui bisognerebbe preferire UDP a TCP sta nei tempi risparmiati sulla connessione, controllo di congestione, controllo di flusso e tutto ciò che rende TCP un protocollo affidabile. Infatti, UDP dalla sua non è ritenuto affidabile, ma vedremo poi come si potrà fare totale affidamento su di esso.
Tornando alle inefficienze/pregi TCP si possono riassumere come segue:

- Il servizio offerto da TCP è il trasporto di un flusso di byte bidirezionale tra due applicazioni in esecuzione su host differenti. Il protocollo permette alle due applicazioni di trasmettere contemporaneamente nelle due direzioni, quindi il servizio può essere considerato "Full-duplex" anche se non tutti i protocolli applicativi basati su TCP utilizzano questa possibilità.

- Il flusso di byte viene frazionato in blocchi per la trasmissione dall'applicazione a TCP (che normalmente `e implementato all'interno del sistema operativo), per la trasmissione all'interno di segmenti TCP, per la consegna all'applicazione

che lo riceve, ma questa divisione in blocchi non è necessariamente la stessa nei diversi passaggi.

-TCP garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine e una volta sola (”at most once”). Più formalmente, il protocollo fornisce ai livelli superiori un servizio equivalente ad una connessione fisica diretta che trasporta un flusso di byte. Questo `e realizzato attraverso vari meccanismi di acknowledgment e di ritrasmissione su timeout.

-TCP offre funzionalità di controllo di errore sui pacchetti pervenuti grazie al campo checksum contenuto nella sua PDU.

-TCP possiede funzionalità di controllo di flusso tra terminali in comunicazione e controllo della congestione sulla connessione, attraverso il meccanismo della finestra scorrevole. Questo permette di ottimizzare l’utilizzo dei buffer di ricezione/invio sui due end devices (controllo di flusso) e di diminuire il numero di segmenti inviati in caso di congestione della rete.

-TCP fornisce un servizio di multiplazione delle connessioni su un host, attraverso il meccanismo delle porte.

2.5 Panoramica SPDY

HTTP2, che è la versione standardizzata di quel protocollo nato in casa Google con il nome di SPDY, prova ad affrontare i problemi presentati sopra sulle inefficienze del classico HTTP.
Ad esempio, a suo favore SPDY presenta le seguenti caratteristiche:
- Effettua il multiplexing delle richieste HTTP simultanee in una singola socket TCP.
- Comprime le intestazioni HTTP.
- Abilita il server ad effettuare il push dei dati al client ogni volta che ce ne sia la possibilità.
- Permette priorità tra le richieste parallele.

2.6 Vantaggi e svantaggi

Il fatto che SPDY impieghi una sola socket TCP per consegnare tutte le risorse web al client fa sì che vi sia un vantaggio, quello di diminuire l'utilizzo di porte al server, ma ha uno svantaggio non da poco, effettuando flussi di multiplexing su una singola connessione TCP, SPDY si pone in svantaggio rispetto a molte connessioni http classiche, ognuna delle quali ha una separata finestra di congestione.
Infatti, una singola perdita di pacchetto nella connessione TCP sottostante implica una diminuzione della finestra di congestione per tutti i flussi SPDY di cui `e stato effettuato il multiplexing.
Un altro svantaggio di SPDY è che un invio di pacchetti in modo non ordinato tramite TCP causa un blocco (head of line blocking) per tutti i flussi SPDY di cui è stato effettuato il multiplexing su questa connessione TCP.

2.7 Conclusioni

Quindi per concludere questa piccola digressione su HTTP2, esso ha rappresentato un passo avanti ma non particolarmente rilevante, perché comunque in particolari applicazioni occorre scegliere tra sicurezza e velocità: per accelerare le procedure occorre semplificarle, a discapito dei controlli eseguibili e comunque più un protocollo è complesso, meno è sicuro.
Nonostante le migliorie portate con HTTP2, non si è completamente risolto il problema dell'head of line blocking che, era l'obiettivo della revisione del protocollo; questo perché sotto c'è comunque il TCP.
Tutte le inefficienze sopra menzionate sono riconducibili all'utilizzo di TCP come protocollo di trasporto. Questo ha motivato Google nel proporre nuovi protocolli affidabili mediante l'utilizzo di UDP come protocollo di trasporto.
Motivato da ciò Google ha proposto QUIC tramite UDP per sostituire HTTP tramite TCP.

CAPITOLO 3

Il protocollo QUIC

3.1 Panoramica QUIC

QUIC, acronimo di Quick UDP Internet Connections è un nuovo protocollo di trasporto sicuro funzionante mediante UDP, progettato da zero ed ottimizzato per la semantica di HTTP/2. Mentre quest'ultimo è stato costruito come protocollo a livello di applicazione, QUIC è stato sviluppato basandosi su decenni di esperienza nel livello di trasporto e sicurezza delle reti, implementa inoltre meccanismi che lo rendono più attraente come moderno protocollo di trasporto multiuso. QUIC fornisce multiplexing e controllo di flusso in modo equivalente ad HTTP/2, sicurezza come TLS, ed infine semantica della connessione, affidabilità e controllo di congestione come in TCP.

3.2 Caratteristiche principali

Descriviamo i meccanismi chiave di QUIC ed i suoi benefici. Le funzionalità di QUIC sono equivalenti alla somma di TCP, TLS e HTTP/2, ma implementate mediante UDP. I vantaggi chiave di QUIC sono:

- Latenza nell'instaurazione della connessione
- Controllo di congestione flessibile
- Multiplexing senza il blocco head-of-line
- Intestazioni autenticate, criptate e Payload
- Controllo di flusso a livello di stream e di connessione
- Forward error correction
- Migrazione della connessione

3.2.1 Latenza nell'istaurazione della connessione

QUIC combina l'handshake crypto con l'handshake di trasporto, riducendo così il numero di RTT richiesti per stabilire una connessione sicura. Le connessioni QUIC sono comunemente 0 RTT, significa che in molte connessioni QUIC i dati possono essere inviati immediatamente senza attendere una risposta dal server, invece dei 1-3 RTT richiesti per TCP+TLS prima che i dati possano essere inviati.
QUIC fornisce uno stream dedicato per essere utilizzato durante l'handshake, i dettagli dell'handshake crypto verranno argomentati nel capitolo successivo.

3.2.2 Controllo di congestione flessibile

QUIC ha un controllo di congestione innestabile, e una migliore segnalazione rispetto a TCP, ci`o significa che può fornire informazioni più ricche all'algoritmo di congestione rispetto a TCP. Attualmente, l'implementazione di QUIC fatta da Google usa una reimplementazione di TCP Cubic, ma allo stesso tempo si stanno sperimentando approcci alternativi.
Un esempio di informazioni migliori è che ogni pacchetto, originale o ritrasmesso che sia, porta con s'è un nuovo numero di sequenza. Questo permette al mittente QUIC di distinguere gli acknowledgement di ritrasmissione da quelli per la trasmissione originale, così facendo si evitano problemi di ambiguità nella ritrasmissione TCP. Gli acknowledgement di QUIC trasportano inoltre esplicitamente il ritardo tra la ricezione del pacchetto e l'invio del suo acknowledgement, e contemporaneamente con l'incremento monotono dei numeri di sequenza, questo permette un calcolo accurato dell'RTT.
Infine, i frame ACK di QUIC supportano fino a 256 intervalli di NACK, quindi QUIC è più elastico nel riordino rispetto a TCP, nonché in grado di mantenere più byte sul collegamento fisico quando si presenta un riordino o una perdita. Sia il client che il server hanno una visione più accurata di quali pacchetti il peer ha ricevuto.

3.2.3 Controllo di flusso a livello di stream e di connessione

QUIC implementa il controllo di flusso a livello di stream e a livello di connessione, seguendo molto da vicino il controllo di flusso di HTTP/2. Il controllo di flusso a livello di stream funziona come segue. Un destinatario QUIC comunica il byte offset assoluto entro ogni flusso fino a quando il destinatario è disposto a ricevere dati. Quando i dati vengono inviati, ricevuti e consegnati in un particolare stream, il destinatario invia dei frame WINDOW_UPDATE che incrementano il limite dell'offset comunicato per questo flusso, permettendo al peer di inviare più dati in esso.
In aggiunta a questo controllo, QUIC implementa anche il controllo di flusso a livello di connessione per limitare il buffer aggregato che un destinatario QUIC `e disposto ad allocare ad una connessione. Il controllo di flusso a livello di connessione lavora similmente a quello a livello di stream, ma i byte consegnati e l'offset ricevuto più alto sono tutti aggregati attraverso tutti gli stream.
In modo simile all'autoregolazione della finestra di ricezione di TCP, QUIC implementa l'autoregolazione del controllo di flusso per entrambi i controller, a livello di stream e a livello di connessione. L'autoregolazione di QUIC accresce la dimensione dei crediti inviati per il frame WINDOW_UPDATE se esso sembra limitare il tasso di invio e strozza il mittente quando l'applicazione ricevente è lenta.

3.2.4 Multiplexing

HTTP/2 su TCP soffre del blocco head-of-line. Da quando HTTP/2 effettua il mulplexing di molti stream tramite l'astrazione di un singolo stream di byte TCP, una perdita di un segmento TCP blocca tutti i successivi segmenti fino a che non arriva una ritrasmissione, a prescindere dal fatto che lo stream HTTP/2 è incapsulato nei segmenti successivi.
Siccome QUIC è stato progettato da zero per operazioni di multiplexing, i pacchetti persi che trasportavano dati per uno specifico stream generalmente impattano solo su quest'ultimo. Ogni frame di stream può essere immediatamente spedito a questo stream al suo arrivo, così gli stream senza perdita possono continuare ad essere riassemblati e a progredire nell'applicazione.

3.2.5 Intestazioni autenticate, criptate e payload

Le intestazioni TCP si mostrano in chiaro e non autenticate sul collegamento fisico, causando una marea di attacchi e problemi di manipolazione dell'intestazione per TCP, come la manipolazione della finestra di ricezione e la sovrascrittura dei numeri di sequenza. Mentre alcuni di questi sono attacchi attivi, altri sono invece meccanismi usati dai middlebox nella rete talvolta nel tentativo di migliorare in modo trasparente le performance di TCP. Per`o, anche il meccanismo di miglioramento delle performance dei middlebox è attualmente ancora un limite all'evoluzione del protocollo di trasporto, come è stato osservato nel progettare MPTCP e nei suoi successivi problemi di dispiegamento. I pacchetti QUIC sono sempre autenticati e tipicamente il payload è totalmente criptato. Le parti dell'intestazione di pacchetto che non sono criptate sono ancora autenticate dal destinatario, per fare in modo di contrastare qualsiasi iniezione o manipolazione di pacchetto da terze parti. QUIC protegge le connessioni da manipolazioni consapevoli o meno dei middlebox alle comunicazioni end-to-end.

3.2.6 Forward Error Correction

Per recuperare i pacchetti persi senza attendere la loro ritrasmissione, QUIC attualmente utilizza un semplice schema FEC basato su XOR. Un pacchetto FEC contiene un numero di pacchetti pari a quelli del gruppo FEC. Se uno dei pacchetti del gruppo FEC viene perso, i contenuti di quel pacchetto possono essere recuperati dal pacchetto FEC e dai pacchetti rimanenti nel gruppo. Il mittente potrebbe decidere se inviare pacchetti FEC per ottimizzare specifici scenari come, ad esempio, l'apertura e la chiusura di una richiesta.

3.2.7 Migrazione della connessione

Le connessioni TCP sono identificate da quattro tuple, indirizzo sorgente, porta sorgente, indirizzo di destinazione e porta di destinazione. Un problema noto con TCP è che le connessioni non sopravvivono ad un cambio di indirizzo IP, ad esempio nello switch tra una connessione WiFi e una rete mobile in uno smartphone, o ad un cambio di numero di porta. Mentre MPTCP indirizza il problema della migrazione della connessione per TCP, esso è ancora afflitto dalla mancanza di supporto da parte del middlebox e la mancanza di distribuzione da parte del SO.

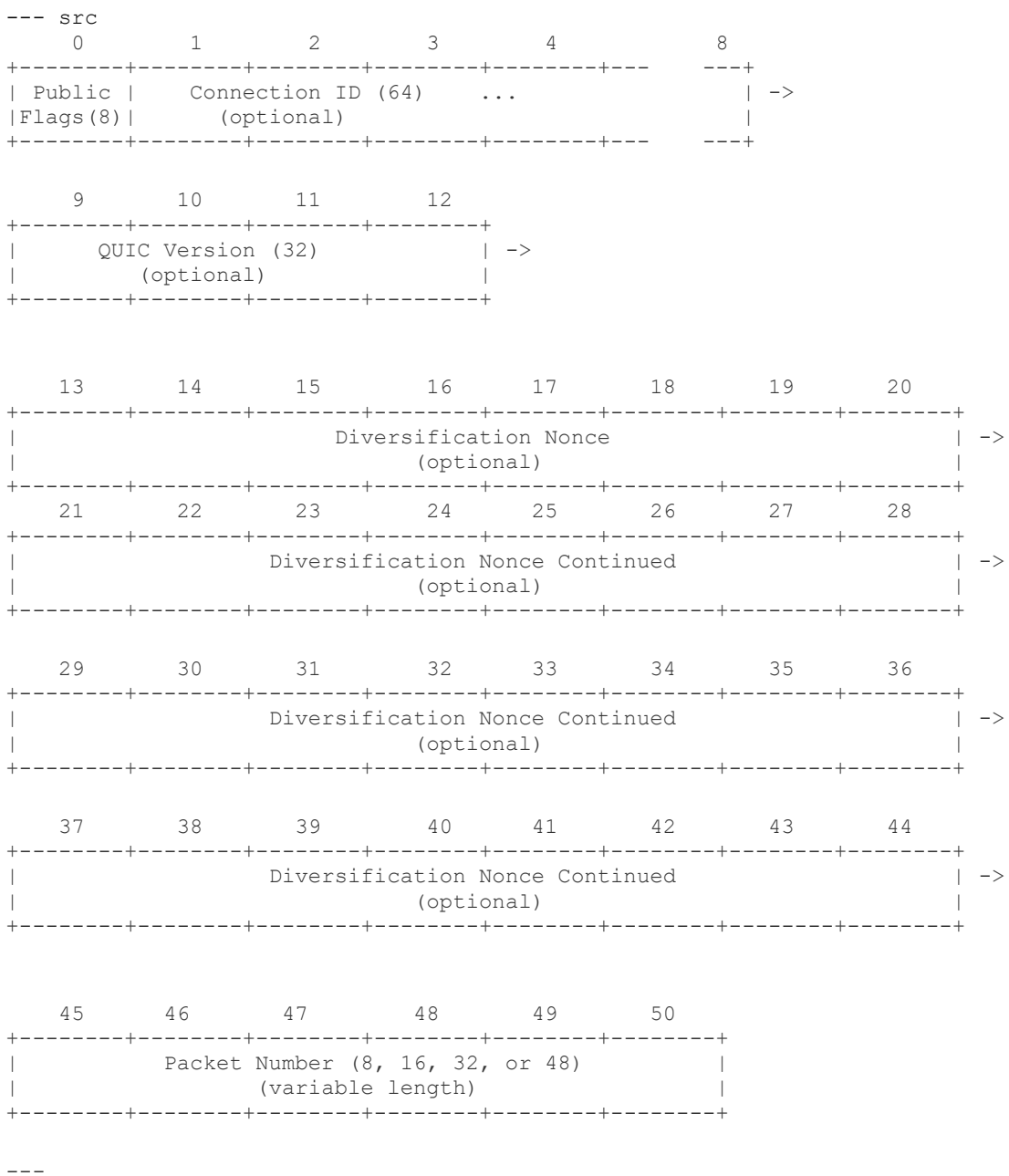
Le connessioni QUIC sono identificate da un ID di connessione a 64-bit, generato casualmente dal client. QUIC può sopravvivere ad un cambio di indirizzo IP e ad un cambio di porta dal momento che l'ID di connessione resta lo stesso attraverso queste migrazioni. QUIC provvede automaticamente anche alla verifica crittografica del client in migrazione, poiché esso continua ad utilizzare la medesima chiave di sessione per criptare e decriptare i pacchetti.

3.3 Tipologie di pacchetti e formati

QUIC usa pacchetti speciali e pacchetti normali. Esistono due tipi di pacchetti speciali, i pacchetti di negoziazione della versione e i pacchetti pubblici di reset, e i pacchetti normali vengono chiamati pacchetti frame. Tutti i pacchetti QUIC dovrebbero essere dimensionati per adattarsi all'interno dei percorsi MTU per evitare la frammentazione dell'IP. La scoperta dei percorsi MTU è tutt'ora in corso, e l'attuale implementazione di QUIC usa un pacchetto QUIC di massimo 1350 byte per IPv6 e di 1370 byte per IPv4.

3.3.1 Intestazione pubblica pacchetto

Tutti i pacchetti QUIC sul collegamento fisico iniziano con un'intestazione pubblica dimensionata tra 1 e 51 byte. Il formato per questa intestazione è composto come segue:



Il payload potrebbe includere vari byte di intestazione tipo-dipendenti come descritto qui sotto. I campi nell'intestazione pubblica sono i seguenti:

Flag pubblici

0x01 = PUBLIC_FLAG_VERSION. L'interpretazione di questo flag dipende da chi invia il pacchetto che sia esso il server o il client. Quando viene inviato dal client, impostare tale flag indica che l'intestazione contiene una versione QUIC. Questo bit deve essere impostato da un client in tutti i pacchetti fino alla conferma da parte del server che accetta la versione proposta dal client. Un server si dice in accordo su una versione quando invia pacchetti senza aver impostato questo bit. Quando tale bit è impostato dal server, il pacchetto è un pacchetto di negoziazione della versione, descritto nel seguito di questo testo.

0x02 = PUBLIC_FLAG_RESET. Impostato per indicare che il pacchetto è un pacchetto pubblico di reset.
Due bit impostati a 0x0C indicano la dimensione dell'ID di connessione che è presente nel pacchetto. Questi bit devono essere impostati a 0x0C in tutti i pacchetti fino a che non vengono negoziati ad un valore diverso per direzione data.

+ 0x0C indica che è presente un ID di connessione a 8 byte
+ 0x08 indica che è presente un ID di connessione a 4 byte
+ 0x04 indica che è utilizzato un ID di connessione ad 1 byte
+ 0x00 indica che è omesso l'ID di connessione

Due bit impostati a 0x30 indicano il numero di byte del numero di pacchetto che sono presenti in ogni pacchetto. I bit sono usati solo per i pacchetti frame. Per i pacchetti pubblici di reset e quelli di negoziazione della versione che non hanno un numero di pacchetto, questi bit non sono utilizzati e devono essere impostati a 0. All'interno di questa maschera di 2 bit:

+ 0x30 indica che sono presenti 6 byte del numero di pacchetto
+ 0x20 indica che sono presenti 4 byte del numero di pacchetto
+ 0x10 indica che sono presenti 2 byte del numero di pacchetto
+ 0x00 indica che è presente 1 byte del numero di pacchetto

0x40 `e riservato per l'utilizzo del multipath.
0x80 attualmente è inutilizzato e dev'essere impostato a 0.

ID di connessione: numero senza segno a 64 bit generato casualmente

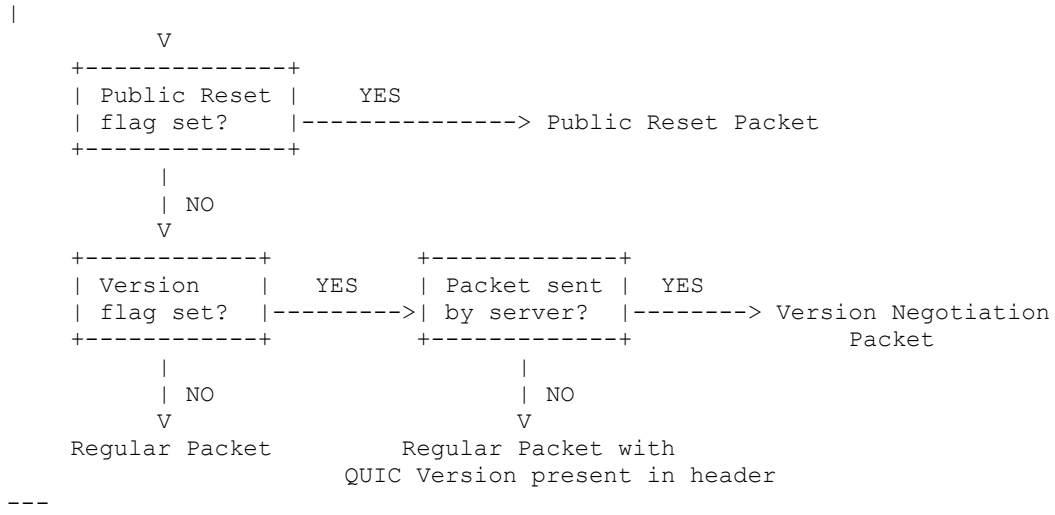
dal client che fa da identificatore della connessione. Siccome le connessioni QUIC sono progettate per restare connesse anche se il client va in errore, i quattro campi dell'intestazione del datagramma IP (IP sorgente, porta sorgente, IP destinazione, porta destinazione) potrebbero essere insufficienti ad identificare la connessione. Per ogni verso della trasmissione, quando è sufficiente meno unicità ad identificare la connessione, un ID di connessione è negoziabile.

Versione di QUIC: un tag a 32 bit che rappresenta la versione del protocollo QUIC. Presente solo se i flag pubblici contengono FLAG_VERSION. Un client potrebbe impostare questo flag ed includere esattamente una versione proposta, o anche includere dati arbitrari. Un server invece potrebbe impostare questo flag quando la versione proposta dal client non è supportata e fornire una lista di versioni accettabili, ma non deve includere nessun dato oltre alle versioni.

Numero di pacchetto: Il più basso tra 8, 16, 32 o 48 bit del numero di pacchetto, basato su quale flag FLAG_?BYTE_SEQUENCE_NUMBER è impostato nei flag pubblici. Ogni pacchetto normale ha assegnato da parte del mittente un numero di pacchetto. Il primo pacchetto inviato da un host terminale deve avere un numero di pacchetto pari ad 1 e ogni pacchetto successivo deve avere un numero di pacchetto più grande del pacchetto precedente. I 64 bit più in basso del numero di pacchetto sono usati come parte crittografica per il momento, quindi un host terminale QUIC non deve inviare un pacchetto con un numero di pacchetto che non può essere rappresentato a 64 bit. Se un terminale QUIC trasmette un pacchetto con un numero di pacchetto di (2ˆ64-1), quest'ultimo deve includere un frame CONNECTION_CLOSE con un codice di errore QUIC_SEQUENCE_NUMBER_LIMIT_REACHED, e l'host terminale non deve trasmettere nessun altro pacchetto aggiuntivo. Al massimo i più bassi 48 bit del numero di pacchetto vengono trasmessi. Per abilitare una ricostruzione non ambigua del numero di pacchetto da parte del destinatario, un terminale QUIC non deve trasmettere un pacchetto il cui numero di pacchetto sia maggiore di (2ˆ(bitlength-2)) rispetto al numero di pacchetto più grande per il quale si sa che un acknowledgment è stato trasmesso

dal destinatario. Perciò, non ci devono essere mai più di (2^46) pacchetti in viaggio. Qualsiasi numero di pacchetto troncato può essere dedotto di avere il valore più vicino a quello del numero di pacchetto più grande conosciuto del terminale che ha trasmesso il pacchetto che originariamente conteneva il numero di pacchetto troncato. La porzione trasmessa del numero di pacchetto combacia con i bit più bassi del valore dedotto.

L'elaborazione di flag pubblici è rappresentata nel diagramma di flusso seguente:



3.3.2 Pacchetti normali

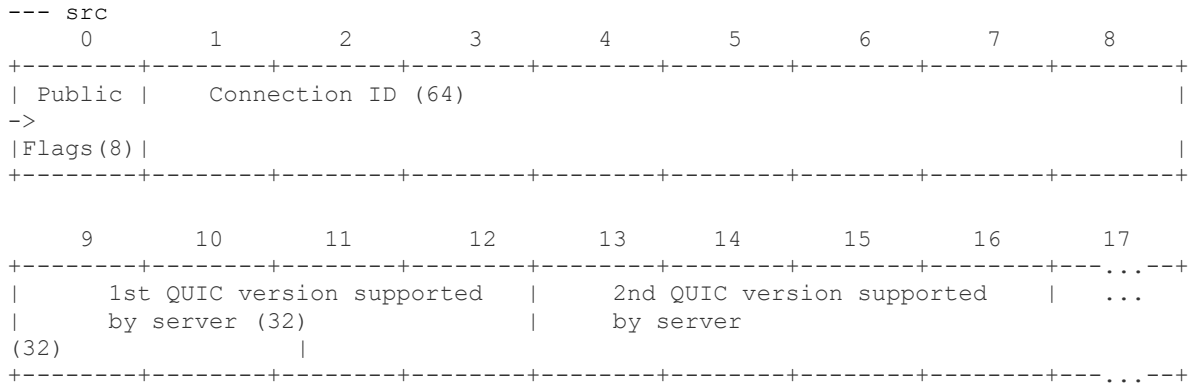
I pacchetti normali sono autenticati e criptati. L'intestazione pubblica è autenticata ma non criptata, e il resto del pacchetto inizia con il campo dei flag privati criptati. Subito dopo segue l'intestazione pubblica, i pacchetti normali contengono dati AEAD (authenticated encryption and associated data). Questi dati devono essere decriptati per interpretarne il contenuto. Dopo la decrittazione, il testo in chiaro inizia con un'intestazione privata.

3.3.3 Pacchetti speciali

Pacchetto di negoziazione della versione

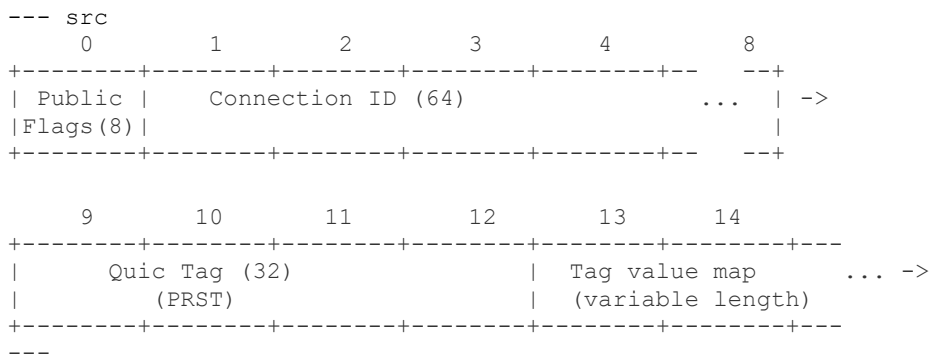
Un pacchetto di negoziazione della versione è inviato solamente dal server. Questi pacchetti iniziano con un flag pubblico ad 8 bit ed un ID di connessione a 64 bit. I flag pubblici devono impostare PUBLIC_FLAG_VERSION ed indicare l'ID di connessione a 64 bit. Il resto del pacchetto è una lista a 4 byte

di versioni che il server supporta:



Pacchetto pubblico di reset

Un pacchetto pubblico di reset inizia con un flag pubblico ad 8 bit ed un ID di connessione a 64 bit. I flag pubblici devono impostare PUBLIC_FLAG_RESET ed indicare l'ID di connessione a 64 bit. Il resto del pacchetto è codificato come se fosse un messaggio di handshake crypto del tag PRST.



Mappa dei valori tag: La mappa contiene i seguenti valori

- RNON (public reset nonce proof) – Un intero a 64 bit senza sengo. Obbligatorio.
- RSEQ (rejected packet number) – Un numero di pacchetto a 64 bit. Obbligatorio.
- CADR (client address) - l'indirizzo IP e il numero di porta rilevati del client. Attualmente utilizzato solamente per scopi di debug e quindi opzionale.

3.4 Ciclo di vita di una connessione QUIC

3.4.1 Instaurazione della connessione

Un client QUIC è il terminale che inizia la connessione. L'instaurazione della connessione QUIC intreccia la negoziazione della versione con gli handshake crypto e
di trasporto per ridurre la latenza di instaurazione. Il tutto viene eseguito nella seguente maniera:

1. Ognuno dei pacchetti iniziali inviati dal client al server devono aver impostato il flag della versione, e devono inoltre specificare la versione del protocollo utilizzata. Ogni pacchetto inviato dal client deve avere il flag della versione attivo fino a che non riceve un pacchetto dal server con il flag disattivato.
   Dopo che il server riceve il primo pacchetto dal client con il flag versione disattivato, deve ignorare qualsiasi pacchetto con il flag attivo.

2. Quando il server riceve un pacchetto con un ID di connessione per una nuova connessione, comparerà la versione del client con le versioni da lui supportate. Se la versione del client è accettabile per il server, quest'ultimo utilizzerà quella versione del protocollo per tutta la durata della connessione. In questo caso tutti i pacchetti inviati dal server avranno il flag versione disattivato.

3. Se invece la versione del client non è accettabile per il server, si incorrerà in un ritardo di 1 RTT. Il server invierà un pacchetto di negoziazione della versione al client. Questo pacchetto avrà il flag versione impostato ed includerà la lista delle versioni supportate dal server.

4. Quando il client riceve un pacchetto di negoziazione della versione dal server, selezionerà una versione del protocollo accettabile e rinvierà tutti i pacchetti usando tale versione. Questi pacchetti devono continuare ad avere il flag versione impostato ed includere la nuova versione del protocollo negoziata. Infine, il client riceve il primo pacchetto normale dal server indicando quindi la fine della negoziazione della versione, e il client ora invia tutti i successivi pacchetti col flag versione disattivato.

Per fare in modo di evitare attacchi di downgrade, la versione del protocollo che il client ha specificato nel primo pacchetto e la lista delle versioni supportate dal server devono essere incluse nei dati dell'handshake crypto. Il client ha necessità di verificare che la lista di versioni del server proveniente dall'handshake corrisponda con la stessa lista proveniente dal pacchetto di negoziazione della versione. Il server invece ha necessità di verificare che la versione del client proveniente dall'handshake rappresenti una versione del protocollo che attualmente non supporta.
Il resto dell'instaurazione della connessione è gestito dall'handshake crypto, che viene attuato su uno stream dedicato. Durante l'instaurazione, l'handshake deve negoziare svariati parametri di trasporto.

3.4.2 Trasferimento dati

QUIC implementa affidabilità della connessione, controllo di congestione e controllo di flusso. Il controllo di flusso QUIC segue molto da vicino quello di HTTP/2.
Una connessione QUIC usa un singolo spazio per il numero di sequenza del pacchetto per un controllo di congestione condiviso e un recupero delle perdite attraverso la connessione.
Tutti i dati trasferiti in una connessione QUIC, incluso l'handshake crypto, sono inviati come dati all'interno degli stream, ma gli ACK accettano i pacchetti QUIC.

3.4.3 Ciclo di vita di uno stream QUIC

Gli stream sono sequenze indipendenti di dati bidirezionali ridotti dentro frame di stream. Gli stream possono essere creati sia dal client che dal server, possono simultaneamente inviare dati intrecciati con altri stream e possono essere cancellati.
Il ciclo di vita di uno stream QUIC è modellato seguendo molto da vicino l'RFC 7540 relativo ad HTTP/2.
La creazione di uno stream avviene in modo implicito, tramite l'invio di un frame stream per un dato stream. Per evitare una collisione di ID di stream, tale ID deve essere pari se il server avvia lo stream e dispari se invece è il client ad avviarlo. 0 non è un ID valido. 1 è un ID riservato per l'handshake crypto, che dovrebbe essere il primo stream avviato dal client. Quando viene usato HTTP/2 tramite QUIC lo stream 3 è riservato per la trasmissione di intestazioni compresse per tutti gli altri

stream, assicurando una consegna affidabile e ordinata ed una lavorazione delle intestazioni. Gli ID degli stream da ogni lato della connessione devono incrementarsi in maniera monotona quando vengono creati nuovi stream. Se il terminale che riceve un frame di stream non vuole accettare lo stream, può immediatamente rispondere con un frame RST_STREAM. Si noti, comunque che il terminale avviato potrebbe avere già i dati inviati sullo stream, questi dati devono essere ignorati.
Una volta che uno stream è stato creato, può essere usato per inviare e ricevere dati. Questo significa che una serie di frame di stream possono essere inviati da un terminale QUIC su uno stream finché quest'ultimo è terminato in quella direzione.
Entrambi i terminali QUIC possono chiudere uno stream normalmente. Ci sono tre modi con cui chiudere uno stream:

1. **Chiusura normale**: dal momento che gli stream sono bidirezionali, possono essere "mezzi-chiusi" o "chiusi". Quando un lato dello stream invia un frame con il bit FIN impostato a true, lo stream `e considerato "mezzo-chiuso" in quella direzione. Il bit FIN indica che nessun dato ulteriore sarà inviato dal mittente di tale bit in questo stream. Quando un terminale QUIC ha sia inviato che ricevuto un bit FIN, il terminale considera lo stream "chiuso". Mentre il FIN dovrebbe essere inviato con gli ultimi dati dell'utente per uno stream, il bit FIN può essere inviato in uno stream frame vuoto dopo gli ultimi dati nello stream.

2. **Chiusura brusca**: il client o il server può inviare un frame RST_STREAM per uno stream in qualsiasi momento. Tale frame contiene un codice di errore per indicare le ragioni del fallimento. Quando questo frame viene inviato dallo stream originario, significa che si è verificato un fallimento nel completare lo stream e che non saranno inviati ulteriori dati. Se inviato invece dallo stream destinatario, il mittente, una volta ricevuto dovrebbe cessare di inviare dati in questo stream. Lo stream destinatario dovrebbe essere consapevole del fatto che c'è una gara tra i dati già in transito dal mittente ed il tempo in cui il frame RST_STREAM viene ricevuto. Per far sì che il controllo di flusso a livello di connessione sia considerato correttamente anche se un frame RST_STREAM è stato ricevuto, un mittente necessita di assicurare entrambi: il FIN e tutti

i byte nello stream ricevuti dal peer o un frame RST_STREAM ricevuto dal peer. Questo significa anche che il mittente di un RST_STREAM necessita di continuare a rispondere ai frame stream in ingresso su questo stream con l'appropriato WINDOW_UPDATE per assicurare che il mittente non abbia il controllo di flusso bloccato tentando di inviare il FIN.

3. Gli stream vengono anche chiusi quando la connessione è terminata, come descritto nel paragrafo successivo.

3.4.4 Termine della connessione

Le connessioni dovrebbero rimanere aperte finché non diventano inattive per un lasso di tempo prestabilito. Quando un server decide di terminare una connessione inattiva, non dovrebbe notificare il client per evitare l'attivazione del comparto connettività nei dispositivi mobili. Una connessione QUIC, una volta stabilita, può concludersi in due modi:

1. Spegnimento esplicito: Un host terminale invia un frame CONNECTION_CLOSE al peer per iniziare la chiusura della connessione. Un host potrebbe inviare un frame GOAWAY al peer prima di un CONNECTION_CLOSE per indicare che la connessione sarà terminata al più presto. Quando un frame GOAWAY invia segnali al peer, qualsiasi stream attivo continuerà ad essere lavorato, ma il mittente del frame GOAWAY non inizierà nessuno stream aggiuntivo e non accetterà nessuno stream in entrata. Al termine degli stream attivi potrebbe essere inviato un CONNECTION_CLOSE. Se un host terminale invia un frame CONNECTION_CLOSE mentre sono attivi stream non terminati, il peer deve assumere che gli stream erano incompleti e terminati in modo anomalo.

2. Spegnimento implicito: Il timeout predefinito di inattività per una connessione QUIC è di 30 secondi, ed è un parametro richiesto nella negoziazione della connessione. Il massimo è di 10 minuti. Se non c'è nessuna attività di rete per tutta la durata del timeout di inattività, la connessione viene chiusa. Per impostazione predefinita verrà inviato un frame CONNECTION_CLOSE. Un

opzione di chiusura tacita può essere abilitata quando è dispendioso inviare una chiusura esplicita, come le reti mobili che devono attivare il comparto connettività.
Un host terminale potrebbe anche inviare un pacchetto PUBLIC_RESET in qualsiasi momento durante la connessione per terminare bruscamente una connessione attiva. Un PUBLIC_RESET è l'equivalente QUIC di un TCP RST.

3.5 Stratificazione di HTTP/2 tramite QUIC

Sin da quando QUIC integra vari meccanismi di trasporto di HTTP/2, implementa anche un buon numero di funzionalità che sono già specificate in HTTP/2.
Di conseguenza, QUIC permette ai meccanismi di HTTP/2 di essere sostituiti dall'implementazione QUIC, riducendo la complessità insita nel protocollo HTTP/2.
Questa sezione andrà a descrivere come la semantica di HTTP/2 può essere offerta al di sopra di un'implementazione QUIC.

3.5.1 Gestione dello stream

Quando le intestazioni HTTP/2 e i dati sono inviati tramite QUIC, lo strato QUIC si occupa della maggior parte della gestione degli stream. Gli ID degli stream HTTP/2 vengono sostituiti dagli ID degli stream QUIC. HTTP/2 non ha necessità di effettuare nessun tipo di stream framing esplicito quando usa QUIC, i dati inviati attraverso uno stream QUIC consistono semplicemente in intestazioni o corpo HTTP/2.
Richieste e risposte vengono considerate complete quando lo stream QUIC è chiuso nella direzione corrispondente.
Il controllo di flusso di dati è gestito da QUIC, e non ha bisogno di essere re implementato in HTTP/2. Il controllo di flusso QUIC sostituisce i due livelli presenti attualmente nelle implementazioni di HTTP/2, uno a livello di applicazione (HTTP/2) e l'altro a livello di trasporto (TCP).

3.5.2 Compressione dell'intestazione HTTP/2

QUIC implementa la compressione delle intestazioni HPACK per HTTP/2, che sfortunatamente introduce problemi di blocchi head-of-line dato che i blocchi di intestazione devono essere decompressi nell'ordine in cui sono stati compressi.
Dal momento che gli stream possono essere trattati in modo arbitrario ad un destinatario, viene applicato un rigoroso ordinamento attraverso le intestazioni, con l'invio di tali intestazioni in uno stream dedicato ad esse, con l'ID stream pari a 3.

Un lavoro futuro sarà modificare il compressore e il decompressore in QUIC, cosicché l'output compresso non dipenda da un precedente stato compresso privo di acknowledgement. Questo potrebbe essere fatto, forse, attraverso la creazione di "checkpoint" di stato HPACK che vengono aggiornati quando le intestazioni hanno ricevuto l'acknowledgement. Nella compressione delle intestazioni QUIC si comprimerebbero solo quelle relative al precedente "checkpoint".

3.5.3 Negoziazione QUIC in HTTP

L'intestazione Alternat-Protocol viene usata per negoziare l'utilizzo di QUIC nelle future richieste HTTP. Per specificare QUIC come Alternat-Protocol disponibile sulla porta 123, un server utilizza:

"Alternate-Protocol: 123:quic"

Quando un client riceve un'intestazione del genere, può tentare di utilizzare QUIC per le future connessioni sicure su quel dominio. Da quando i middlebox e/o i firewall possono bloccare comunicazioni QUIC e/o UDP, un client dovrebbe implementare una valida alternativa TCP quando QUIC è irraggiungibile.
Il server potrebbe rispondere con più valori di campo o con un valore separato da virgole per Alternat-Protocol per indicare i vari tipi di trasporto supportati.
Un server può anche inviare un'intestazione per notificare che QUIC non dovrebbe essere usato in un determinato dominio. Se invia l'intestazione di Alternat-Protocol richiesta, il client dovrebbe ricordarsi di non usare QUIC su quel dominio in futuro, e non fare nessun tipo di sondaggio UDP per vedere se QUIC è disponibile.

3.6 QUIC Discovery

3.6.1 Primo collegamento

Quando Chrome fa una richiesta a un'origine con cui non ha mai parlato prima, non sa che l'origine supporta QUIC e quindi invierà la prima richiesta tramite TCP. La risposta del server indicherà che supporta anche QUIC tramite l'intestazione della risposta HTTP Alt-Svc. (Ad esempio, la risposta. "Alt-Svc: 443:quic" dice a Chrome che l'origine supporta QUIC sulla porta 443.). Ora Chrome sa che QUIC è supportato e può tentare di utilizzare QUIC per la richiesta successiva. Quando viene effettuata la richiesta, Chrome eseguirà un processo di connessione QUIC job con un processo di connessione job TCP. (Questi job sono processi che stabiliscono una connessione ma non emettono la richiesta.)

Poiché la prima richiesta è stata inviata su TCP, il job TCP vincerà la gara quasi immediatamente e la seconda richiesta verrà inviata su TCP. Ad un certo punto, la connessione QUIC avrà esito positivo. Una volta che ciò accade, tutte le richieste future verranno inviate tramite la connessione QUIC.

3.6.2 Collegamenti successivi

Chrome mantiene memorizzato che un'origine supporta QUIC, così come mantiene memorizzato che la configurazione del server che abilita l'handshake 0-RTT di QUIC. Quando Chrome fa una richiesta a un'origine con cui ha parlato in precedenza QUIC, eseguirà un job TCP con un job QUIC. Dal momento che Chrome sarà in grado di eseguire un handshake 0-RTT, il processo QUIC vincerà immediatamente e la richiesta verrà emessa sulla connessione QUIC.

3.6.3 Collegamenti interrotti

Nel caso in cui un handshake QUIC non vada a buon fine (ad esempio se UDP è bloccato o se il server non parla una versione compatibile di QUIC), Chrome contrassegnerà QUIC come interrotto per quell'host. Tutte le richieste in corso verranno emesse nuovamente tramite TCP. Mentre QUIC è contrassegnato come “interrotto” per un host, non verrà tentata alcuna connessione QUIC. Dopo 5 minuti,

la voce “interrotto” sarà scaduta e QUIC verrà contrassegnato con la voce "recently broken" per quell'origine. Quando la richiesta successiva viene inviata all'origine, Chrome eseguirà un job QUIC con un job TCP. Poiché QUIC è stato "recently broken", l'handshake 0-RTT verrà disabilitato. Se l’handshake fallisce di nuovo, questa volta QUIC verrà contrassegnato come “interrotto” per quell'origine per 10 minuti e il processo si ripete contrassegnando QUIC come “interrotto” per il doppio del periodo precedente. Se l'handshake ha esito positivo, la richiesta verrà inviata tramite QUIC e QUIC non verrà più contrassegnato come "recently broken".

CAPITOLO 4

QUIC Crypto Handshake

Il protocollo QUIC Crypto è la parte di QUIC che fornisce la sicurezza di trasporto a una connessione. Questo protocollo è destinato a morire. Sarà sostituito da TLS 1.3 nel futuro, ma QUIC necessitava di un protocollo crypto prima che TLS 1.3 fosse ideato.
Con l'attuale protocollo QUIC crypto, quando il client ha informazioni in cache riguardanti il server, può instaurare una connessione crittografata senza nessun RTT.
TLS, invece richiede almeno 2 RTT. L'handshake QUIC dovrebbe essere circa 5 volte più efficiente rispetto al comune handshake TLS e ad un livello di sicurezza più elevato.
Si parlerà in questo capitolo di come eliminando gli RTT sia necessario far fronte a due problemi di sicurezza, l'IP spoofing e il replay-attack.
L'IP spoofing è una tecnica di attacco alla sicurezza informatica tramite il quale si crea un pacchetto IP nel quale viene falsificato l'indirizzo IP del mittente. Questa tecnica può essere utilizzata per superare alcune tecniche difensive contro le intrusioni, in primis quelle basate sull'autenticazione dell'indirizzo IP.
Il replay-attack invece è una forma di attacco di rete che consiste nell'impossessarsi di una credenziale di autenticazione comunicata da un host ad un altro, e riproporla successivamente simulando l'identità dell'emittente. In genere l'azione viene compiuta da un attaccante che s'interpone tra i due lati comunicanti. Questo attacco permette operazioni fraudolente come falsa autenticazione e/o transazioni duplicate, senza dover necessariamente decrittare la password, ma soltanto ritrasmettendola in un tempo successivo.

4.1 IP Spoofing

Un numero molto ridotto di protocolli su Internet funziona senza almeno un RTT di inizializzazione. Molti protocolli hanno un RTT a causa di TCP e i protocolli basati su TLS in aggiunta hanno almeno un RTT in più prima che i dati possano fluire. Entrambi gli RTT scambiano valori: numeri di sequenza o SYN cookie nel caso di TCP e valori casuali crittografati (client_random e server_random) in

TLS. Il valore TCP previene l'IP spoofing e quello TLS previene attacchi di replay. Qualsiasi protocollo che cerca di eliminare gli RTT deve in qualche modo indirizzare questi due problemi.
Come controesempio, DNS è un protocollo che non ha nessun tipo di RTT di inizializzazione, così deve occuparsi autonomamente di IP spoofing e attacchi di replay.
DNS ignora semplicemente l'IP spoofing e così gli attacchi DDoS sono un problema reale. Per una protezione dal replay, DNSSEC si basa su sincronizzazione dell'orologio e firme a vita breve. Questo permette attacchi di replay per un tempo limitato come da design perché è compatibile con la semantica di caching DNS. Ma non è una forte forma di protezione dagli attacchi di replay perché in fondo questi ultimi sono permessi.
In QUIC questi due problemi vengono trattati separatamente.
Il problema dell'IP spoofing è gestito mediante l'emissione su richiesta da parte del client di un token di indirizzo sorgente. Questo dal punto di vista del client è una stringa di byte poco chiara, invece dal punto di vista del server è un blocco criptato autenticato che contiene, almeno, l'indirizzo IP del client e una marcatura oraria da parte del server. Il server invierà solo un token di indirizzo sorgente per un dato IP su quell'IP. La ricezione del token da parte del client viene presa come prova di proprietà` dell'indirizzo IP nello stesso modo in cui TCP riceve un numero di sequenza.
I client possono includere il token di indirizzo sorgente nelle richieste future in modo da dimostrare il possesso del loro indirizzo IP sorgente. Se il client ha modificato gli indirizzi IP, il token è molto vecchio o il client non ha un token, poi il server potrebbe rifiutare la connessione e far arrivare un nuovo token al client.
Ma se il client è rimasto sullo stesso indirizzo IP può riutilizzare un token di indirizzo sorgente per evitare l'RTT necessario ad ottenerne uno nuovo. La durata di vita di un token è un problema per il server, ma fin da quando i token di indirizzo sorgente sono portatori di token, possono essere rubati e riusati in modo da bypassare le restrizioni basate sull'indirizzo IP. Sebbene l'attaccante non voglia ricevere la risposta. I token possono anche essere collezionati e possibilmente usati dopo che la proprietà dell'indirizzo IP è cambiata. Riducendo la durata di vita dei token si migliorano entrambi i problemi al costo di ridurre il numero di richieste che possono essere gestite senza l'ausilio di RTT aggiuntivi. I token di indirizzo sorgente, a differenza di uno scambio di numeri di sequenza TCP, non richiede la sorgente per dimostrare una capacità continua di ricevere pacchetti inviati all'indirizzo IP sorgente. Questo permette ad un token di essere usato per richiedere continuamente traffico da un server anche una volta

che il downstream è stato saturato e il tasso di diminuzione è abbastanza alto che una connessione TCP non possa essere stabilita.
Però si può notare che un trucco simile è possibile attuarlo anche con TCP e così QUIC non rende le cose peggiori in questo senso. Infatti, una volta che la connessione è stabilita, QUIC include un bit di entropia nei pacchetti e richiede che i destinatari inviino un hash delle entropie di cui hanno rivendicato la ricezione, risolvendo così il problema con TCP.
Allo scopo di minimizzare la latenza, i server potrebbero decidere di allentare dinamicamente le restrizioni sull'indirizzo sorgente. Si può immaginare un server che traccia i numeri di richiesta provenienti da diversi indirizzi IP e richiedere solamente i token di indirizzo sorgente quando il contatore delle connessioni "non richieste" eccede il limite globale o per un certo intervallo di IP. Questo potrebbe essere efficace, ma non è chiaro se sia globalmente stabile. Se un gran numero di server QUIC implementasse questa strategia, allora un potenziale attacco DDoS potrebbe essere diviso attraverso essi cosicché la soglia d'attacco non venga raggiunta da nessun server.

4.2 Attacco Replay

In TLS, ogni lato della connessione genera un valore che viene usato per assicurare che l'altra parte sia aggiornata, forzando quest'ultima per includere il valore nella chiave di derivazione. Senza un RTT il client può ancora decidere di includere un valore e così assicurare che il server sia aggiornato, ma quest'ultimo non ha la possibilità di fare lo stesso per il client. Fornire la protezione dagli attacchi di replay senza alcun input da parte del server è molto costoso. Richiede un consistente carico di lavoro al server.
Così QUIC non fornisce protezione da attacchi replay per i dati del client prima di ottenere la prima risposta del server. Sarà compito dell'applicazione assicurare che ogni informazione sia salva se replicata da un attaccante. Ad esempio in Google Chrome solo le richieste GET sono inviate prima della conferma dell'handshake.

4.3 Costi dell'handshake

In TLS il server sceglie i parametri di connessione per ogni connessione basandosi sul supporto pubblicizzato del client per esse. In QUIC le preferenze del server sono totalmente numerate e statiche. Sono in bundle insieme ai valori pubblici Diffie-Hellman in un "server config". Quest'ultimo ha una scadenza ed è firmato dalla chiave privata del server. Dato che il server config è statico, non è necessaria un'operazione di firmatura per ogni connessione, piuttosto che una singola firma sia sufficiente per molte connessioni.
Le chiavi per una connessione sono accettate usando Diffie-Hellman. Il valore Diffie-Hellman del server si trova nel server config e il client ne fornisce uno nel suo primo messaggio di handshake. Poiché il server config dev'essere mantenuto per un po' di tempo in modo da permettere gli handshake 0 RTT, questo pone un limite superiore nella forward security della connessione. Fintanto che il server mantiene segreti Diffie-Hellman per un server config, i dati criptati che usa quel server config potrebbero essere decifrati se venissero persi.
Cosìı QUIC fornisce due livelli di segretezza: i dati iniziali dal client sono cifrati usando il valore Diffie-Hellman nel server config del server, che potrebbe persistere per molti giorni. Subito dopo aver ricevuto la connessione, il server replica con un valore Diffie-Hellman effimero e la connessione viene accordata nuovamente.
Una singola connessione è il solito ambito per la forward security, ma la differenza di sicurezza tra una chiave effimera usata per una singola connessione ed una utilizzata per tutte le connessioni per 60 secondi è trascurabile. Così è possibile ammortizzare la generazione della chiave Diffie-Hellman al server su tutte le connessioni in un piccolo lasso di tempo.
Se assumiamo come S un'operazione di chiave segreta, come P un'operazione di chiave pubblica, come F Diffie-Hellman, punto fisso, moltiplicazione scalare e come A un punto arbitrario, allora la moltiplicazione scalare è:

1. TLS, handshake con security non forward: server, 1S (1100us); client, 1P (34us).
2. TLS, handshake con security forward: server, 1S + 1F + 1A (1301us); client, 1P + 1F + 1A (235us).
3. QUIC: server 2A (100us): client, 1F + 2A + 1P (184us).

Scegliendo le primitive comuni per ognuno di queste (RSA 2048 per le operazioni pubbliche e private, ECDH P-256 per TLS forward security e Curve25519 per QUIC) si ottengono i tempi di esempio nelle parentesi con un i7-3770S. Se QUIC usasse P-256 il tempo del server sarebbe stato di 300us e del client di 385us, quindi una discreta quantità di guadagno si ottiene nella scelta e nell'utilizzo di primitive migliori.

I tassi di ripresa della sessione TLS sono circa del 50%, ma QUIC non include esplicitamente la ripresa della sessione. Tuttavia, può raggiungere molti benefici della ripresa senza alcun tipo di supporto nel protocollo, ma avendo client e server che mantengono una cache dei risultati Diffie-Hellman. Questa cache opzionale elimina l'onere computazionale dell'effettuare l'handshake molte volte sullo stesso server così a lungo che il client non ha cambiato la sua chiave effimera. Assumendo un tasso di ripresa del 50% per TLS ed assumendo che la cache QUIC non fa nulla, allora si ottiene approssimativamente un aumento di cinque volte superiore in relazione a TLS come menzionato nell'introduzione.

CAPITOLO 5

Conclusioni

Come presentato nelle prime pagine di questo documento, il mondo internet e web è in continuo sviluppo e continua evoluzione, per questo motivo è necessario stare al passo con le necessità di quest'epoca. Il protocollo QUIC fa proprio al caso nostro, da una approfondita analisi di quest'ultimo, e dal confronto con ciò che si utilizzava in passato, si è in grado di capire quali e quanti sostanziosi vantaggi porterebbe l'uso di questo protocollo in vasta scala e su tutte le piattaforme. Inizialmente si notano subito le grandi difficoltà di HTTP1, infatti si cerca subito uno sviluppo passando a HTTP2 con SPDY, però ancora basati su TCP come protocollo di trasporto. La vera e propria innovazione si presenta con QUIC tramite UDP.

Rispetto ai protocolli predecessori, QUIC offre diversi vantaggi, presentati precedentemente, che si riassumono in connessioni più rapide e stabili. Tuttavia, il protocollo QUIC non è ancora perfetto, d'altronde è ancora in via di sviluppo e in generale, qualsiasi tecnologia comporta vantaggi e svantaggi, che hanno un maggior peso e rilevanza a seconda della casistica di utilizzo. Per ora, infatti, QUIC deve ancora essere implementato da altri siti e non è pienamente compatibile con firewall e altri strumenti di sicurezza. Questi software, infatti, potrebbero non riconoscere il traffico di QUIC come traffico internet. Così facendo però, quelle trasmissioni non verrebbero controllate, minacciando non poco la tua sicurezza online.

Per concludere si spera che col tempo questo protocollo o comunque protocolli simili, possano entrare a far parte delle connessioni di tutti i giorni, in modo da rendere tutto più stabile e veloce. Questo perché come detto all'inizio dell'elaborato Internet fa parte della routine quotidiana di chiunque, e poter fruire un servizio sempre migliore può essere d'aiuto nello sviluppo di qualsiasi cosa, in qualsiasi ambito.

BIBLIOGRAFIA

1. CoreTech: https://www.coretech.it/it/service/knowledge_base/Sicurezza/Sviluppo-Sicuro/Il-protocollo-HTTP3-e-Quic.php
2. IETF Documentation: https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/edit#heading=h.z2ju224lr24y
3. IETF Documentation: https://docs.google.com/document/d/1i4m7DbrWGgXafHxwl8SwIusY2ELUe8WX258xt2LFxPM/edit
4. IETF Documentation: https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit

Ringraziamenti

Mi sento in dovere di dedicare questa pagina del presente elaborato alle persone che mi hanno supportato nella redazione dello stesso e *durante tutto il percorso fino alla laurea.*

Innanzitutto, ringrazio il mio relatore Cerroni Walter per la sua infinita disponibilità e tempestività ad ogni mia richiesta, nonostante i vari slittamenti. Grazie per avermi fornito ogni materiale utile alla stesura dell'elaborato.

Ringrazio di cuore i miei genitori. Grazie per avermi sempre sostenuto nonostante le tante indecisioni e difficoltà, grazie per avermi aiutato e soprattutto permesso di portare a termine gli studi universitari. Se oggi sono arrivato a questo impensabile traguardo è anche grazie a voi che non avete mai smesso di credere in me. Nonostante il mio caratteraccio, vi voglio bene.

Grazie a mia sorella, che prima di me ha affrontato fino in fondo un percorso di studi, mi hai saputo aiutare, dare forza e coraggio a non mollare mai. Anche se non l'ho mai dimostrato hai tutta la mia stima per essere arrivata dove io non arriverò mai, grazie Ge.

Ringrazio due miei amici prima e colleghi poi, Isma e Dema, con cui ho affrontato questi ultimi anni di difficoltà, tra gioie e tanti dolori siamo stati sempre uniti, e chi prima e chi dopo siamo arrivati tutti alla fine di questo percorso.

Ringrazio i miei amici di una vita, Nico, Edo, Mike e Fanto, ci siete sempre stati e dico sempre, anche quando questo percorso non era nemmeno nei miei piani, mi avete sempre a modo vostro dato manforte, mi avete sempre aiutato a staccare da tutto e farmi divertire, senza che io vi chiedessi nulla. Solo voi sapete di cosa parlo, da sempre e per sempre con voi, grazie fratelli.

Ringrazio la mia fidanzata Camilla, che in questa parte finale mi ha aiutato come non mai, in mezzo a mille vicissitudini, anche quando le cose in questo percorso non andavano sei riuscita a farmi felice e farmi capire il mio valore. Grazie per tutto quello che mi dimostri ogni giorno e per spingermi a credere nelle mie potenzialità.

Infine, voglio ringraziare tutti coloro che ho incontrato in questo percorso e che ho avuto vicino, tutti coloro che mi hanno aiutato in passato, chiunque abbia contribuito in qualsiasi modo a farmi andare avanti e a farmi capire quanto valessi.