HTTP/3 Explained

by Daniel Stenberg

目次

詳解 HTTP/3

この本の試みは2018年3月に始まりました。HTTP/3 と、その根幹のプロトコルである QUIC を文書化することがその目的です（なぜ、どのようにして動作するのか、プロトコルの詳細、その実装など）。

この本は完全に無償で提供され、援助したいと考えるすべての人を巻き込んだ共同作品です。

前提条件

この本の読者は、TCP/IP ネットワーキングの基礎や、HTTP、Web の基本を理解しているものとみなされます。 HTTP/2 に関する詳細や特徴については、http2 explained を最初に読むことを推奨しています。

著者

この本は Daniel Stenberg によって作成されました。 Daniel は、HTTP クライアントソフトウェアとして世界中で最も幅広く使われている curl の作者であり、リードデベロッパーです。 Daniel は20年以上にわたり HTTP やインターネットのプロトコルに関して取り組んでおり、 http2 explained の著者でもあります。

訳者:

- inductor
- ebiiim
- kousukekikuchi1984
- MATTENN
- beagle
- MakTak
- akihirok2k2
- OldBigBuddha
- hidesuke
- ichika
- peacock
- gim_kondo
- hykw
- misato8310
- aoi
- morin_river
- waku-tkd
- smaeda-ks

ホームページ

この本のホームページは daniel.haxx.se/http3-explained にあります。

ヘルプ！

本文書に関する誤字脱字やあからさまな間違いを見つけた場合は、修正した状態の文書を送っていただければ、改訂版を作成します。

助けていただいたすべての方に、適切なクレジットを提供します！ 時間をかけてこの文書を良くしていければと思っています。

よろしければ、誤字の指摘 または プルリクエスト を本の GitHub ページに送ってください。

License

なぜ QUIC なのか

QUIC は略語ではなく名称です。英単語の ”quick” と全く同じ発音です。

QUIC は多くの点で、HTTP のようなプロトコルに適した、新しく信頼性の高い安全なトランスポートプロトコルであり、TCP や TLS 上で HTTP/2 を動かす上で知られるいくつかの欠点に対応できる方法と見なすことができます。

Web トランスポートの進化における論理的な次のステップです。

QUIC は HTTP のトランスポートだけに限定されるものではありません。この新しいトランスポートプロトコルを作り始めた、おそらく最大の理由にして最初のきっかけは、エンドユーザーに Web とデータをより早く配信したいという願望です。

それでは、なぜ新しいトランスポートプロトコルなのでしょうか、なぜ UDP の上に作ったのでしょうか？

HTTP/2、覚えていますか？

HTTP/2 の仕様、RFC 7540 は2015年の5月に公開され、インターネット及び Web に広く展開・実装されてきました。

2018年はじめには、世界トップ1000のウェブサイトのうちほぼ40 %が HTTP/2 で動作しており、Firefox が発行する get レスポンスのうちおよそ70 %の HTTPS リクエストを HTTP/2 が占め、すべての主要なブラウザやサーバー、プロキシが HTTP/2 をサポートしています。

HTTP/2 は HTTP/1 におけるおびただしい数の欠点に対処しており、導入によって HTTP ユーザーが抱える多くの回避策を使わなくてすむようになります。

HTTP/2 の主要な特徴の一つには、多重化があり、論理的な複数のストリームを物理的に単一な TCP 通信で転送することができます。

輻輳制御の動作が飛躍的に向上し、ユーザーが TCP をより効率的に使うことができるため、帯域幅を適切に使い切ることができ、TCP コネクションをより長く持たせるようになります。

結果として、以前よりもより頻繁に最高速の通信に達することができます。

ヘッダー圧縮により帯域幅をより小さくできます。

HTTP/2 では、ブラウザが各ホストに対して確立する TCP コネクションの数は、前バージョンの6個とは異なり、単一が一般的です。

実際、HTTP/2 で使われているコネクションの統合と「デシャーディング」技術はそれよりもさらにコネクション数を減らすことがあります。

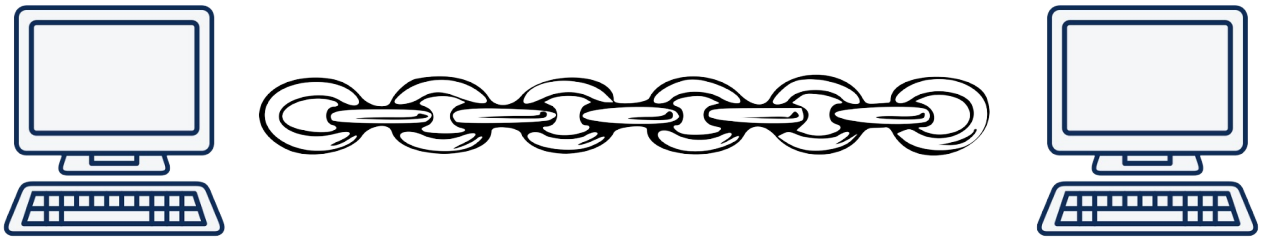
HTTP/2 は HTTP における head-of-line ブロッキング問題（クライアントは、次のリクエストを送信するために既存のリクエストのデータを取得し終わるまで待たなければならない問題）を解消しています。

TCP head-of-line ブロッキング

HTTP/2 は TCP を使って動作し、従来のバージョンの HTTP を使用するよりも少ない TCP コネクション数になります。

TCP は信頼性の高い転送のためのプロトコルで、基本的には2つのマシンの間につながった仮想的な鎖のように考えることができます。

一方の端からネットワーク上に出されたデータは、結果的に、もう一方の端に全く同じ順番で到達します（または通信が途切れます）。



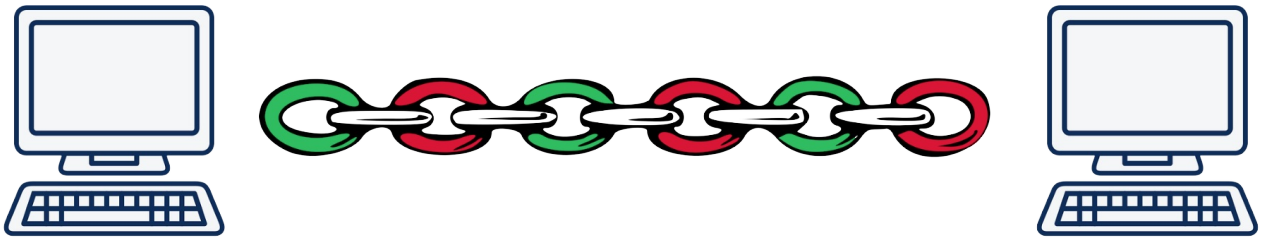
HTTP/2 を使うことで、典型的なブラウザは数十または数百の並列データ転送を単一の TCP コネクション上で行います。

HTTP/2 を話す2つのエンドポイント間にあるネットワーク上で一つのパケットがドロップされると、それはすなわち、その損失したパケットが再送され、宛先に届けられるまでの間、TCP コネクション全体が停止することを意味します。

TCP がこの「鎖」であるため、一つのつなぎ目が突然欠落すると、その欠落したもの以降すべてのつながりが待つ必要があります。

2つの別々のストリームを単一コネクションで送信するときに鎖を使って図解したイラスト。

赤色のストリームと緑色のストリーム:



これが、TCP ベースの head-of-line ブロックになります！

パケットロス率が上がるにつれて、HTTP/2 のパフォーマンスはますます低下します。

2 %のパケットロス（これはかなりひどいです、念のため。）があるネットワーク環境では、大抵の場合において HTTP/1 のほうがパフォーマンスが良くなることがテストで証明されています。

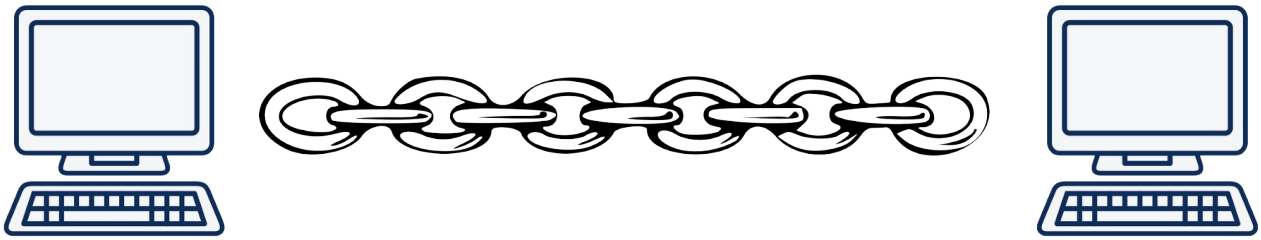
これは、HTTP/1 では通常6つまでの TCP コネクションを使ってパケットを送信するためです。

つまり、パケットが損失した箇所があっても他のコネクションが止まることはないということです。

TCPを使用する限り、この問題を解決することは簡単ではありません。

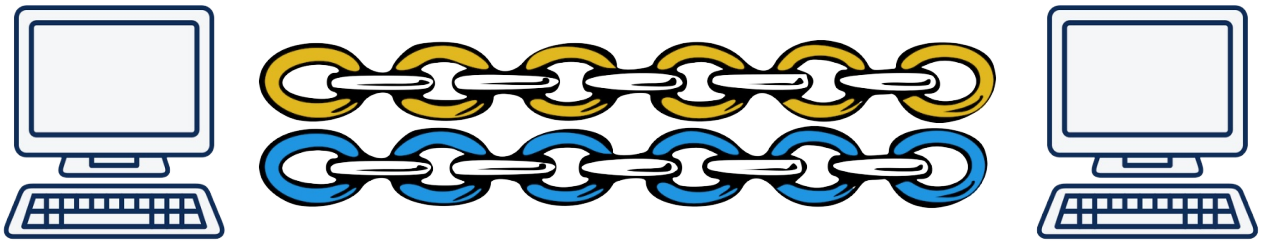
ブロックを解消するための独立したストリーム

QUIC では、2つのエンドポイントの間に、接続を安全にし、データ配信を信頼できるものにするコネクションのセットアップが依然として存在します。



このコネクション上で2つの異なるストリームをセットアップする際、それらは独立したものとして扱われるため、一つのストリームのあるつなぎ目が損失した場合、そのストリームの、特定のチェインのみが、停止し再送制御を行います。

黄色のストリームと青色のストリームがそれぞれ2つのエンドポイント間で通信を行うイラストがこちらです。



TCP か UDP か

TCP の枠組みで head-of-line ブロッキングを直せないのであれば、理論的には、新しいトランスポートプロトコルをネットワークスタックの中、UDP と TCP に隣接して作れるようになるべきです。もしくは 最悪 SCTP を使うべきかもしれません。SCTP は RFC 4960 の中で IETF によって標準化されたトランスポートプロトコルの一つで、必要とされるいくつかの機能を持っています。

しかし、近年新規のトランスポートプロトコルを作る取り組みは、ほとんどすべて休止しています。何故なら、インターネット上に実際に展開することに困難が伴っていたからです。新規プロトコルの展開は、到達すべきユーザーとサーバーの間に展開されている TCP もしくは UDP のみ許可する、多数のファイアーウォール、NAT、ルーターなど、その他のミドルボックスによって阻まれてきました。他のトランスポートプロトコルを導入することは、N %のコネクションを失敗させることになります。何故なら UDP もしくは TCP ではないということは、何らかの形で不正、もしくは間違っているとボックスにみなされ、ブロックされるからです。多くの場合、 N %の失敗率は努力に対して高すぎるとみなされます。

加えて、通常ネットワークスタックのトランスポートプロトコルレイヤーの中を変更することは、OS のカーネルによって実装されているプロトコルを変更することを意味しています。OS のカーネルを更新して展開することは、多大な努力を必要とする遅いプロセスです。IETF によって標準化された多くの TCP の改善は、広くサポートされていないため、広範囲に渡って展開されたり使用されたりしていません。

なぜ SCTP-over-UDP ではないのか

SCTP はストリームを用いた信頼性のあるプロトコルで、WebRTC には UDP を使用する実装さえ存在します。

これは QUIC に取って代わるものとして十分ではありませんでした。以下を含む幾つかの理由に原因があります。

- SCTP がストリームの head-of-line ブロッキング問題を解決しないこと
- SCTP では、ストリームの数をコネクションのセットアップ時に決めなければならないこと
- SCTP が確かな TLS/security レイヤーを持たないこと
- SCTP は 4-way ハンドシェイクを使用し、QUIC は 0-RTT を提供すること
- QUIC は TCP 同様バイトストリームで、SCTP はメッセージベースであること
- QUIC コネクションは IP アドレス間を移動することができ、SCTP はできないこと

更なる詳細と違いについては、A Comparison between SCTP and QUIC インターネットドラフトが参考になります。

硬直化

インターネットは「ネットワーク同士のネットワーク」です。

このネットワーク同士のネットワークを想定通り確実に動作させるため、インターネット上の実に様々な場所に機器が設置されています。

ネットワーク上に散りばめられたこれら機器のことを「ミドルボックス」と呼びます。

2つのエンドポイント間のあちこちに存在しているこれらミドルボックスが、旧来からのネットワークデータ転送に関わっています。

ミドルボックスは様々な目的で様々な動作をしていますが、それらはインターネットを正しく動作させるために必要だと誰かが考えた結果設置されたものだ、と広い意味で捉えることができると思います。

ミドルボックスはネットワーク間で IP パケットのルーティングを行います。

また、悪意のあるトラフィックをブロックします。

NAT (Network Address Translation) を行ったり、パフォーマンスを向上させたり、通過するトラフィックを監視したり、それ以外にも様々なことを行います。

これらの役割を担うため、ミドルボックスは「ネットワーク間がどうつながっているのか」や、監視・管理の対象である、プロトコルの詳細について知らなければなりません。

こういった目的で、ミドルボックスはソフトウェアを動かします。

これらのソフトウェアは必ずしも頻繁にアップデートされるとは限りません。

ミドルボックスはインターネットを維持するための接着剤の役割を果たしていますが、最新技術に追従できていないこともよくあります。

世界中のクライアントとサーバーの間にあるミドルボックスは、一般的には末端の機器ほどすぐに変化しません。

ネットワークプロトコルのうち、ミドルボックスにプロトコルを分析させ、通信を許可・ブロックさせたいものは、以下の問題を抱えます。

ミドルボックスは設置した当時の仕様で動作するという問題です。

当時知られていなかった、新しい仕様の追加や振る舞いの変化があると、その機器にとっては不具合や不正といったリスクと見なされます。

このようなトラフィックは、単に捨てられたり、あるいはその機能を使用したくないと思うほど遅延させられたりします。

このような状況は「プロトコルの硬直化」と呼ばれます。

TCP も硬直化によって変化が難しい状況にあります。

クライアントとサーバーの間にある機器が、新しい TCP オプションを不明なものとしてブロックするかもしれません。

プロトコルの詳細が分析できると、それを取り巻くシステムはプロトコルの一般的な振る舞いを学習していき、やがてプロトコルの変更を不可能にしてしまいます。

この硬直化と「戦う」ための真に効果的な唯一の方法は、可能な限り通信を暗号化し、ミドルボックスにそこを通過するプロトコルの詳細を観察させないようにすることです。

セキュア

QUIC は常にセキュアです。

平文版のプロトコルが存在しないので、QUIC コネクションをネゴシエートすることは TLS 1.3 を用いて暗号化とセキュリティを行うことと同義です。

上記のように、（拡張性を損なうという意味での）硬直化や他の種類のブロックと特別な処理を防ぐと同様に、QUIC は Web ユーザーが望んでいた HTTPS のセキュアなプロパティを全て備えています。

暗号化プロトコルがネゴシエートされる前に、平文で送信されるわずかな初期ハンドシェイクパケットがあるのみです。

Earlier data

QUIC は新規コネクションのネゴシエート時間を短縮するため、0-RTT ハンドシェイクと 1-RTT ハンドシェイクを提供します。

TCP における 3-way ハンドシェイクと比較してみてください。

加えて、QUIC はより多くのデータを処理するため、最初から ”early data” をサポートしており、TCP Fast Open よりも簡単に利用できます。

既存のコネクションが終了するのを待つことなく同じホストへ別のコネクションを接続できることが、ストリームのコンセプトとして挙げられています。

TCP Fast Open の問題について

TCP Fast Open は、2014年12月に RFC 7413 として公開されました。

この仕様では、初回の TCP SYN パケットが既に配信されているサーバーに対して、アプリケーションがどのようにデータを渡すのかについて説明しています。

有志によるこの機能のサポートには時間がかかっており、2018年の現在でも問題が発生しています。

TCP スタックの実装者やアプリケーションがこの機能の利点を利用するためには、OS バージョンに応じた有効化の方法や、問題が発生した際にどのように正常な処理に遷移するのか、の両方を理解する必要があります。

いくつかのネットワークが TFO トラフィックを妨げ TCP ハンドシェイクを台無しにする事が確認されています。

これまでの歩み

最初の QUIC プロトコルは Google の Jim Roskind によって設計され、最初の実装は2012年に行われました。2013年に Google の実験として世界中に公開されました。

当時は QUIC は ”Quick UDP Internet Connections” の略語だとされていましたが、そのときからこのように言われなくなりました。

Google はプロトコルを実装し、それに続いて広く使われている Google 製のブラウザ（Chrome）と、広く使われている Google のサーバーサイドのサービス（Google検索、gmail、youtube などなど）に実装しました。彼らは非常に早くプロトコルを改善し、このプロトコルが大量のユーザーでも信頼性が高く動作することを証明しました。

2015年5月に最初の QUIC のドラフトが標準化のために IETF に提出されましたが、QUIC ワーキンググループが承認されスタートするのは2016年後半までかかりました。しかし、多くの人々から注目を集め QUIC ワーキンググループは早急に立ち上がりました。

2017年には Google の QUIC エンジニアが 全ての インターネットトラフィックの約7 %がすでに Google 版の QUIC プロトコルを利用していると報告されました。

IETF

IETF によってプロトコルの標準化が始まり、QUIC ワーキンググループが発足した当初から、QUIC を HTTP ”だけ”ではなく他のプロトコルにも適用し、標準化することにしていました。Google-QUIC は HTTP だけを対象にしていましたが、実際は HTTP/2 のフレーム構文を利用し、HTTP/2 でも効果的に動作していました。

また、IETF-QUIC では Google が ”カスタム”した手法ではなく、TLS 1.3 に基づいた暗号化とセキュリティを取り入れることにしました。

HTTP 以外のプロトコルにも適用したいという要求を満たすため、IETF QUIC プロトコルのアーキテクチャは2つの異なるレイヤーに分割されました。”transport QUIC”レイヤーと ”HTTP over QUIC”レイヤーです（後者は ”hq”と呼ばれることがあります）。

このレイヤー分割は、一見無害のようにも見えますが、IETF-QUIC と Google-QUIC で大きな差分を生み出しました。

ワーキンググループは QUIC の最初のバージョンを予定通りに提供するために、HTTP に範囲を絞り、HTTP で無いものは後に回すことにしました。

私達がこの本を書き始めた2018年3月には QUIC バージョン1の仕様の最終版は2018年11月にリリースされることになっていましたが、現在は延期されて2019年7月となっています。

IETF-QUIC の作業が進むに連れ Google チームは IETF 版から詳細を取り込み、IETF 版が目指しているQUICバージョンにプロトコルを近づけています。Google は Google 版の QUIC を 彼らのブラウザやサービスで引き続き運用しています。

開発中の新しい実装のほとんど は IETF バージョンにフォーカスすると決めており、Google バージョンとの互換性はありません。

HTTP/2 からの経験

HTTP/2 の仕様が RFC 7540 として公開されたのは2015年5月で、QUIC が IETF に最初に提出されるわずか1ヶ月前でした。

HTTP/2 では、既存のHTTPを変えようという機運があり、HTTP/2 を作ったワーキンググループは新しい HTTP は、バージョンが1から2に16年かけて上がったときよりも早いものとなるだろうと確信していました。

HTTP/2 の時点ではソフトウェアスタックもユーザーも HTTP が今後もテキストベースのプロトコルのまま、何かができるとは考えないようになってきました。

HTTP-over-QUIC が HTTP/3 と名前が変わったのは2018年11月でした。

現在の状況

QUIC ワーキンググループは2016年後半からプロトコルの策定のために活発に活動し、現在2019年7月までにリリースする予定で動いています。

2018年の11月現在では、いまだ HTTP/3 の大規模な相互運用テストは実施されていません。2つの実装が存在し、いずれについても、ブラウザや主要なサーバーソフトウェアにも実装が行われていません。

QUIC ワーキンググループの wiki ページには15個ほどの QUIC 実装リスト が掲載されていますが、いずれの実装も最新版の仕様との互換性はまだありません。

QUIC の実装は簡単ではなく、プロトコルは毎日のように変更され続けています。

サーバー

Apache や nginx が QUIC をサポートしたという公式な発表はありません。

クライアント

IETF バージョンの QUIC や HTTP/3 をサポートしたブラウザをリリースした大規模ベンダーはまだありません。

Google Chrome には Google 版の QUIC を何年も前から組み込まれています。しかし、IETF QUIC プロトコルとの互換性はなく、使われている HTTP の実装も HTTP/3 とは異なります。

実装の障害

QUIC は TLS 1.3 を暗号化とセキュリティのために採用することにしました。これは車輪の再発明を避け、信頼できる既存のプロトコルを利用するためです。しかしながら、これと並行して、QUIC での TLS の利用を本当に効率化することにしました。QUIC では ”TLS messages” のみを利用し ”TLS records” は利用しないことにしました。

これは無害な変更に聞こえますが、この決定は QUIC を実装している人たちにとってとても高いハードルとなりました。既存の TLS 1.3 をサポートしているライブラリにはこれらの機能にアクセスする API が不足しており、QUIC ではアクセスできないのです。一方で多くの QUIC の実装者は大きな組織に所属しており、それぞれがもっている TLS スタックを別々に使用しているため、全員にとって問題とはなっていません。

2018年11月現在、例えば広く使われているオープンソースの OpenSSL では、これらの必要な API は全くなく、また近々で追加するような要望も上がっていません。

これにより QUIC スタックは OpenSSL 以外の TLS ライブラリを使う、パッチをあてた OpenSSL のビルドを使う、将来の OpenSSL に対しての更新を求める、といったいずれかの選択を取る必要があり、最終的にはデプロイの障害となります。

カーネルと CPU 負荷

Google と Facebook は QUIC を彼らの膨大なトラフィックに適用した場合、HTTP/2 over TLS に比べ約2倍の CPU が必要になると言っています。

しかし、下記のような説明が含まれています。

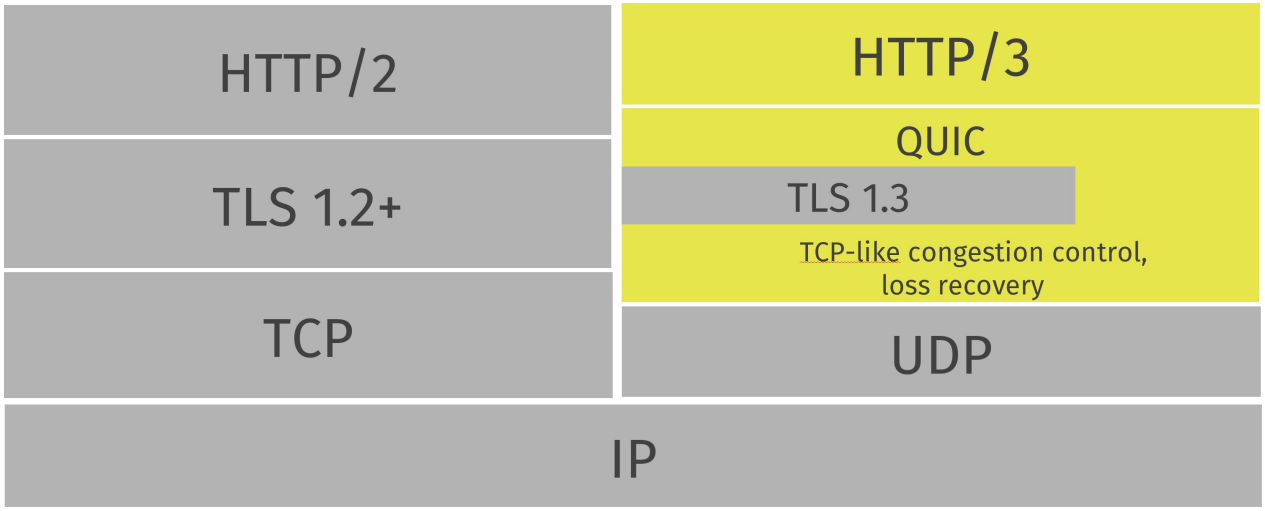
- 歴史的に多くの Linux の UDP スタックはこのような高速通信に利用されることを想定していなかったため、TCP スタックに比べて最適化がされていない
- TCP と TLS の負荷を軽減するハードウェアは存在しているが、UDP のものはほとんどない。基本的に QUIC 向けのものは存在していない

このような問題点がありますが、パフォーマンスと CPU の要求が将来的に改善されると信じています。

プロトコルの機能

高レベルからの QUIC プロトコル

下の図は、HTTP トランスポートとして使われる際の HTTP/2 ネットワークスタックを左側に、QUIC ネットワークスタックを右側に示しています。



UDP 上の転送プロトコル

QUIC は UDP のユーザー空間に実装された転送プロトコルです。

あなたのネットワークトラフィックを軽く確認してみれば、QUIC は UDP パケットとして見えるでしょう。

UDP 上に実装されている以上、QUIC もまた、UDP のポート番号を利用して、与えられた IP アドレス上にある特定のサービスを識別します。

現在、既知の QUIC のすべては、ユーザー空間で動作するように実装されています。

これは通常、カーネル空間で実装するよりも素早い発展が可能です。

うまく動作しますか？

エンタープライズのネットワークなどではポート53（DNS に使われる）以外の UDP トラフィックをブロックするように設定していることがあります。

他にも、制御技術が QUIC の性能を TCP を利用したプロトコルよりも悪くすることがあります。

このような、運用者が行うかもしれないことに関する議論には果てがありません。

近いうちに、すべての QUIC をもとにした転送の用途は、もう一つの（TCP をもとにした）代替手段に正常に切り替えられる必要があるでしょう。

これに関しては、Google のエンジニアによると、計測では1桁台前半の失敗率だったとのことです。

良くなりますか？

QUIC がインターネットの世界にとって価値のある追加物だと証明されれば、人々はそれを使いたいと考え、自分たちのネットワークで機能することを望み、企業はそれに対する障害について考え直すでしょう。

長年にわたり QUIC の開発は進んでおり、インターネットにおいて QUIC を利用した接続の成功率は向上しています。

高信頼性のデータ転送

UDP は信頼性の高い転送プロトコルではありませんが、QUIC は UDP 上に信頼性をもたらすレイヤーを追加します。

これはパケットの再送、輻輳制御、ペーシングおよび現在の TCP が持つこれら以外の機能を提供します。

一方のエンドポイントが QUIC を用いて転送したデータは、コネクションが維持されている限り、遅かれ早かれ他方に届きます。

コネクションは複数のストリームを扱う

SCTP、SSH、HTTP/2 と似たように、QUIC は単一の物理的なコネクションで別々の論理的なストリームを扱えるという特徴があります。

複数の並列ストリームは、それぞれ単一のコネクションで接続しているかのように他のストリームに影響を与えずにデータを転送できます。

QUIC コネクションは、TCP コネクションと似たような仕組みで、2つのエンドポイント間でのネゴシエーションを経て確立します。

QUIC コネクションは UDP ポートと IP アドレスで構成されますが、一旦コネクションが確立すると、それは自身が持つ「コネクション ID」により関連付けられます。

確立済みのコネクションを使って、どちら側もストリームを作成して相手にデータを送信できます。

ストリームのデータは到着順序が保証され、また信頼性があります。

しかし、異なるストリーム間では到着順序は保証されません。

QUIC はコネクションとストリームの両方においてフロー制御を提供します。

詳細は コネクション 節と ストリーム 節で確認できます。

到着順序の保証

QUIC はストリーム内での到着順序を保証しますが、ストリーム間の順序を保証しません。

これは、ストリームは送信したデータの順序を維持しますが、各ストリームが宛先に到達する順序はアプリケーションがそれらを送信したときとは異なるものになり得ることを意味します！

例: ストリーム A と B がサーバーからクライアントに転送されました。

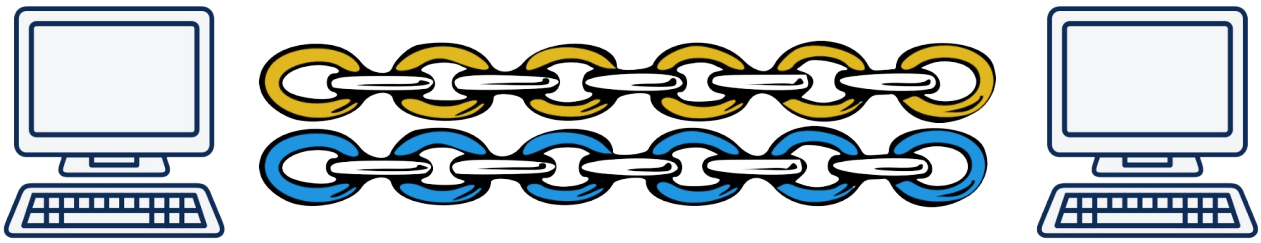
ストリーム A は最初に処理が開始され、ストリーム B はそれに続きました。

QUIC では、パケットの損失はそのパケットが属するストリームのみに影響を及ぼします。

ストリーム A でパケット損失が発生し、ストリーム B では発生しなかった場合、ストリーム A がパケットの再送を行っている間もストリーム B は転送を継続し、先に完了するかもしれません。これは HTTP/2 では不可能でした。

ここに、2つの QUIC エンドポイントが1つのコネクションを介して黄色のストリームと青色のストリームの転送を行う様子を示す図があります。

それぞれのストリームは独立しており、異なる順序で到達する可能性がありますが、いずれのストリームも信頼性と到達順序を損なうことなくアプリケーションに配信されます。



素早いハンドシェイク

QUIC は 0-RTT と 1-RTT 両方のコネクションセットアップを提供し、これは理想的には新しいコネクションを設定する際に追加のやりとりを行う必要がないことを意味します。

0-RTT ハンドシェイクはこの2つのうちで速い方で、事前にコネクションが確立し、かつ、秘密鍵がキャッシュされている状態のときのみ利用することができます。

Early data

QUIC はクライアントが 0-RTT ハンドシェイクの際にデータを含めることを許可しています。

この機能により、クライアントは可能な限り速くピアにデータを配信することができます。

そして、当然のことながら、サーバーは応答して、より早くデータを返すことができます。

TLS 1.3

QUIC におけるトランスポートセキュリティには TLS 1.3（RFC 8446）を使用しており、暗号化されない QUIC コネクションは一切ありません。

TLS 1.3 には従来の TLS と比較していくつか利点があります。 QUIC が TLS 1.3 を使用する主な理由は、1.3からはラウンドトリップ回数が少なくなるようにハンドシェイクに変更が加えられているためです。これにより、プロトコルの待ち時間を短縮することができます。

かつて Google が提案した QUIC では独自の暗号化技術を使用していました。

トランスポートとアプリケーションレベル

IETF QUIC プロトコルはトランスポートプロトコルであり、その上に他のアプリケーションプロトコルも使用可能です。

最も主要なアプリケーションレイヤープロトコルは HTTP/3 (h3) です。

トランスポートレイヤーではコネクションとストリームをサポートします。

従来の Google 版の QUIC はトランスポートと HTTP が一括で動作するよう一つにまとめられており、HTTP/2 フレームを UDP 上で送ることにより特化したプロトコルでした。

HTTP/3 over QUIC

HTTP/3 と呼ばれる HTTP レイヤーは HTTP に準拠したトランスポートを行います。HTTP ヘッダ圧縮では QPACK を利用しており、これは HTTP/2 における HPACK に似ています。

HPACK アルゴリズムはストリームが順番に届くことを前提としているため、修正せずに HTTP/3 で再利用することは不可能です。

これは QUIC の提供するストリームにおいては転送がばらばらに行われるためです。

QPACK は HPACK の QUIC 対応版といえます。

HTTP 以外のプロトコルを QUIC 上で動かす

HTTP 以外のプロトコルを QUIC を使って通信するための対応は、QUIC のバージョン1が実際にリリースされるまで延期されました。

QUIC の仕組み

ネットワーク上でのビットやバイトに関して詳細な説明はしませんが、このセクションでは QUIC トランスポートプロトコルの基本的な要素がどのように機能するのかを説明します。

独自に QUIC スタックを実装したい場合、この説明はあなたの理解に繋がるはずですが、詳細については IETF のインターネットドラフトと RFC を参照してください。

1. コネクション をセットアップ
2. TLS セキュリティ を組み込み
3. ストリーム を開始

コネクション

QUIC コネクションは2つの QUIC エンドポイント間における一対の対話です。QUIC のコネクションの確立は、レイテンシを低減するために、バージョンネゴシエーションと暗号化およびトランスポートハンドシェイクを組み合わせています。

実際にそのようなコネクションを介してデータ送信するには、1つまたはそれ以上のストリームを作成し、使用する必要があります。

コネクション ID

各コネクションは一組のコネクション ID（コネクション識別子）を持ち、コネクションを識別するために、その ID を使用することができます。コネクション ID はエンドポイントにより独立して選ばれます。各エンドポイントは、ピアが使用するコネクション ID を選択します。

これらのコネクション ID の主な機能は、下位のプロトコル層（UDP、IP、 及びそれ以下）でのアドレッシングの変更が、QUIC コネクションのパケットを間違ったエンドポイントに配信しないようにすることです。

コネクション ID を利用することにより、TCP が絶対にできない方法で、IP アドレスとネットワークインターフェースの間でコネクションをマイグレーションすることができます。 例えばダウンロードの進行中に、端末がモバイル回線から Wi-Fi に接続を切り替えたとき、セッションを維持しつつ、ダウンロードがより速い Wi-Fi に移行できます。その逆でも同様に、セッションを維持できます。

ポート番号

QUIC は UDP の最上層に組み込まれているので、16bit のポート番号フィールドは、着信相手を区別するために使用されています。

バージョンネゴシエーション

クライアントからの QUIC コネクションリクエストでは、どの QUIC プロトコルバージョンを使用するかサーバーに通知します。サーバーは、クライアントが次に進む際に選択できるように、自身のサポートしているバージョンのリストを返答します。

接続で TLS を使う

接続の設定に利用される最初のパケットが到着するとすぐに、通信の開始者はセキュアレイヤーにおけるハンドシェイクのセットアップを開始する crypto フレームを送信します。

セキュリティレイヤーは TLS 1.3 セキュリティが用いられています。

TLS を使用せずに QUIC コネクションを行う方法はありません。プロトコルの硬直化を防ぐためにQUICはプロトコルレベルでミドルボックスによる通信の改ざんが難しくなるように設計されています。

ストリーム

QUIC におけるストリームは軽量で順序付けられたバイトストリームの概念を提供します。

QUIC には2種類の基本的なストリームが存在します:

- イニシエータからピアへデータを1方向に転送する単方向ストリーム
- お互いにデータを送ることができる双方向ストリーム

どちらのエンドポイントも、両方のタイプのストリームを作成でき、交互に配置した複数ストリームのデータを並行で送信したり中止したりできます。

QUIC のコネクションを経由してデータを送る際には、1つ以上のストリームが利用されます。

フロー制御

ストリームは各自独立にフロー制御が行われ、エンドポイントがメモリの割当量を制限したり、バックプレッシャーをかけたりできるようになっています。

ストリーム作成も同様にフロー制御が行われ、それぞれのピアが一時的に許可される最大 Stream ID を宣言します。

ストリームの識別名

ストリームは 62bit の符号なし整数により識別され、この整数を Stream ID と呼びます。

Stream ID の最下位 2bit はストリームの種類（単方向もしくは双方向）とストリームのイニシエータの識別に利用されます。

Stream ID の 最下位 bit (0x1) は ストリームのイニシエータを識別します。

クライアントは偶数のストリームを開始し（このときの最下位 bit は 0 に設定されます)、サーバーは奇数のストリームを開始します（このときの最下位 bit は 1 に設定されます)。

Stream ID の最下位 2bit (0x2) は 単方向ストリームと双方向ストリームの両者を区別します。

単方向ストリームでは常にこの bit は 1 に設定され、双方向ストリームではこの bit は 0 に設定されます。

ストリームの並行性

QUIC では任意の数のストリームを並行で操作することができます。エンドポイントは最大の Stream ID を制限することにより、並行して受信できる有効な入力ストリームの個数を制限できます。

Stream ID の上限はエンドポイント特有で、設定を受け取ったピアにのみ適用されます。

データの送受信

エンドポイントはデータの送受信にストリームを利用します。それがつまるところストリームの究極の目的です。

ストリームは順序付けられたバイトストリームの概念です。

別々のストリームは必ずしも元の順序で配信されるとは限りません。

ストリームの優先順位付け

ストリームに割り当てられたリソースに正しい優先順位付けがされているのならば、ストリームの多重化はアプリケーションのパフォーマンスに莫大な効果を与えます。

HTTP/2 のような他の多重化されたプロトコルでの経験から言って、効果的な優先順位付けの計画はパフォーマンスに莫大なプラスの影響を持ちます。

QUIC 自身は優先順位付けの情報を交換するフレームを持ちません。そのかわり、QUIC を利用するアプリケーションからの優先順位情報を信頼します。

QUIC を利用するプロトコルはそのアプリケーションのセマンティクスに合った優先順位付けのスキームを定義することが出来ます。

HTTP/3 に QUIC を利用する場合、HTTP レイヤーにおいては優先順位付けは不要です。

0-RTT

新しいコネクションを確立するのに必要な時間を短縮するため、以前にサーバーと接続していたクライアントは特定のパラメータをキャッシュし、そのパラメータを使ってサーバーとの 0-RTT コネクションを確立してよいことになっています。これにより、ハンドシェイクの完了を待つことなく、クライアントからすぐにデータを送信することが可能になります。

Spin Bit

QUIC ワーキンググループの中でおそらく最も長い間、多数のメールと議論の時間を費やしてきた主題の一つが、単一ビット、つまり Spin Bit です。

このビットの支持者たちは、2つの QUIC エンドポイント間の経路上に存在するオペレータや人々がレイテンシの観測を出来るようになる必要がある、と主張しています。

この機能の反対者たちは、その潜在的な情報漏洩が気に入らないのです。

Spinning a bit

クライアントとサーバー、双方のエンドポイントは各 QUIC コネクションごとに0か1のスピン値が保持されており、その QUIC コネクションの通信パケットは、spin bit を適切な値にセットして送信しています。

両サイドは spin bit が1往復する間は同じ値がセットされたパケットを送り、次の値にトグルします。その影響は、観測者がモニタ可能な形でビットフィールド内の0か1のパルスとして現れます。

この観測は、送信者がアプリケーションやフロー制御による制限を受けていないときにのみ機能し、ネットワーク上のパケットの並び替えはデータを意図しないものにしてしまうことがあります。

ユーザー空間

ユーザー空間にトランスポートプロトコルを実装することで、プロトコルの開発サイクルを速めることができます。これは、クライアントとサーバー OS のカーネルをアップデートすることなくプロトコルの更新を比較的簡単に行えるためです。

技術的には、QUIC 固有の仕組みをユーザー空間ではなくカーネル空間にて実装することも可能です。一部の開発者にとってはそのほうが都合が良いこともあるでしょう。

多数の実装

ユーザー空間に新しいトランスポートプロトコルを実装する明らかな効果は、独立した実装を多数期待できることです。

将来、種々のアプリケーションは異なる HTTP/3 や QUIC の実装を取り込むことが（最上層に実装することも）起こりうるでしょう。

API

通常の TCP やそれを使うプログラムの成功要因の一つは、標準化されたソケット API です。

機能は明確に定義されており、その API を利用することで TCP 同様に、同一のプログラムを多くの異なる OS 間で動かすことができます。

QUIC の場合は違います。QUIC には標準 API はありません。

QUIC においては既存のライブラリ実装を一つ選択し、その API を使い続ける必要があります。

これによりアプリケーションを、単一のライブラリにある程度まで「ロックイン」することになります。

別のライブラリへの変更は別の API への変更を意味し、多くの作業が必要になるかもしれません。

また QUIC は通常はユーザー空間で実装されているため、ソケット API を単に拡張することも、既存の TCP や UDP の機能が行っていることと同じようなことを提供することも簡単ではありません。

QUIC の使用は、ソケット API とは別の API を使おうとすることを意味します。

HTTP/3

前述のとおり、QUIC 上でトランスポートする最初の基本的なプロトコルは HTTP です。

かつて HTTP/2 が完全に新しい方法を用いて、ネットワーク経由で HTTP をトランスポートするために導入されたのとよく似ていて、HTTP/3 もまた、ネットワーク上で HTTP 送信を行う新しい方法を導入します。

HTTP は今でも以前と同じパラダイムとコンセプトを維持しています。

ヘッダとボディがあり、リクエストとレスポンスがあります。

HTTP メソッドがあり、クッキーがあり、キャッシュ機能があります。

HTTP/3 で主に変わったことは、どのようにビットを伝達相手に送るかです。

HTTP over QUIC を実現するために、様々な変更が必要とされ、その成果がいま私達が HTTP/3 と呼んでいるものです。

これらの変更は QUIC が TCP とは異なった性質を持つために必要とされました。

これらの変更は、以下のものを含んでいます。

- QUIC において、ストリームはトランスポート自体により提供されますが、その一方で HTTP/2 において、ストリームは HTTP レイヤーで実行されます。
- ストリームが互いに独自のものであるため、HTTP/2 で使われているヘッダ圧縮プロトコルを使用すると head-of-line ブロック状態を引き起こすようになりました。
- QUIC ストリームは HTTP/2 ストリームとは僅かに異なります。HTTP/3 セクションでは、これについてやや詳細に説明します。

HTTPS:// URLs

HTTP/3 は `HTTPS://` URL を利用して実行されます。

世界はすでにこれらの URL で満ち溢れていて、新しいプロトコルのために別の URL スキームを導入することは現実的ではなく、とても無理があると考えられています。

HTTP/2 が新しいスキームを必要としなかったように HTTP/3 もまた同様です。

HTTP/3 で状況がさらに複雑になったのは、HTTP/2 が完全に新しい方法で HTTP を転送するプロトコルであったものの、それがまだ TLS や TCP など HTTP/1 の仕様に基づいていたからです。

HTTP/3 が QUIC 上で行われるということは、いくつかの重要な側面において物事を変えることになります。

従来の平文 `HTTP://` URL は現在のまま残りますが、私達が安全な転送へとさらに進んでいくにあたり、おそらく徐々に使われなくなっていくでしょう。こうした平文 URL へのリクエストは簡単には HTTP/3 へアップグレードされません。現実にはその他の理由により HTTP/2 にも稀にしかアップグレードされません。

最初のコネクション

以前に訪れたことがない新しい HTTPS:// URL のホストへの最初のコネクションは、おそらく TCP 経由で行われる必要があります（加えて QUIC 経由での接続を並行して試みることもあるでしょう）。

接続先のホストは QUIC をサポートしないレガシーなサーバーかもしれませんし、その中間に QUIC コネクションの障壁となるミドルボックスが存在するかもしれません。

モダンなクライアントやサーバーであれば、おそらく最初のハンドシェイクで HTTP/2 をネゴシエートします。

コネクションが確立され、サーバーがクライアントの HTTP 要求に応答する時に、サーバーはクライアントに対して自身の HTTP/3 のサポートとその優先度を伝えることができます。

Alt-svc

Alternate service (Alt-svc:) ヘッダとそれに対応する `ALT-SVC` HTTP/2 フレームは、QUIC や HTTP/3 用に設計されたわけではありません。

これらはサーバーがクライアントに対して「ちなみに私は同じサービスをこのホスト、プロトコル、ポートでも実行していますよ」と伝えるための、既に設計・作成されているメカニズムの一部です。

詳細は RFC 7838 を参照してください。

このような Alt-svc 応答を受け取ったクライアントは、クライアントが代替プロトコルをサポートしておりかつそれを希望する場合に、指定されたプロトコルで当該ホストへバックグラウンドで並行接続をし、その接続が成功した場合には最初のコネクションの代わりにそのプロトコルへと切り替えを行います。

最初のコネクションが HTTP/2、または HTTP/1 を使用している場合、サーバーはクライアントに対して HTTP/3 で再接続可能であると伝えることができます。それは同じホストに対してかもしれませんし、そのオリジンを提供することが可能な別のホストかもしれません。

こうした Alt-svc 応答で与えられる情報には、ある特定の期間だけクライアントが提案された代替プロトコルを使用して後続のコネクションおよび要求を代替ホストに直接指示できるよう、有効期限が設けられます。

Example

HTTP サーバーはレスポンスヘッダに `Alt-Svc:` を含みます:

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

これは HTTP/3 がこのレスポンスを得るために使われた同じホスト名の UDP ポート 50781 番で利用可能であることを示します。

クライアントはその宛先ホストに対して QUIC コネクションの確立を試みることが可能で、成功した場合は最初の HTTP バージョンではなく、確立されたその代替接続でオリジンと通信を続けることができます。

QUIC ストリームと HTTP/3

HTTP/2 では完全なストリームと多重化のコンセプトを TCP 上で独自に設計する必要がありました。

一方 HTTP/3 は QUIC 用に作られたので、QUIC のストリームを最大限に活用することができます。

HTTP/3 を介して行われる HTTP リクエストはストリームの特定のセットを利用します。

HTTP/3 フレーム

HTTP/3 はつまり QUIC ストリームを作成し、フレームのセットを通信先の相手へ送信することを意味します。

HTTP/3 にはいくつかの（2018年12月18日の時点では9個！）フレームが存在します。中でも最も重要なものは次のとおりです:

- HEADERS 圧縮された HTTP ヘッダを送信
- DATA バイナリデータを送信
- GOAWAY このコネクションをシャットダウンしてください

HTTP リクエスト

クライアントは、自身が開始した 双方向 QUIC ストリーム上で HTTP リクエストを送信します。

1つのリクエストは1つの HEADERS フレームで構成され、場合によっては他に1、2つのフレームが続きます: 一連の DATA フレーム、またはトレーラー用の 最後の HEADERSリクエストを送信した後、クライアントはストリームを閉じます。

HTTP レスポンス

サーバーは、HTTP レスポンスを双方向ストリーム上で返します。

一連の DATA フレーム、またはトレーラー HEADERS フレームからなる1つの HEADERS フレーム。

QPACK ヘッダ

HEADERS フレームには QPACK アルゴリズムにより圧縮された HTTP ヘッダが含まれています。

QPACK は HTTP/2 で使われている HPACK 圧縮（RFC7541）と似ていますが、送信されるストリームの順序に関わらず動作するように変更されています。

QPACK は エンドポイント間で2つの単方向 QUIC ストリームを加えて利用します。

これらのストリームは動的なテーブル情報をそれぞれ伝えるために使われます。

HTTP/3 プライオリティ制御

HTTP/3 のストリームフレームの1つに `PRIORITY` があります。

これはストリーム上の優先順位と依存関係を設定するために使われる HTTP/2 と似た方法です。

フレームは特定のストリームを他の特定ストリームへと依存させ、”weight”を設定できます。

依存関係にあるストリームは、それらが依存するすべてのストリームのいずれかがクローズされている、またはそれ以上処理を進めることができない場合にのみ、リソースが割り当てられるべきです。

ストリームの weight 値は1から256の間で設定可能で、同じ親を持つストリームにはその weight に比例してリソースが割り当てられる べきである と定義されています。

HTTP/3 サーバープッシュ

HTTP/3 サーバープッシュは HTTP/2 で説明されているもの（RFC7540）と似ていますが、異なるメカニズムを利用します。

サーバープッシュは、クライアントからのリクエストがなくても返すことができるレスポンスです。

サーバープッシュはクライアント側が合意した場合にのみ受け取ることが可能です。

加えて HTTP/3 では、クライアントがサーバーに対してプッシュストリームの最大 ID を通知することでプッシュをいくつまで受け入れ可能か制限を設定しています。この制限を超えると、コネクションエラーとなります。

サーバーが、クライアントからの要求はなかったが、いずれ必要になる追加のリソースが他にもあると判断する場合、レスポンスがプッシュであることを示す `PUSH_PROMISE` フレームをリクエストフレームを通して送ることができ、その後実際のレスポンスを新しいストリーム上で送信します。

予めクライアントによってプッシュの受け入れが可能であると通知されている場合であっても、それぞれのプッシュストリームはクライアント側の判断によって拒否することができます。

その際には `CANCEL_PUSH` フレームがサーバーへ送信されます。

問題点

サーバープッシュは HTTP/2 の開発時に当初議論され、プロトコルの策定の後インターネットへと展開されたにもかかわらず、その有用性について数えきれないほどの議論が行われ、叩かれ嫌われ続けてきました。

プッシュは決して ”無料” ではありません。ラウンドトリップの半分を削減することができますが、それでも帯域を使うことに変わりないからです。サーバー側にとって、リソースが実際にプッシュされるべきかどうかをハイレベルで確実に判断することは非常に難しく、おおよそ不可能です。

HTTP/2 と比較した HTTP/3

HTTP/3 は、自身でストリームを処理するトランスポートプロトコルである QUIC に合わせた設計になっています。

HTTP/2 は TCP を前提とした設計のため、HTTP レイヤーでストリームを処理します。

類似点

この2つのプロトコルは、クライアントに対して事実上同じ機能を提供します。

- どちらのプロトコルも、ストリームを提供します。
- どちらのプロトコルも、サーバープッシュのサポートを提供します。
- どちらのプロトコルもヘッダー圧縮が提供され、QPACK と HPACK は設計上似ています。
- どちらのプロトコルも、1つの接続でストリームによる多重化が提供されます。
- どちらのプロトコルも、ストリームの優先順位付けを行います。

相違点

差異は主に詳細にあり、HTTP/3 が QUIC を利用することによるものです。

- TCP Fast Open と TLS では、一般的にあまりデータ量を送信せず、しばしば問題を引き起こす一方で、HTTP/3 は QUIC の 0-RTT ハンドシェイクのおかげで、early data のサポートがあります。
- HTTP/3 は QUIC のおかげで TCP と TLS の組み合わせより高速なハンドシェイクを実現します。
- HTTP/3 では全て暗号化された安全な通信です。HTTP/2 は、インターネット上においては稀なことであるにせよ、 HTTPS なしで実装することも可能です。
- HTTP/2 では TLS ハンドシェイクを ALPN 拡張 を用いて、直接ネゴシエーションが可能です。一方で、HTTP/3 は 事実上、QUIC 上で動くため、クライアントにこの内容を知らせるため `Alt-Svc:` ヘッダーレスポンス が最初に必要です。

よくある疑問点

UDP は多くの企業や組織で動くものではない

多くの企業、運用者、組織は、昨今においてほとんどが攻撃に悪用されるという理由のため、ポート53（DNS に使われる）以外の UDP トラフィックをブロックするか、あるいは利用制限をかけています。

特に、いくつかの既存 UDP プロトコルや UDP プロトコルを用いた一般的なサーバー上の実装はアンプ攻撃に対して脆弱であり続けていて、攻撃者は大量のトラフィックを無関係な第三者に送りつけることができます。

QUIC ではアンプ攻撃を軽減する方法が備わっています。サーバーから受け取る最初のパケットは最低でも1,200バイトを要求するとともに、クライアントからのレスポンスのパケットを受け取っていない場合では、応答としてサーバーは3つよりも多くのパケットを送ってはならない（MUST NOT)、とプロトコル上で明記されています。

UDP はカーネル内で遅い

少なくとも2018年においては、 UDP がカーネル内で遅い点は正しいように思われます。

純粋に長年の間 UDP 転送のパフォーマンスは開発者の関心を集めてこなかったため、果たしてどの程度遅いのか、今後どのように発展していくのかは分かりません。

ほとんどのクライアントにとって、この「遅さ」が気になることはないはずです。

QUIC は CPU 使用量が高すぎる

先述の「UDP は遅い」と同様ですが、これも TCP や TLS の世界的な普及がより成熟したり、ハードウェア支援ができるまでに必要な時間をかけているため、CPU 使用量が高すぎる点に関してもあまり当てはまりません。

もちろん、時間を経るごとに、こういった改善が見込めます。問題は利用者がどれだけ余剰な CPU 使用の増加を気にしなければならないかです。

これは Google の規格でしょ？

いいえ、違います。 Google はインターネット規模で UDP を用いた QUIC 同様の規格が正しく動き、良いパフォーマンスであることを証明し、最初の仕様を IETF へ提案しました。

それ以来、多数の企業や組織から参加している個人が、ベンダーニュートラル な IETF でプロトコルの標準化に取り組んでいます。

もちろん Google の従業員は参加していますが、他にも Mozilla、Fastly、Cloudflare、Akamai、Microsoft、Facebook、Appleなど、インターネット上のトランスポートプロトコルの発展に興味を持つ多くの企業の従業員が参加しています。

改善にしては小さすぎる

それは批判ではなく、1つの意見です。おそらくその通りで、HTTP/2 が展開されてからの改善としては非常に小さいものかもしれません。

HTTP/3 はパケットロスの多いネットワーク上でより優れた性能を発揮し、より高速なハンドシェイクによって数字上でも体感でもレイテンシの改善が見込まれます。

しかしそれがサーバーやサービスへ HTTP/3 サポートを導入するほどのメリットであると言えるでしょうか？

時が経てば、そして将来のパフォーマンス測定結果が間違いなくそれを教えてくれるでしょう！

The specifications（仕様）

ここでは QUIC や HTTP/3 の構成に関する最新のドラフトをまとめています。

Invariants（不変事項）

Version-Independent Properties of QUIC（バージョンに依存しない QUIC の要素）

Transport（トランスポート プロトコル）

QUIC: A UDP-Based Multiplexed and Secure Transport（QUIC: UDP をベースにした多重でセキュアなトランスポートプロトコル）

Recovery（パケットの修復）

QUIC Loss Detection and Congestion Control（QUIC のパケットロスの検知機能と輻輳制御）

TLS

Using Transport Layer Security (TLS) to Secure QUIC（TLS を用いたセキュアな QUIC）

HTTP

Hypertext Transfer Protocol (HTTP) over QUIC

QPACK

QPACK: Header Compression for HTTP over QUIC（QPACK: HTTP over QUIC のためのヘッダー圧縮方法）

QUIC v2

コアの QUIC プロトコルに重点を置き、予定通りリリースできるようにするため、もともとコアプロトコルの一部として計画されていたいくつかの機能の実装が延期されました。延期された機能は QUIC バージョン2もしくはそれ以降のバージョンで実装される予定です。

しかし、この文書の著者はかなり不完全な水晶玉を持っているため、バージョン2ではどのような機能が実装されるのか正確に予測できません。

ただし、v1 から明示的に削除された機能や事柄については「今後実装する」とも言えます。それらはバージョン2にて再び実装される可能性があります。

前方誤り訂正

前方誤り訂正（FEC）は、送信機が冗長データを送信し受信機がエラーを含まないデータのみを認識する、データ送信においてエラーを制御する方法です。

Google はオリジナルの QUIC の動作でこれを実験しましたが、その後実験はうまくいかなかったため再度削除されました。この機能は QUIC v2 の議論の対象となりますが、おそらく誰かが多くのペナルティなしで有用な追加になることを証明する必要があります。

マルチパス

マルチパスとは、トランスポート自体が複数のネットワーク経路を使用してリソースの使用率を最大化することで、冗長性を高めることを意味します。

世界中の SCTP 支持者達は、SCTP はマルチパスを備えていると主張していますが、現代の TCP もマルチパスを備えています。

信頼できないデータ

「信頼できない」ストリームをオプションとして提供することが検討されているため、UDP スタイルのアプリケーションを QUIC へ置き換えることができます。

HTTP 以外の対応

DNS over QUIC は、QUIC v1 と HTTP/3 がリリースされる際に注目されるかもしれない非 HTTP プロトコルの1つでした。

しかし、新しいトランスポートが世界中にもたらされれば、他にもこのようなトランスポートが登場するかもしれません。