HTTP/3 Explained

by Daniel Stenberg

Cuprins

HTTP/3 explicat

Lucrul la această carte a început în martie 2018. Planul este documentarea HTTP/3 și protocolul pe care se bazează: QUIC. De ce, cum funcționează, detaliile protocolului, implementările și multe altele.

Cartea, complet gratuită, este menită a fi un efort cooperativ care să implice pe oricine și pe toată lumea care dorește să ajute.

Precondiții

Se presupune că o persoană care citește această carte are o înțelegere de bază a conexiunilor TCP/IP, conceptelor din spatele HTTP și internetului. Pentru mai detalii și caracteristici ale HTTP/2, recomandăm mai întâi citirea http2 explained.

Autorul

Cartea este începută de Daniel Stenberg. Daniel este creatorul și dezvoltatorul principal al curl, cel mai folosit client HTTP din lume. Daniel lucrează cu (și la) HTTP și protocoale de internet de mai bine două decenii și este autorul http2 explained.

Pagina principală

Pagina principală a acestei cărți se găsește la daniel.haxx.se/http3-explained.

Cum puteți ajuta

Dacă găsiți greșeli, scăpări, erori sau minciuni evidente în acest document, vă rugăm să ne trimiteți o versiune mai bună a paragrafului cu pricina și noi vom face modificările necesare. Vom menționa cum se cuvine pe toată lumea care ajută. Sper să pot îmbunătăți cu timpul acest document.

Este de preferat să trimiteți problemele sau pull requests pe pagina de github a cărții.

Licență

Acest document și întregul lui conținut sunt licențiate cu licența Creative Commons Attribution 4.0.

De ce QUIC

QUIC este un nume, nu un acronim. Este pronunțat la fel ca "quick" în limba engleză.

QUIC este din mai multe puncte de vedere ceea poate fi considerat o modalitate de a construi un nou protocol de transport, de încredere și sigur, pentru protocoale precum HTTP. QUIC rezolvă câteva dintre neajunsurile conexiunilor HTTP/2 peste TCP și TLS. Este următorul pas logic în evoluția comunicării web.

QUIC nu este limitat doar la transportul HTTP. Cel mai mare motiv și motivație care a dus la crearea acestui protocol este dorința de a face ca informațiile și datele, în general, să poată fi livrate mai repede utilizatorilor finali.

Deci, de ce să creezi un protocol de transport nou și de ce este construit peste UDP?

Vă mai aduceți aminte de HTTP/2?

Specificația HTTP/2, RFC 7540, a fost publicată în mai 2015, iar, de atunci, protocolul a fost implementat și publicat la o scară largă pe internet și pe World Wide Web.

La începutul lui 2018, aproape 40% din primele 1000 de pagini de internet din lume rulau pe HTTP/2, aproximativ 70% din toate cererile HTTPS făcute de Firefox primeau răspunsuri HTTP/2, iar toate browsere-le, servere și proxy-uri ofereau suport pentru el.

HTTP/2 rezolvă o multitudine de neajunsuri ale HTTP/1 și, o dată cu introducerea celei de-a doua versiuni a HTTP, utilizatorii se pot opri din a folosi unele metode neoficiale menite să rezolve aceste neajunsuri. Câteva dintre aceste metode sunt destul de dificile pentru programatorii web.

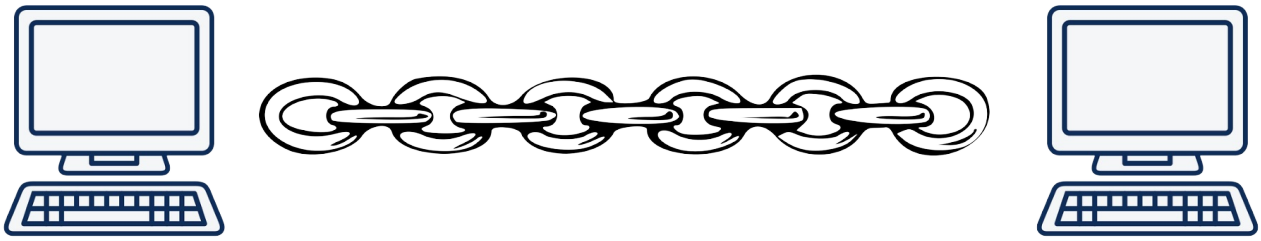
Una dintre funcționalitățile principale ale HTTP/2 este că se folosește de multiplexing, astfel încât mai multe fluxuri logice sunt trimise peste aceeași conexiune TCP. Acest lucru face ca totul să se întâmple mai bine și mai repede. Efortul pentru controlul congestionărilor este mai bun, iar utilizatorii au posibilitatea să folosească conexiunile TCP mult mai bine și, deci, să satureze corespunzător lărgimea de bandă. Astfel, conexiunile TCP sunt active mai multe, ceea ce este bine pentru că ajung la viteza maximă mult mai frecvent ca înainte. Compresia headere-lor ajută conexiunile să folosească mai puțină lărgime de bandă.

Prin HTTP/2, browsere-le folosesc, de obicei, *o conexiune* TCP pentru fiecare host, în loc de cele *șase conexiuni* folosite în trecut. De fapt, tehnicile de contopire și "desharding" ale conexiunilor, folosite cu HTTP/2, pot reduce numărul de conexiuni și mai mult de atât.

HTTP/2 a reparat și problema pe care protocolul o avea cu blocarea vârfului stivei (head of line blocking), în urmă căreia clienții trebuia să aștepte ca prima cerere din stivă să fie finalizată înainte ca următoarea cerere să poată fi trimisă.

Blocarea vârfului stivei în TCP

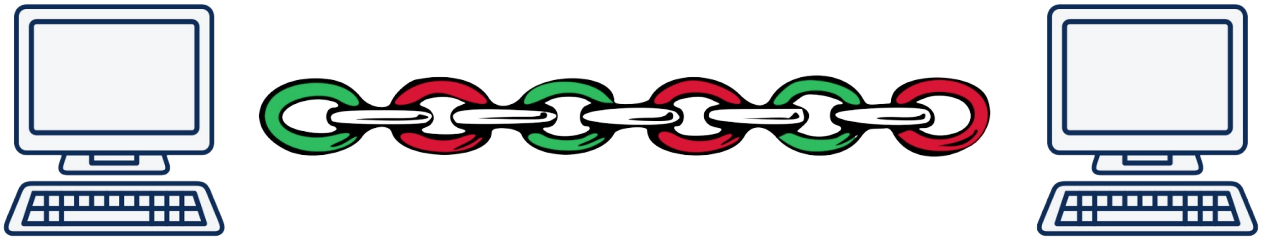
HTTP/2 este implementat peste TCP și folosește mult mai puține conexiuni decât versiunile sale anterioare. TCP este un protocol pentru transferuri fiabile - vă puteți gândi la el ca la un lanț imaginar între două sisteme. Ce este trimis prin rețea de la un capăt va ajunge, eventual, la celălalt capăt. Altfel, conexiunea va fi întreruptă.



Cu HTTP/2, browsere-le normale fac zeci sau sute de transferuri în paralel peste o singură conexiune TCP.

Dacă un singur pachet este ignorat, sau pierdut în rețea undeva între două componente care "vorbesc" HTTP/2, înseamnă că toată conexiunea TCP este oprită până când pachetul pierdut este retransmis și își regăsește calea către destinatar. De vreme ce TCP este un "lanț", dacă o za a acestui lanț dispare subit, tot ce vine după ea trebuie să aștepte.

O ilustrație care se folosește de metafora "lanțului" pentru a transmite două fluxuri, unul roșu și unul verde, printr-o singură conexiune:



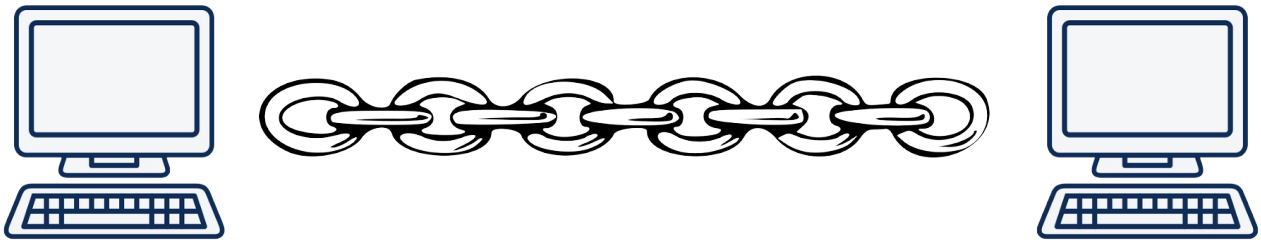
Blocarea vârfului de stivă TCP:

Pe măsură ce rata de pierdere a pachetelor crește, HTTP/2 se mișcă din ce în ce mai rău. La o pieredere de 2% (care reprezintă o calitate groaznică a rețelei), testele au arătat că utilizatorii HTTP/1 sunt mai avantajați, deoarece au până la 6 conexiune TCP deschise prin care se distribuie rata de pieredere a pachetelor. Asta înseamnă că, dacă o conexiune pierde un pachet, celelalte pot continua.

O rezolvare a acestei probleme nu este ușoară sau, poate chiar posibilă, atunci când se folosește TCP.

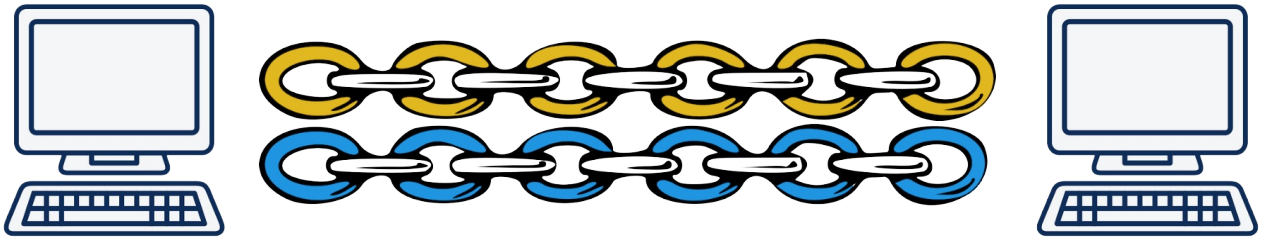
Fluxurile independente evită blocarea

Atunci când este folosit QUIC, se face, de asemenea, o organizare a unei conexiuni între două sisteme, ceea ce face conexiunea să fie sigură și datele să fie livrate fiabil.



Două fluxuri separate trimise prin această conexiune sunt prelucrate independent, astfel că, dacă o parte lipsește dintr-unul dintre fluxuri, numai acel flux și acel lanț trebuie să se oprească și să aștepte ca acea bucată să fie retrimisă.

Conceptul este ilustrat aici cu un flux galben și unul albastru, trimise între două sisteme:



TCP sau UDP

Dacă nu putem să rezolvăm problema blocării vârfului de stivă din TCP, atunci, teoretic, ar trebui să putem să creăm un nou protocol de transport, diferit de UDP și TCP. Sau, poate chiar să folosim SCTP, care este un protocol de transport standardizat de IETF în [RFC4960] (https://tools.ietf.org/html/rfc4960) și care include câteva dintre caracteristicile dorite.

Totuși, în ultimii ani, eforturile de a crea noi protocoale de transport au fost oprite aproape în totalitate de dificultățile în a le publica pe internet. Publicarea de noi protocoale este limitată de multe firewall-uri, NAT-uri, routere și alte middlebox-uri care permit doar conexiuni TCP și UDP între utilizatori și servere-le cu care au nevoie să comunicae. Introducerea unui nou protocol de transport face ca N% dintre conexiuni să eșueze pentru că sunt blocate de sisteme care văd că nu sunt UDP sau TCP, deci le consideră dăunătoare sau, pur și simplu, greșite. Rata de eșec de N% este considerată, deseori, prea mare pentru a merita efortul.

În plus, schimbările în layer-ul protocolului de transport al stivei de rețea, înseamnă, de obicei, protocoale implementate de kernel-urile sistemelor. Actualizarea și publicarea kernel-urilor sistemelor de operare este un proces încet, care presupune efort semnificativ. Multe îmbunătățiri ale TCP, standardizate de IETF, nu sunt publicate la scară largă sau folosite pentru că nu sunt implementate, în general.

De ce nu SCTP-peste-UDP?

SCTP este un protocol de transport fiabil cu fluxuri șim pentru WebRTC, există deja implementări care îl folosesc peste UDP.

SCTP nu a fost considerat îndeajuns de bun față de QUIC din cauza câtorva motive, printre care:

- SCTP nu rezolvă problema blocării vârfului de stivă pentru fluxuri
- SCTP are nevoie ca numărul de fluxuri să fie predefinit la momentul stabilirii conexiunii
- SCTP nu are o istorie puternică cu TLS/securitate
- SCTP funcționează cu un handshake în 4 pași, în timp ce QUIC oferă 0-RTT
- QUIC este un flux de octeți ca TCP, în timp ce SCTP se folosește de mesaje
- Conexiunile QUIC pot migra între adrese IP, în timp ce ale SCTP nu

Pentru mai multe detalii legate de diferențe, vedeți A Comparison between SCTP and QUIC.

Osificarea

Internetul este o rețea de rețele. Există dispozitive instalate pe internet în multe locuri diferite care se asigură că această rețea de rețele funcționează cum ar trebui. Aceste dispozitive, sisteme care sunt distribuite pe rețea, sunt ceea ce numim middlebox-uri - echipamente care sunt poziționate undeva între două sisteme și care sunt părțile principale implicate într-un transfer tradițional pe rețea.

Aceste echipamente servesc mai multe scopuri diferite, dar cred că putem spune cu siguranță că sunt puse acolo pentru că cineva crede că trebuie să fie acolo astfel încât lucrurile să funcționeze cum trebuie.

Middlebox-uri redirecționează pachete IP între rețele, blochează traficul răuvoitor, fac NAT (Network Address Translation - traducerea adreselor de rețea), îmbunătățesc performanța, unele încearcă să spioneze traficul care trece prin ele și așa mai departe.

Ca să își poată îndeplini sarcinile, aceste sisteme trebuie să aibă cunoștințe de rețelistică și despre protocoalele pe care trebuie să le monitorizeze sau să le modifice. Pentru acest scop, rulează aplicații software, iar aceste aplicații nu sunt mereu actualizate frecvent.

Chiar dacă sunt componente care fac internetul să funcționeze, de multe ori nu țin pasul cu ultimele tehnologii. Mijlocul rețelei, de obicei, nu se mișcă la fel de repede ca marginile - aplicațiile-client și server-le din lume.

Protocoalele de rețea pe care aceste echipamente ar vrea să le inspecteze și despre care ar vrea să stie dacă sunt OK sau nu au această problemă: aceste echipamente au fost publicate cu ceva timp în urmă, când protocoalele avea un set de caracteristici, valabil la acel moment. Introducerea unor noi funcționalități sau schimbări în comportament care nu erau cunoscute atunci sunt considerate rele sau interzise de aceste echipamente. Astfel de trafic poate fi oprit sau întârziat până la punctul în care utilizatorii nu își mai doresc să folosească funcționalitățile.

Asta se numește "osificarea protocoalelor".

Schimbările TCP "suferă", de asemenea, de osificare: unele echipamente între o aplicație-client și server o să identifice opțiuni noi TCP pe care nu o să le recunoscă și o să blocheze întreaga conexiune. Dacă le este permis să detecteze detalii legate de protocol, sistemele învață cum ar trebui să se comporte protocoalele de obicei și, cu timpul, devin imposibil de schimbat.

Singura metodă eficientă de a preveni osificarea este criptarea a unei bucăți cât mai mari din comunicare pe cât posibil, astfel încât middlebox-urile să nu poată vedea detalii ale protocoalelor care trec prin ele.

Securitate

QUIC oferă întotdeauna securitate. Nu există o versiune în clar a protocolului, așa că, pentru a negocia o conexiune QUIC este nevoie de criptografie și securitate bazată pe TLS 1.3. Cum am menționat mai sus, asta previne osificarea, la fel ca alte tipuri de blocaje sau tratamente speciale, asigurându-se, în același timp că QUIC folosește toate proprietățile HTTPS pe care utilizatorii au ajuns să și le dorească și să le aștepte.

Există doar câteva pachete inițiale trimise în clar, la handshake, înainte ca protocoalele criptografice să fie negociate.

Informații din timp

QUIC oferă handshake-uri 0-RTT și 1-RTT care reduc timpul necesar negocierii și stabilirii unei noi conexiuni. În comparație, TCP folosește un handshake în trei pași.

În plus, QUIC oferă suport pentru "informații din timp", care îi dau posibilitatea să transmită mai multe informații și este folosit mult mai ușor decât TCP Fast Open.

Folosindu-se conceptul de fluxuri, o nouă conexiune logică poate fi deschisă către același host fără a fi nevoie să se aștepte terminarea uneia existente.

TCP Fast Open este problematic

TCP Fast Open a fost publicat ca RFC 7413 în decembrie 2014, iar specificația descrie cum pot programele transmite informații către un server, livrându-le deja în primul pachet TCP SYN.

Implementarea acestei componente în realitate a durat mult timp și este plină de probleme, chiar și acum în 2018. Cei care implementează specificația TCP au avut probleme și, astfel, și programele care încearcă să folosească această funcționalitate au avut la fel - în primul rând în a ști ce sisteme de operare să activeze și apoi cum să anuleze schimbările atunci când au apărut probleme. Au fost identificate câteva rețele care au interferat cu traficul TCP Fast Open, stricând, deci, permanent, astfel de TCP handshakes.

Procesul

Prima versiune a protocolului QUIC a fost proiectată de Jim Rosking de la Google, și a fost implementată prima oară în 2012. A fost anunțată public în 2013 când Google au extins experimentele.

Atunci, QUIC reprezenta un acronim pentru "Quick UDP Internet Connections" (Conexiuni UDP Rapide pe Internet)

Google au implementat și publicat, ulterior, protocolul în browser-ul popular pe care îl dezvoltă (Chrome) și în serviciile server-side proprii, la fel de populare (motorul de căutare Google, gmail, youtube și altele). Au dezvoltat noi versiuni repede și, cu timpul, au demonstrat fiabilitatea conceptului pentru o plajă largă de utilizatori.

În iunie 2015, prima schiță pentru QUIC a fost trimisă către IETF pentru standardizare, dar a durat până la sfârșitul lui 2016 pentru ca un grup de lucru dedicat să fie aprobat și să înceapă. După aceea, lucrul a început aproape imediat, mai mulți participant declarându-și interesul.

În 2017, numele citate de inginerii Google care lucrează la QUIC menționau că aproximativ 7% din *tot* traficul de pe internet folosește deja acest protocol. Versiunea Google a protocolului.

IETF

Grupul de lucru QUIC, care a fost organizat pentru a standardiza protocolul în interiorul IETF, a decis repede că acesta ar trebui să poată să transfere și alte protocoale în afară de "doar" HTTP. Google-QUIC nu a transferat niciodată altceva în afară de HTTP. În practică, a transferat frame-uri HTTP/2, folosindu-se de sintaxa pentru frame-uri a HTTP/2.

S-a spus, de asemenea, că IETF-QUIC ar trebui să își bazeze criptarea și securitatea pe TLS 1.3, în loc de implementarea "proprie" folosită de Google-QUIC.

Ca să îndeplinească cerință de trimite-mai-mult-decât-HTTP, arhitectura protocolului QUIC al IETF a fost separată în două părți: transferul QUIC și "HTTP peste QUIC". A doua parte este numită uneori și "hq".

Această despărțire, chiar dacă sună inofensivă, a făcut ca specificația IETF-QUIC să difere destul de mult de cea originală, Google-QUIC.

Grupul de lucru a decis repede, totuși, că, pentru a putea avea puterea de concentrare și abilitatea de a livra versiunea 1 a QUIC, se va ocupa de livrarea HTTP, lăsând transferurile non-HTTP pe mai târziu.

În martie 2018, când am început să lucrez la această carte, planul era ca specificația finală a versiunii 1 să fie publicată în noiembrie 2018. Această dată a fost amânată până în iulie 2019.

În timp ce munca la IETF-QUIC a progresat, echipa de la Google a implementat deja detalii din versiunea IETF și a început să migreze versiunea proprie a protocolului către ce ar putea să devină versiunea IETF. Google au continuat să își folosească propria versiune a QUIC în browser și servicii.

[Majoritatea implementărilor în curs de desfășurare] (https://github.com/quicwg/base-drafts/wiki/Implementations) au decis să se concentreze pe versiunea IETF și nu sunt compatibile cu versiunea Google.

Experiența acumulată cu HTTP/2

Specificația HTTP/2, RFC 7540, a fost publicată în mai 2015, cu doar o lună înainte ca QUIC să fie prezentat IETF pentru prima oară.

Cu HTTP/2, fundația pentru schimbarea HTTP a fost deja pusă și grupul de lucru a adoptat o mentalitate potrivită pentru trecerea la versiuni noi de HTTP mult mai repede ca înainte (trecerea de la HTTP/1 la 2 a durat aproximativ 16 ani).

După lansarea HTTP/2, utilizatorii și creatorii de software s-au obșinuit cu ideea că HTTP nu mai poate fi considerat a fi un protocol finalizat.

HTTP-peste-QUIC a fost redenumit HTTP/2 în noiembrie 2018.

Starea curentă

Grupul de lucru QUIC a muncit asiduu începând cu sfârșitul lui 2016 pentru a specifica particularitățile protocolului, iar planul este ca totul să fie gata până în iulie 2019.

La timpul scrierii acestui text, în noiembrie 2018, încă nu există teste de interoperabilitate la scară largă pentru HTTP/3 - există doar două implementări majore, și niciuna dintre ele nu este făcută de un browser sau de un software open-source popular.

Există 15 [implementări QUIC] (https://github.com/curl/curl/wiki/QUIC-implementation) înregistrate în paginile wiki ale grupului de lucru, dar foarte puține dintre ele pot funcționa pe ultimele modificări ale schiței specificației.

Implementarea QUIC nu este ușoară și protocolul a fost actualizat încontinuu, schimbându-se chiar și la această dată.

Servere

Nu a fost făcut niciun anunț public de implementare a QUIC de către Apache sau ngnix.

Programele-client

Niciunul dintre browsere-le mari nu a publicat până acum vreo versiune, în orice stare, care să poată să ruleze versiunea IETF a QUIC sau HTTP/3.

Google Chrome este publicat deja de mulți ani cu o implementare funcțională a propriei implementări QUIC, dar funcționalitățile ei nu sunt compatibile cu protocolul dezvoltat de IETF, iar implementarea lor pentru HTTP/3 este diferită.

Obstacolele din calea implementării

Pentru QUIC, s-a decis folosirea TLS 1.3 ca fundație pentru criptare și layer-ul de securitate (pentru a evita inventarea a ceva nou, comunicarea bazându-se, în schimb, pe un protocul de încredere deja existent. Totuși, făcând asta, grupul de lucru a decis, de asemenea, ca pentru a optimiza folosirea TLS în QUIC, acesta ar trebui să folosească numai "mesaje TLS" și nu "înregistrări TLS".

Chiar dacă sună ca o schimbare inofensivă, această decizie a pus piedici semnificative multora dintre cei care implementează QUIC. Librăriile software care implementează TLS 1.3 pur și simplu nu au destule API-uri pe care să le expună pentru ca QUIC să le acceseze. În timp ce câțiva dintre cei care implementează vin din organizații mai mare care lucrează la propria stivă TLS în paralel, acest lucru nu se aplică la toată lumea.

Bine-cunoscutul OpenSSL, de exemplu, nu implementează niciun API în acest moment și nu a făcut niciun anunț care să exprime planuri în această direcție în viitorul apropiat (afirmație valabilă în noiembrie 2018).

Asta va duce, eventual, și la obstacole în calea implementării din cauză că QUIC va avea nevoie fie să se bazeze pe alte librării TLS, fie să folosească o versiune separată și modificată a OpenSSL, fie să ceară o actualizare a viitoarelor versiune OpenSSL.

Kernel-uri și capacitatea de procesare

Și Google și Facebook au menționat că implementările proprii la scară largă ale QUIC vor necesita aproximativ dublul capacității de procesare (CPU) față de același trafic servit prin HTTP/2 peste TLS.

Câteva explicatii pentru asta sunt:

- componenta UDP, mai ales în Linux, nu este deloc optimizată la fel cum este stiva TCP, deoarece nu a fost folosită în

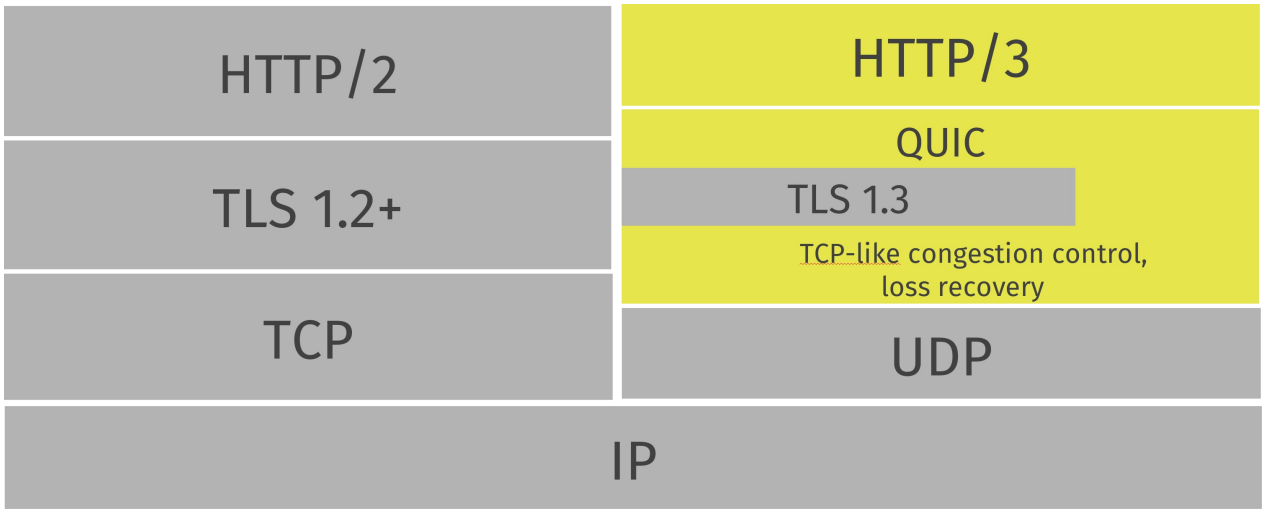
mod normal pentru transferuri de mare viteză.
- Pentru TCP și TLS există descărcare către hardware (offloading to hardware), dar pentru UDP exemplele sunt mult mai rare, iar pentru QUIC este practic inexistentă.

Există motive să credem că performanța și cerințele CPU se vor îmbunătăți cu timpul.

Particularitățile protocolului

O privire de ansamblu asupra protocolului QUIC.

Mai jos, în stânga, sunt ilustrate componentele unei rețele HTTP/2, iar în dreapta componentele rețelei QUIC când este folosită pentru a face un transfer HTTP.



Protocolul de transfer peste UDP

QUIC este un protocol de transfer implementat peste UDP. Dacă vă urmăriți traficul rețelei, o să vedeți că QUIC apare ca pachete UDP.

Tot bazându-se pe UDP, folosește apoi port-uri UDP pentru a identifica servicii de rețea specifice care rulează la o anumită adresă IP.

Toate implementările QUIC cunoscute sunt, acum, în user-space, ceea ce permite o evoluție mult mai rapidă decât permit implementările kernel-space.

Va funcționa?

Există organizații și alte structuri de rețele care blochează traficul UDP pe alte porturi decât 53 (folosit pentru DNS). Altele limitează astfel de date în feluri care fac ca QUIC să funcționeze mai greu ca protocoalele bazate pe TCP. Nu există limite la ce pot face unii operatori.

În viitorul predictibil, toate transferurile bazate pe QUIC va trebui să poată să regreseze (fall-back) la alte alternative (bazate pe TCP). Inginerii Google au menționat în trecut rate de eșec de ordinul numerelor cu o singură cifră.

Se va îmbunătăți?

Șansele sunt ca, dacă QUIC se dovedește să fie o adiție valoroasă pentru internet, oamenii să vrea să îl folosească și să funcționeze în rețelele lor, iar atunci companiile își vor reconsidera obstacolele. În timpul anilor de dezvoltare, rata de succes în a stabili și folosi conexiuni QUIC pe internet a crescut.

Transferuri fiabile de date

În timp ce UDP nu este un transfer fiabil, QUIC adaugă un layer deasupra UDP care introduce această fiabilitate. Oferă retransmiterea pachetelor, controlul congestiilor, pacing și toate celelalte funcționalități prezente în TCP.

Datele trimise prin QUIC de la un capăt vor apărea în celălalt capăt mai devreme sau mai târziu, atâta timp cât conexiunea este menținută.

Mai multe fluxuri în conexiuni

Similar cu SCTP, SSH și HTTP/2, QUIC oferă fluxuri logice separate în interiorul conexiunilor fizice - in număr de fluxuri paralele care pot transfera informații simultan peste o singură conexiune fără să afecteze celelalte fluxuri.

O conexiune este negociere între două sisteme, similar cu modul de funcționare al TCP. O conexiune QUIC este făcută pe un port UDP și o adresă de IP, dar, odată stabilită, conexiunea este asociată cu propriul "ID de conexiune".

Peste o conexiune stabilită, oricare dintre părți poate crea fluxuri și să trimită informații la celălalt capăt. Fluxurile sunt livrate în ordinea în care sunt trimise și sunt fiabile, dar este posibil ca fluxuri diferite să fie livrate într-o altă ordine.

QUIC oferă control și asupra conexiunilor și a fluxurilor.

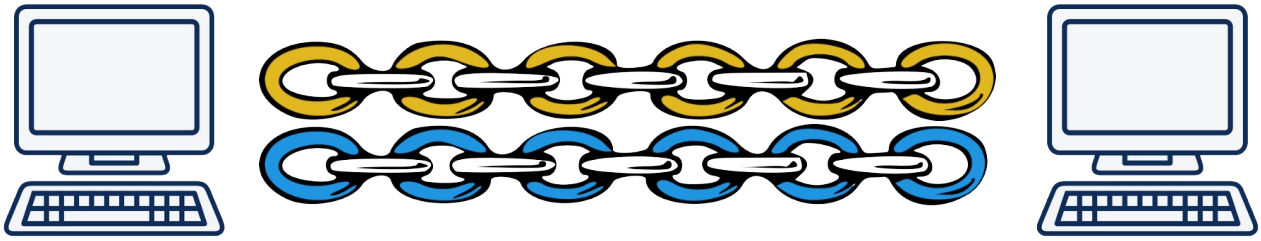
Puteți citi mai multe detalii în secțiunile conexiuni și fluxuri.

În aceeași ordine

QUIC garantează livrarea în aceeași ordine a fluxurilor în sine, dar nu și între fluxuri. Asta înseamnă că fiecare flux va trimite informații și va menține ordinea informațiilor, dar fluxurile pot ajunge la destinație în altă ordine decât cea în care au fost trimise de la sursă.

De exemplu: fluxul A și B suntr transferate de la server către un program. Fluxul A este început primul și apoi fluxul B. În QUIC, un pachet pierdut afectează doar fluxul căruia îi aparține. Dacă fluxul A pierde un pachet, dar fluxul B nu, fluxul B își poate continua și finaliza transferul, în timp ce pachetul pierdut de fluxul A este retransmis. Acest lucru nu era posibil cu HTTP/2.

Ilustrat aici cu două fluxuri, unul galben și unul albastru, trimise între două sisteme QUIC, peste o singură conexiune. Fluxurile sunt independente și pot ajunge la destinație într-o ordine diferită, dar fiecare dintre ele este livrat cu siguranță.



Handshakes rapide

QUIC oferă stabilirea conexiunilor cu 0-RTT și 1-RTT, însemnând că, într-un scenariu ideal, QUIC nu face round-trip-uri în plus atunci când stabilește o conexiune. Cea mai rapidă metodă dintre cele două, handshake-ul 0-RTT, funcționează doar dacă a fost stabilită în trecut o conexiune la același host și o cheie secretă de la acea conexiune a fost salvată în cache.

Informații din timp

QUIC permite unui sistem-client să includă informații deja în handshake-ul 0-RTT. Această funcționalitate îi permite sistemului-client să livreze informații celuilalt sistem pe cât de repede este posibil, iar asta înseamnă, bineînțeles, că server-ul va răspunde și va trimite informații mai repede.

TLS 1.3

Securitatea transferurilor în QUIC este implementată folosing TLS 1.3 (RFC 8446) și nu există conexiuni QUIC necriptate.

TLS 1.3 are câteva avantaje asupra versiunilor mai vechi de TLS, dar un motiv principal de a-l folosi în QUIC a fost că versiunea 1.3 a micșorat numărul de handshakes necesare. Asta reduce timpii de încărcare ai protocolului.

Vechea versiune QUIC de la Google folosea un algoritm propriu de criptare.

Nivelurile de aplicație și de transfer

Protocolul IETF QUIC este un protocol de transfer, peste care pot fi folosite alte protocoale. Layer-ul inițial de aplicație al protocolului este HTTP/3 (h3).

Layer-ul de transfer susține și conexiuni și fluxuri.

Versiunea inițială de QUIC a Google lipea transferul și HTTP împreună într-un singur protocol "bun la toate" și reprezenta un protocol specializat de trimitere-frame-uri-http/3-peste-udp.

HTTP/3 peste QUIC

Layer-ul HTTP face transferuri tip-HTTP, inclusiv compresia headere-lor HTTP folosind QPACK - similar cu compresia HTTP/2 numită HPACK.

Algoritmul HPACK se bazează pe livrarea *ordonată* a fluxurilor, astfel că nu a fost posibil să fie refolosit pentru HTTP/3 fără modificări, de vreme ce QUIC oferă fluxuri care pot fi livrate în orice ordine. QPACK poate fi considerat a fi versiunea adaptă pentru QUIC a HPACK.

Non-HTTP peste QUIC

Munca legată de trimiterea altor protocoale, în afară de HTTP, peste QUIC a fost amânată până după publicarea versiunii 1.

Cum funcționează QUIC

Fără a explica exact biți și baiții de pe rețea, această secțiunea descrie cum funcționează conceptele de bază ale protocolului de transfer QUIC. Dacă doriți să aveți propria implementare a QUIC, această descriere vă va oferi o înțelegere generală, dar, pentru toate detaliile, citiți ciornele și RFC-urile IETF.

1. Stabilirea unei conexiuni
2. ... care include securitatea TLS
3. Și apoi folosiți fluxuri

Conexiuni

O conexiune QUIC este o singură conversație între două sisteme QUIC. Stabilirea conexiunii QUIC combină negocierea versiunii cu handshake-urile criptografice și de transport pentru a reduce timpii de încărcare.

Pentru a trimite efectiv informații prin acest tip de conexiune, unul sau mai multe fluxuri trebuie să fie create și folosite.

ID-ul conexiunii

Fiecare conexiune deține un set de identificatori, sau ID-uri de conexiune, iar fiecare dintre ei poate fi folosit pentru a identifica o conexiune. ID-urile de conexiune sunt alese independent de capetele conexiunii; fiecare capăt alege ID-urile pe care le folosește celălalt capăt.

Funcționalitatea principală a acestor ID-uri de conexiune este să se asigure că schimbările de adresare ale protocoalelor din layere-le de dedesubt (UDP, IP și mai jos) nu cauzează ca pachetele unei conexiuni QUIC să fie livrate către un alt sistem.

Profitând de ID-uri, conexiunile pot, deci, migra între adrese de IP și interfețe de rețea în moduri în care TCP nu putea. De exemplu, migrația permite ca download-uri în curs de desfășurare să se mute de la o conexiune la o rețea celulară la una mai rapidă peste wifi atunci când utilizatorul își aduce device-ul într-un loc care oferă wifi. Similar, download poate continua peste rețeaua celulară dacă cea wifi devine indisponibilă.

Port-urile

QUIC este construit peste UDP, așa că un număr de port pe 16 biți este folosit pentru a diferenția conexiunile primite.

Negocierea versiunii

O cerere de conexiune QUIC venită de la o aplicație-client o să îi spună server-ului ce versiune de protocol QUIC vrea să folosească, iar server-ul va răspunde cu o listă de versiuni din care aplicația poate alege.

Conexiunile folosesc TLS

Imediat după primul pachet care stabilește o conexiune, inițiatorul trimite un frame criptografic care începe stabilirea handshake-ului pentru layer-ul de securitate.

Nu există nicio cale de a evita acest lucru sau de a evita folosirea TLS pentru conexiune QUIC. Protocolul este proiectat pentru a îngreuna manipulările middlebox-urilor, cu scopul de a evita osificarea protocolului.

Fluxuri

Fluxurile în QUIC oferă o abstractizare simplă a unui flux de octeți ordonat.

Există două tipuri de fluxuri de bază în QUIC:

- Fluxuri unidirecționale care transferă informațiile într-o singură direcție: de la inițiatorul fluxului la receptor.
- Fluxuri bidirecționale care permit ca informațiile să fie transmise în ambele direcții.

Oricare tip de flux poate fi creat de oricare dintre capetele conexiunii, poate trimite informații amestecat cu alte fluxuri și poate fi anulat.

Pentru a trimite date printr-o conexiune QUIC, unul sau mai multe fluxuri sunt folosite.

Controlul debitului

Debitul fluxurilor este controlat individual, permițqnd ca unul dintre capete să limiteze memoria alocată și să aplice presiune înapoi. Crearea de fluxuri are, de asemenea, debitul controlat, cu fiecare dintre sisteme declarând ID-ul maxim pe care este dispus să îl accepte la un moment dat.

Identificatorii fluxurilor

Fluxurile sunt identificate de un integer unsigned pe 62 de octeți, numit și ID-ul fluxului (Stream ID). Cei mai puțini semnificativi doi octeți ai ID-ului sunt folosiți pentru identificarea tipului fluxului (unidirecțional sau bidirecțional) și inițiatorul fluxului.

Cel mai puțin semnificativ bit (0x1) al id-ului identifică inițiatorul fluxului. Aplicațiile-client inițiază fluxuri folosind numere pare (acelea cu cel mai puțin semnificativ bit setat pe 0); servere-le inițiază fluxuri folosind numere impare (cu bitul setat pe 1);

Al doilea cel mai puțin semnificativ bit (0x2) al ID-ul fluxului diferențiază între fluxuri unidirecționale și bidirecționale. Fluxurile unidirecționale au acest bit setat pe 1, iar fluxurile bidirecționale au acest bit setat pe 0.

Paralelismul fluxurilor

QUIC permite ca un număr arbitrar de fluxuri să funcționeze în paralel. Un capăt al conexiunii limitează numărul de fluxuri paralele active prin limitarea ID-ului maxim pe care îl poate avea un flux.

ID-ul maxim al fluxului este specific fiecărui capăt și se aplică numai sistemului care primește această setare.

Trimiterea și primirea de informații

Sistemele folosesc fluxuri ca să trimită și să primească informații. Acesta este, în fond, scopul lor principal. Fluxurile sunt abstracții de octeți ordonați. Fluxurile separate nu sunt, totuși, livrate în ordinea originală.

Prioritizarea fluxurilor

Multiplexing-ul fluxurilor are un efect semnificativ asupra perfomanțelor aplicației dacă resursele alocate fluxului nu sunt corect prioritizate. Experiența cu alte protocoale care fac multiplexing, cum ar fi HTTP/2, arată că strategiile de prioritizare eficiente au un efect pozitiv asupra performanței.

QUIC, în sine, nu oferă cadre pentru prioritizarea informației. În schimb, se bazează pe primirea prioritizării de la aplicația care folosește QUIC. Protocoalele care folosesc QUIC au posibilitatea de a defini orice schemă de prioritizare care se potrivește semanticii proprii.

Când HTTP/3 este folosit peste QUIC, prioritizarea este făcută în layer-ul HTTP.

0-RTT

Pentru a reduce timpul necesar stabilirii unei noi conexiuni, o aplicație-client, care s-a conectat în trecut la un server, are posibilitatea să salveze în cache anumiți parametri și, ulterior, să stabilească o conexiune **0-RTT** cu server-ul. Asta permite aplicației-client să trimită informații imediat, fără să aștepte ca handshake-ul să fie finalizat.

Bit-ul rotativ (Spin Bit)

Una dintre cele mai lungi discuții de design din cadrul grupului de lucru QUIC, care a fost subiectul a câtorva sute de ore de e-mail-uri și ore de discuție, se referă la un singur bit: Bit-ul rotativ.

Susținătorii acestui bit insită e nevoie ca operatorii și oamenii, care sunt pe drumul dintre două capete ale unei conexiuni QUIC, să poată să măsoare timpii de încărcare.

Opozanților acestei funcționalități nu le plac potențialele pierderi de informații.

Rotirea unui bit

Ambele capete, aplicația-client și server-ul, mențin o valoare de rotație, 0 sau 1, pentru fiecare conexiune QUIC, și setează valoarea potrivită pe bit-ul rotativ în pachetele pe care le trimite.

Ambele părți, apoi, trimit pachete cu acel bit rotatic setat pe aceeași valoare, pe toată durata unui drum dus-întors și apoi comută valoarea. Efectur este, apoi, un puls de unu și zero în câmpul acelui bit, pe care observatorii îl pot monitoriza.

Această măsurătoare funcționează doar când expeditorului nu îi sunt limitate nici aplicația nici debitul, iar reordonarea pachetelor în rețea, poate face ca informațiile neclare.

User-space

Implementarea unui protocol de transfer în user-space ajută la iterarea rapidă a protocolului, deoarece evoluția protocolului este mult mai ușoară fără necesitatea ca aplicațiile-client și servere-le să își actualizeze kernel-ul sistemului de operare pentru a publica noi versiuni.

Nu există nimic inerent în QUIC care să îl împiedice să fie implementat și oferit prin kernel-urile sistemelor de operare în viitor, dacă cineva va considera că asta este o idee bună.

Mai multe implementări

Un efect evident al implementării protocolului de transport în user-space este că ne putem aștepta să vedem multe implementări independente.

Diferitele aplicații vor include probabil (sau vor fi construite ca un strat deasupra a) diferite implementări HTTP/3 și QUIC în viitorul previzibil.

API

Unul dintre motivele succesului pentru TCP-ul obșinuit și programele care îl folosesc este că folosesc un API de socket-uri standardizat. Are funcționalități bine definite și poți muta programe între sisteme de operare diferite, iar TCP funcționează la fel.

QUIC încă nu a ajuns la acest nivel. Nu există un standard de API pentru QUIC.

Cu QUIC, ai nevoie să alegi una dintre librăriile deja implementate și să rămâi la API-ul ei. Asta face, la un anumit nivel, ca aplicațiile să fie "legate" de o singură librărie. Schimbarea la o altă librărie înseamnă un alt API, iar asta poate implica foarte multă muncă.

De asemenea, de vreme ce QUIC este implementat, de obicei, în user-space, nu poate să extindă pur și simplu API-ul socket-urilor sau să arate la fel ca funcționalitățile existente TCP și UDP. Folosirea QUIC va însemna folosirea unui alt API decât cel al socket-urilor.

HTTP/3

După cum am menționat mai devreme, primul și principalul protocol de transfer peste QUIC este HTTP.

La fel cum HTTP/2 a fost introdus pentru a transfera HTTP într-un mod complet nou, HTTP/3 introduce, din nou, o modalitate nouă de a transfera HTTP prin rețea.

HTTP încă păstrează aceleași paradigme și concepte ca înainte. Încă există headere și un body, există o cerere și un răspuns. Există verbe, cookie-uri și caching. Ce se schimbă, în principal, în HTTP/3 este cu sunt transmișii biții către partea cealaltă a comunicației.

Ca să se poată face HTTP peste QUIC, a fost nevoie de anumite schimbări, iar asta numim acum HTTP/3. Aceste schimbări au fost necesare din cauza naturii diferite a QUIC față de TCP. Aceste schimbări incluc:

- în QUIC, fluxurile sunt susținute de transferul în sine, în timp ce în HTTP/2 fluxurile sunt făcute în cadrul layer-ului HTTP.
- Din cauză că fluxurile sunt independente unul față de celălalt, protocolul pentru compresia headere-lor folosit pentru HTTP/2 nu a putut fi folosit fără a crea o situație de blocare a capului de stivă.
- Fluxurile QUIC sunt puțin diferite de fluxurile HTTP/2. Secțiunea HTTP/3 va detalia, oarecum, asta.

Adresele HTTPS://

HTTP/3 va rula folosind adrese `HTTPS://` . Lumea este plină de aceste URL-uri și a fost stabilit nepractic și cu totul nerezonabil să se introducă o nouă schemă de URL pentru noul protocol. Cum HTTP/2 nu a avut nevoie de o nouă schemă, nici HTTP/3 nu va avea nevoie.

Complexitatea adăugată în situația HTTP/3 este, totuși că, în timp ce HTTP/2 a fost o modalitate complet nouă de a transfera HTTP, era în continuare bazat pe TLS și TCP, la fel cum era și HTTP/1. Faptul că HTTP/3 este făcut peste QUIC schimbă lucurile în câteva aspecte importante.

Adresele tradiționale, în text clar, `HTTP://` for fi lăsate așa cum sunt și, pe măsură ce avansăm într-un viitor cu transferuri mai sigure, vor fi folosite din ce în ce mai puțin frecvent. Cererile către astfel de URL-uri pur și simplu nu vor fi actualizate să folosească HTTP/3. În realitate, sunt rareori actualizate să folosească chiar și HTTP/2, dar din cu totul alte motive.

Conexiunea inițială

Prima conexiune către o gazdă nouă, nefolosită în trecut pentru un URL HTTPS://, va fi făcută, probabil, peste TCP (posibil, în plus față de o încercare paralelă de conectare prin QUIC). Gazda poate fi un server tradițional, fără suport pentru QUIC sau poate exista un middlebox care ridică obstacole care previn successul conexiunii QUIC.

O aplicație-client și un server moderne vor negocia, presupus, HTTP/2 în primul handsjake. Când conexiunea a fost stabilită și server-ul răspunde la o cerere HTTP de la client, el îi poate spune clientului despre ce capacilități are și despre o preferință pentru HTTP/3.

Alt-svc

Header-ul pentru serviciul alternativ (Alt-svc) și cadrul corespondent din HTTP/2, `ALT-SVC` nu au fost create special pentru QUIC sau HTTP/3. Ele sunt parte a unui sistem deja proiectat și creat pentru ca un server să îi poată spune unei aplicații-client: *"uite, rulez același serviciu pe ACEASTĂ GAZDĂ, folosind ACEST PROTOCOL, pe ACEST PORT"*. Vedeți mai multe detalii în RFC 7838.

Un client care primește un astfel de răspuns Alt-svc este, apoi, sfătuit, dacă implementează și dorește, să se conecteze la cealaltă gazdă folosită în paralel în fundal - folosind protocolul specificat - și, dacă schimbul reușește, să își schimbe toate operațiile pe acelea în loc de conexiunea inițială.

Dacă conexiunea inițială folosește HTTP/2 sau chiar HTTP/1, server-ul poate răspunde și să îi spună clientului că se poate conecta înapoi și să încerce HTTP/3. Poate fi către aceeași gazdă sau către o alta care știe cum să servească cererea originală. Informația oferită într-un astfel de răspuns Alt-svc are un timp de expirare, făcând ca clienții să poată să redirecționeze conexiuni și cereri ulterioare către gazdă alternativă folosind protocolul alternativ sugerat, pentru o anumită perioadă de timp.

Exemplu

Un server HTTP include un header `Alt-Svc` în răspunsul său:

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

Asta indică că HTTP/3 este disponibil pe port-ul UDP 50781, pe același nume de gazdă care a fost folosit pentru a primi acest răspuns.

Un client poate apoi să încerce să stabilească o conexiune QUIC către acea destiație și, dacă reușește, să continuie să comunice cu acea origin așa în loc de versiunea HTTP originală.

Fluxurile QUIC și HTTP/3

HTTP/3 este făcut pentru QUIC, așa că profită la maxim de fluxurile QUIC, în timp ce HTTP/2 a trebuit să își proiecteze întregul conept de fluxuri și multiplexing peste TCP.

Cererile HTTP făcute peste HTTP/3 folosesc un set specific de fluxuri.

Frame-uri HTTP/3

HTTP/3 înseamnă stabilirea unor fluxuri QUIC și trimiterea unui set de frame-uri către celălalt capăt. Există un număr mic de frame-uri (de fapt doar nouă pe data de 18 decembrie 2018!) cunoscute în HTTP/3. Cele mai importante sunt, probabil:

- HEADERS, care trimite headere HTTP compresate
- DATA, trimite conținut binar
- GOAWAY, te rog închide această conexiune

Cererea HTTP

Clientul își trimite cererea HTTP printr-un flux QUIC bidirecțional inițiat de client.

O cerere este formată dintr-un singur frame HEADERS și poate, opțional, fi urmat de unul sau două alte frame-uri: o serie de frame-uri DATA și, posibil, un frame HEADERS final pentru trailers [date decalate].

După ce a trimis o cerere, un client închide fluxuri pentru trimiteri.

Răspunsul HTTP

Server-ul trimite înapoi răspunsul său HTTP pe un flux bidirecțional. Un frame HEADERS, o serie de frame-uri DATA și, posibil, un frame HEADERS decalat.

Headere-le QPACK

Frame-urile HEADERS conțin headere HTTP compresate folosind algoritmul QPACK. QPACK este similar în stil cu algoritmul de compresie al HTTP/2 numit HPACK (RFC 7541), dar modificat să lucreze cu fluxuri trimise neordonat.

QPACK în sine folosește două fluxuri QUIC unidirecționale între două sisteme. Sunt folosite pentru a transfera infomații tabulare dinamice în oricare direcție.

Prioritizarea HTTP/3

Unul dintre frame-urile fluxurilor HTTP/3 se numește `PRIORITY`. Este folosit ca să se stabilească prioritatea și dependințele pe un flux într-un mod similar cu modul de funcționare din HTTP/2.

Frame-ul poate stabili că un flux specific depinde un alt flux specific și poate da unui anumit flux "greutate".

Unui flux dependent ar trebui să îi fie alocate resurse ori dacă toate celelalte fluxuri de care depinde sunt închise sau dacă nu este posibil să progreseze.

O greutate a fluxului este o valoarea între 1 și 256 și este specificat ca fluxuri cu același părinte **ar trebui** să aibă alocate resurse proporțional pe baza greutății lor.

Trimiterea de către server (Server push) în HTTP/3

Trimiterea de către server în HTTP/3 este similară cu cea descrisă în HTTP/2 (RFC 7540), dar folosește mecanisme diferite.

O trimitere de către server este, efectiv, răspuns la o cerere pe care un client nu a trimis-o niciodată!

Trimiterile de către server sunt permise numai dacă clientul a fost de acord cu ele. În HTTP/3, clientul stabilește o limită pentru cât de multe trimiteri acceptă, informqnd server-ul care este ID-ul maxim de flux. Trecerea peste acea limită va cauza o eroare de conexiune.

Dacă server-ul consideră probabil că clientul dorește o resursă în plus, pe care nu a cerut-o, dar pe care ar trebui să o aibă oricum, poate trimite un frame `PUSH_PROMISE` (prin fluxul de cerere) arătând cum arată cererea pentru care trimiterea este un răspuns, și apoi să trimită răspunsul în sine printr-un nou flux.

Chiar și atunci când s-a spus în prealabil că trimiterile sunt acceptate de către un client, fiecare flux individual poate fi anulat la orice moment dacă clientul consideră necesar. Atunci trimite un frame `CANCEL_PUSH` către server.

Problematic

Încă de când această funcționalitate a fost discutată prima oară în dezvoltarea HTTP/2 și apoi, după ce protocolul a fost publicat pe internet, această funcționalitate a fost discutată, displăcută și lovită în nenumărate moduri pentru a o face utilizabilă.

Trimiterea nu este niciodată "gratuită", deoarece, chiar dacă salvează jumătate de round-trip, tot folosește lățime de bandă. Este deseori greu sau imposibil ca server-ul să știe cu un nivel ridicat de certitudine dacă o resursă ar trebui să fie trimisă sau nu.

HTTP/3 comparat cu HTTP/2

HTTP/3 este proiectat pentru QUIC, care este un protocol de transport care manipulează fluxurile singur.

HTTP/2 este proiectat pentru TCP și, deci, manipulează fluxurile în layer-ul HTTP.

Similarități

Cele două protocoale oferă clienților seturi de funcționalități practic identice.

- Ambele protocoale oferă fluxuri
- Ambele protocoale oferă suport pentru trimiteri de către server
- Ambele protocoale au compresia headere-lor, iar QPACK și HPACK sunt similare în proiectare.
- Ambele protocoale oferă multiplexing peste o singură conexiune folosind fluxuri
- Ambele protocoale oferă prioritizare pe fluxuri

Diferențe

Diferențele sunt în detalii și sunt acolo mulțumită folosirii de către HTTP/3 a QUIC:

- HTTP/3 are un suport pentru date timpurii mai bun și mai probabil să meargă, mulțumită handshake-urilor 0-RTT ale QUIC, în timp ce TCP Fast Open și TLS trimit mai puține date și întâlnesc, deseori, probleme.
- HTTP/3 are handshakes mult mai rapide datorită QUIC față de TCP + TLS.
- HTTP/3 nu există într-o versiune nesigură și necriptată. HTTP/2 poate fi implementat și folosit fără HTTPS - chiar dacă asta este acum rar peste internet.
- HTTP/2 poate fi negociat direct într-un handshake TLS cu extensia ALPN, în timp ce HTTP/3 există peste QUIC, deci are nevoie înainte de un header de răspuns `Alt-Svc` pentru a informa clientul de acest lucru.

Critici

UDP nu va merge niciodată

Multe companii, operatori și organizații blochează sau limitează traficul UDP în afara port-ului 53 (folosit pentru DNS) din moment ce, în ultimul timp, a fost abuzat pentru atacuri. În mod special, câteva dintre protocoalele UDP existente și implementări de server populare pentru ele au fost vulnerabile la atacuri de amplificare, în care un atacator poate genera o cantitate mare de trafic de ieșire pentru a afecta victime nevinovate.

QUIC are o construită o temperare pentru atacurile de amplificare prin cererea ca pachetul inițial să fie de minim 1200 de octeți și printr-o regulă în protocol care spune că un server NU TREBUIE să trimită în răspuns mai mult de trei mărimea cererii, fără să fi primit un pachet de la client în răspuns.

UDP este încet în kernel-uri

Asta pare a fi adevărat, cel puțin acum, în 2018. Nu putem, bineînțeles, să spunem cum se va dezvolta acest lucru și cât din asta este pur și simplu că rezultatul performanței transferurilor UDP nu a fost în prim plan pentru dezvoltatori de mulți ani.

Pentru majoritatea clienților, această "încetinire" nu va fi probabil observată vreodată.

QUIC folosește prea multă capacitate de procesare

Similar cu remarca "UDP este încet" de deasupra, asta este parțial din cauză că folosirea TCP și TLS în lume a avut mult timp să se maturizeze, îmbunătățească și să primească asistență hardware.

Există motive să ne așteptăm că asta se îmbunătăți cu timpul. Întrebarea este cât de mult această capacitate de procesare în plus va afecta dezvoltatorii.

Asta e numai Google

Nu, nu este. Google a adus specificația inițială la IETF după ce au demonstrat, la o scară largă pe internet, că publicarea acestui tip de protocol peste UDP chiar funcționează și are performanțe bune.

De atunci, persoane de la un număr mare de companii și organizații au lucrat în organizația neutră IETF să pună la punct un protocol de transport din ea. La acea muncă au participat, bineînțeles, și angajați Google, dar la fel au participat și angajați de la un număr mare de alte companii interesate în avansarea stării protocoalelor de transfer pe internet, inclusiz Mozilla, Fastly, Cloudfare, Akamai, Microsoft, Facebook și Apple.

Asta e o îmbunătățire mult prea mică.

Asta nu este cu adevărat o critică, cât o opinie. Poate că este, și poate că este o îmbunătățire mult prea mică așa de aproape în timp de când a fost publicat HTTP/2.

HTTP/3 este probabil să funcționeze mult mai bine în rețele pline de pachete pierdute, oferă handshakes mai rapide, așa că va îmbunătăți timpii de încărcare atât percepuți cât și actuali. Dar este asta de ajuns ca benificii pentru a motiva oamenii să publice suport pentru HTTP/3 pe servere-le lor și pentru serviciile lor? Timpul și măsurătorile de perfomanță din viitor vor spune cu siguranță.

Specificațiile

Aici este o colecție a ultimelor ciorne oficiale pentru diverse părți și componente ale QUIC și HTTP/3.

Invariante

Proprietățile independente de versiune ale QUIC

Transfer

QUIC: Un transfer multiplexed și sigur bazat pe UDP

Recuperare

Detectarea pierderilor și control congestiilor în QUIC

TLS

Folosirea TLS pentru a securiza QUIC

HTTP

Hypertext Transfer Protocol (HTTP) peste QUIC

QPACK

QPACK: Compresia headere-lor HTTP peste QUIC

QUIC v2

Pentru a obține cel mai bun nivel de concentrare posibil pe miezul protocolului QUIC și pentru a putea livra la timp, câteva dintre funcționalitățile planificate inițial să fie parte din protocolul de bază au fost amânate și, acum, sunt planificate să fie gata într-o versiune QUIC ulterioară. Versiunea QUIC 2 sau mai mare.

Autorul acestui document deține, totuși, un glob de cristal relativ stricat, așa că nu putem spune sigur exact ce funcționalități vor apărea sau nu în versiunea

1. Putem, totuși să menționăm câteva dintre funcționalitățile și lucrurile care au fost scoase explicit din versiunea 1 a muncii pentru a fi "lucrate mai târziu" și care e posibil să apară într-o versiune 2.

Forward Error Correction

Forward error correction (FEC) este o metodă de a obține control asupra erorilor în transmisiuni în care emițătorul trimite informații redundante, iar receptorul recunoaște numai o bucată din informația care nu conține, aparent, nicio eroare.

Google a experimentat cu asta în munca lor originală asupra QUIC, dar, ulterior, a înlăturat-o din nou, de vreme ce experimentele nu au ieșit atât de bine. Această funcționalitate este supusă discuției pentru QUIC v2, dar, probabil, va dura mult până când cineva va dovedi că este o adiție folositoare, fără prea multe penalizări.

Multipath

Multipath înseamnă că transferul poate, în sine, folosi mai multe rute de rețea pentru a maximiza folosirea resurselor și creșterea redundanței.

Apărătorii SCTP din lume doresc să menționeze că SCTP deja include multipath, la fel ca și TCP modern.

Informații inconsistente

A fost discutată oferirea ca opțiune de fluxuri "inconsistente", ceea ce ar permite QUIC să înlocuiască și programele stil-UDP.

Adaptări non-HTTP

DNS peste QUIC a fost una dintre mențiunile de protocoale non-HTTP timpurii, care ar putea primi niște atenție odată ce versiunea 1 a QUIC și HTTP/3 sunt lansate. Dar cu un nou transfer ca acesta oferit lumii, nu îmi pot imagina că se va opri acolo