HTTP/3 explained

简体中文

本书的编撰自2018年3月开始，计划是提供HTTP/3以及其底层协议QUIC的文档，介绍它们的目的、原理、协议细节以及实现等。

本书完全免费，并且是一个协作项目，欢迎所有愿意帮忙的人前来协助！

先导知识

本书假定读者对TCP/IP网络、HTTP以及Web技术有着基本的理解。关于HTTP/2的更多知识和细节，我们推荐首先阅读HTTP/2详解。

作者

本书作者为Daniel Stenberg，他是世界上最流行的HTTP客户端软件curl的创造者与牵头开发者。Daniel在HTTP与互联网协议中已有20余年的开发经验，他也是HTTP/2详解的作者。

译者：Yi Bai

主页

本书的主页位于daniel.haxx.se/http3-explained。

帮助我们

如果您发现了本书中的任何纰漏、遗漏、错误或公然的谎言，欢迎您将受影响章节的修正版本传给我们，我们将酌情修改。我们将为提供帮助的每个人给予适当的表彰。我们希望随着时间推移，这份文档越来越好。

提交问题或者拉取请求到本书的GitHub页面是最好的反馈方式。

许可协议

为什么需要QUIC

QUIC就是一个名字，不是什么的缩略词。它的发音与英语单词“quick”相同。

QUIC在许多方面可以被视为一种新型的可靠且安全的传输层协议，它适合为形似HTTP的协议提供服务，并且可以解决一些在基于TCP和TLS传输的HTTP/2协议中存在的缺点。它是合乎情理的次世代Web传输层协议。

QUIC的用途不局限于HTTP的传输。将Web与数据更快地交付给最终用户，是创造QUIC这一新型传输层协议的最大原因和源动力。

我们接下来会解释，我们为什么要创造一个新的传输层协议，以及为什么要基于UDP来实现。

QUIC logo

回顾HTTP/2

HTTP/2协议规范（RFC 7540）于2015年5月发表，在那之后，该协议已在互联网和万维网上得到广泛的实现和部署。

2018年初，最热的前一千个网站中约40%运行着HTTP/2，而在Firefox发出的HTTPS请求中，约70%的请求得到了HTTP/2响应。主流的浏览器、服务器以及代理都支持了HTTP/2。

HTTP/2解决了HTTP/1中存在的一大堆缺点，其中相当一部分对于开发者来说非常麻烦。在HTTP/2出现前，开发者要用许多种变通方法来解决，而HTTP/2解决了它们。

HTTP/2的一个主要特性是使用多路复用（multiplexing），因而它可以通过同一个TCP连接发送多个逻辑数据流。复用使得很多事情变得更快更好，它带来更好的拥塞控制、更充分的带宽利用、更长久的TCP连接————这些都比以前更好了，链路能更容易实现全速传输。标头压缩技术也减少了带宽的用量。

采用HTTP/2后，浏览器对每个主机一般只需要 一个 TCP连接，而不是以前常见的 六个 连接。事实上，HTTP/2使用的连接聚合（connection coalescing）和“去分片”（desharding）技术还可以进一步缩减连接数。

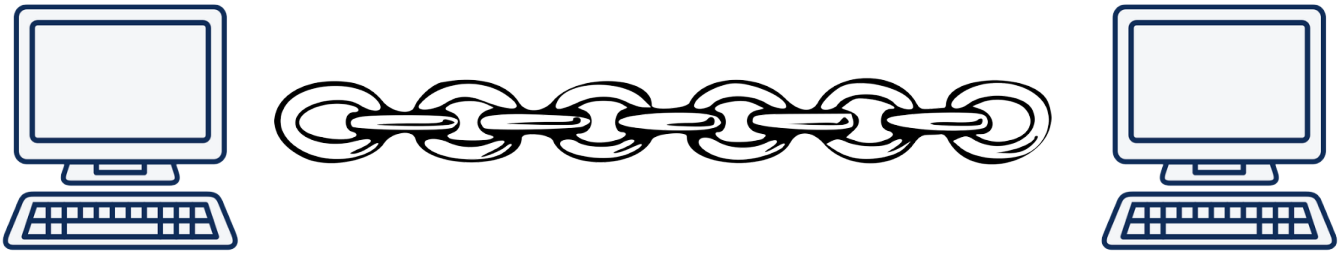
HTTP/2解决了HTTP的队头拥塞（head of line blocking）问题，客户端必须等待一个请求完成才能发送下一个请求的日子过去了。

http2 man

TCP队头阻塞

TCP队头阻塞（head of line blocking）

HTTP/2是基于TCP实现的。相比之前的版本，HTTP/2使用的TCP连接数少了很多。TCP是一个可靠的传输协议，基本上，你可以将它视为在两台计算机间建立的一个虚拟链路，由一端放到网络上的内容，最终总会以相同的顺序出现在另一端。（或者遭遇连接中断）

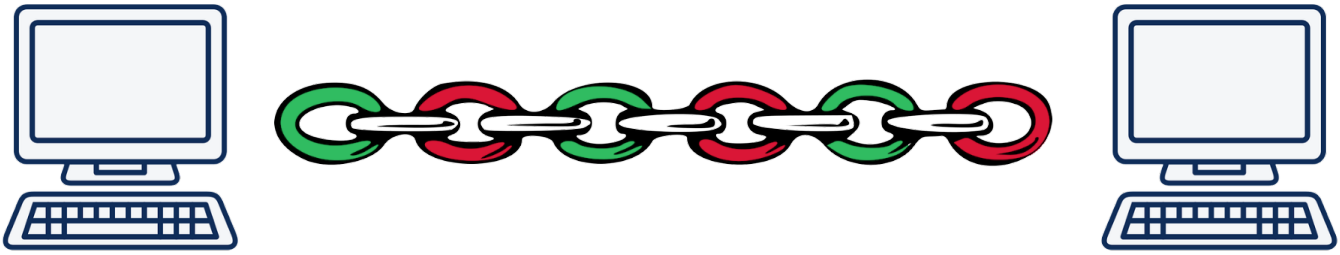


两台计算机间的TCP链路

采用HTTP/2时，浏览器一般会在单个TCP连接中创建并行的几十个乃至上百个传输。

如果HTTP/2连接双方的网络中有一个数据包丢失，或者任何一方的网络出现中断，整个TCP连接就会暂停，丢失的数据包需要被重新传输。因为TCP是一个按序传输的链条，因此如果其中一个点丢失了，链路上之后的内容就都需要等待。

如下图所示，我们一个用链条来表现一个连接上发送的两个流（传输），红色的与绿色的数据流：



不同颜色的链条代表着不同的链路

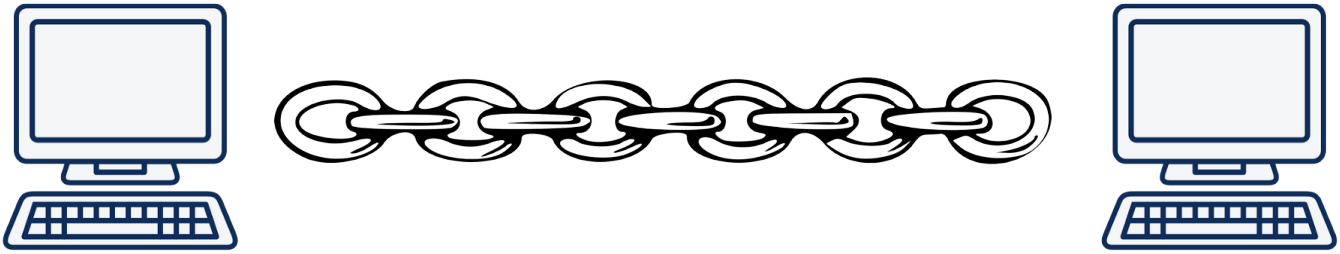
这种单个数据包造成的阻塞，就是TCP上的队头阻塞（head of line blocking）。

随着丢包率的增加，HTTP/2的表现越来越差。在2%的丢包率（一个很差的网络质量）中，测试结果表明HTTP/1用户的性能更好，因为HTTP/1一般有六个TCP连接，哪怕其中一个连接阻塞了，其他没有丢包的连接仍然可以继续传输。

在限定的条件下，在TCP下解决这个问题相当困难。

独立的数据流避免阻塞问题

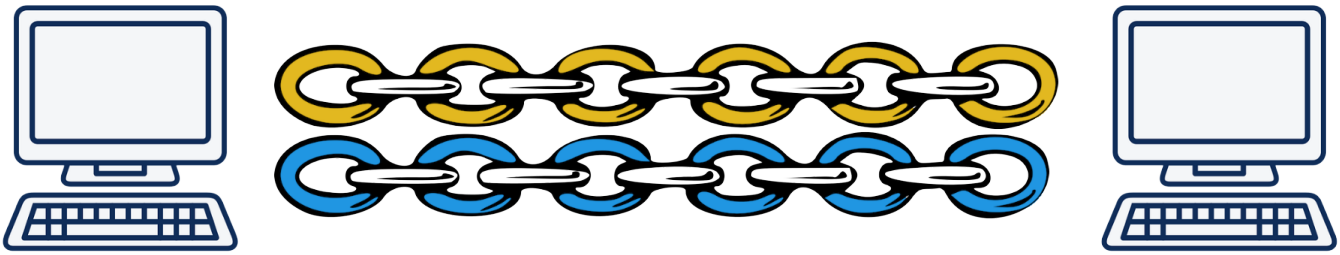
使用QUIC时，两端间仍然建立一个连接，该连接也经过协商使得数据得到安全且可靠的传输。



两台电脑间的一条QUIC链路

但是，当我们在这个连接上建立两个不同的数据流时，它们互相独立。也就是说，如果一个数据流丢包了，只有那个数据流必须停下来，等待重传。

下面是两个端点间的示意图，黄色与蓝色是两个独立的数据流。



两台电脑间的两个QUIC数据流

用TCP还是UDP

用TCP还是UDP

如果我们无法解决TCP内的队头阻塞问题，那么按道理，我们应该在网络栈中发明一个UDP和TCP之外的新型传输层协议。或者我们应该用IETF在RFC 4960中标准化的SCTP传输层协议，它也有多个我们所需的特征。

但在近些年来，因为在互联网上部署遭遇很大的困难，创造新型传输层协议的努力基本上都失败了。用户与服务器之间要经过许多防火墙、NAT（地址转换）、路由器和其他中间设备（middle-box），这些设备有很多只认TCP和UDP。如果使用另一种传输层协议，那么就会有N%的连接无法建立，这些中间设备会认为除TCP和UDP协议以外的协议都是不安全或者有问题的。如此高的的失败率一般被认为不值得再做出努力。

另外，网络栈中的传输层协议改动一般意味着操作系统内核也要做出修改。更新和部署新款操作系统内核的过程十分缓慢，需要付出很大的努力。由IETF标准化的许多TCP新特性都因缺乏广泛支持而没有得到广泛的部署或使用。

为什么不基于UDP使用SCTP

SCTP是一个支持数据流的可靠的传输层协议，而且在WebRTC上已有基于UDP的对它的实现。

这看上去很好，但与QUIC相比还不够好，它：

- 没有解决数据流的队头阻塞问题
- 连接建立时需要决定数据流的数量
- 没有稳固的TLS/安全性支持
- 建立连接时候需要4次握手，而QUIC一次都不用（0-RTT）
- QUIC是类TCP的字节流，而SCTP是信息流（message-based）
- QUIC连接支持IP地址迁移，SCTP不行

若要了解更多SCTP与QUIC的差异，请参阅A Comparison between SCTP and QUIC

协议僵化

互联网（Internet）——多个网络互联之网。为了保证互联网工作正常，我们需要在互联网各处搭建各种设备。这些在互联网上分布式架设的设备被称为中间设备（middle-box）。传统网络传输中两个端点之间的中间设备服务于网络数据传输过程。

这些中间设备有着许多、各式各样的用途和目的，我们简单来说，这些设备是为了实现放置人在该位置所要完成的特定目的而安置。

中间设备的目的包括：在网络之间转发（路由）数据包、阻挡恶意流量、执行地址转换（NAT）、提升性能、监视流量等等。

为了完成这些目的，这些设备必须了解网络以及它们所要监视或修改的数据包协议。它们为此依赖于一些软件——一些很少升级的软件。

虽然这些设备是将互联网“粘”在一起的关键元素，但是它们经常跟不上最新的技术。网络的核心部分与边缘部分（客户端、服务器）相比，更新很慢。

所以当这些设备决定了经过的流量是否合格时，就有些问题了——在这些设备部署之后的一段时间里，协议有了新的特征。而在这些设备引入（了解）这些新特性之前，它们会认为这种特征的数据包是非法的、恶意的，于是会将这种流量直接扔掉，或是拖延到用户不再想使用这些新特征的程度。

这种问题就被称之为“协议僵化”。

协议僵化也影响了TCP协议的改变：当客户端与远程服务器之间的某些中间设备检测到对于它们来说未知的新的TCP选项时，中间设备将拦截这些流量，因为它们不知道这些选项的作用。如果中间设备监测了协议的实现细节，它们会学习到协议的典型行为，在一段时间后，这些行为变得没法更改。

尽可能将通信加密是对抗僵化的唯一有效手段，加密可以防止中间设备看到协议传输的绝大部分内容。

安全性

QUIC始终保证安全性。QUIC协议没有明文的版本，所以想要建立一个QUIC连接，就必须通过TLS 1.3来安全地建立一个加密连接。如上文所说，加密可以避免协议僵化等拦截和特殊处理。这也使QUIC具有了Web用户所期望的所有的HTTPS安全特性。

QUIC只在加密协议协商时会发送几个明文传送的初始握手报文。

减少延迟

与TCP的3次握手相比，QUIC提供了0-RTT和1-RTT的握手，这减少了协商和建立新连接时所需的时间。

除此之外，QUIC提供了提早传输更多数据的“早期数据”（early data）特性，并且它的使用比TCP快速打开（TCP Fast Open）更加简便。

因为数据流概念的引入，客户端不用等待前一个连接结束，便可以与同一个主机建立另一个逻辑连接。

TCP快速打开存在的问题

TCP快速打开选项由2014年12月发布的RFC 7413定义，该规范中介绍了客户端如何通过第一个TCP SYN报文向服务器推送数据。

但在互联网上，对这个选项的实际支持仍需要时间，在2018年的今天，这个选项还是充满很多问题。想要在TCP栈上实现这个选项以获得改进的工程师，不仅要注意操作系统的版本，还要小心地处理发生问题之时，如何优雅地降级到没有该选项的状态。已知有多个网络在积极破坏这种TCP握手，从而干扰了TCP快速打开的流量。

协议进展

最初的QUIC协议由Jim Roskind在Google设计并于2012年实现，经过Google的扩大试验后，于2013年向全世界公开发布。

回顾那时，QUIC还是“快速UDP互联网连接”（Quick UDP Internet Connections）的缩写，但现在已经不是了。

Google实现了QUIC协议，并随后将它部署在其广泛流行的浏览器（Chrome）和最流行的服务（搜索、Gmail、YouTube等等）中。他们相当迅速地迭代该协议的版本，经过一段时间，协议的理念被许多用户的使用证明可以面向大量用户可靠运作。

2015年6月，首个QUIC的互联网草案被提交到IETF以进行标准化，但直到2016年下半年，一个QUIC工作组才被批准成立并投入工作。随后它在各方的高度关注下迅速发展。

在2017年，Google的QUIC工程师称，整个互联网中大约有7%的流量在使用该协议（Google版本）。

IETF

IETF为QUIC标准化成立的QUIC工作组很快就决定，IETF标准化的QUIC协议应该支持HTTP以外的其他应用层协议。Google版的QUIC只传输HTTP——在实践中，它则被用来传输符合HTTP/2帧语义的片段。

另外，工作组最初也决定IETF-QUIC应该基于TLS 1.3进行加密与安全传输，而不使用Google版QUIC定制的方法。

为满足不局限于HTTP的传输需求，IETF QUIC协议的架构被分为两个独立的层：传输层QUIC和“基于QUIC的HTTP”（HTTP over QUIC）层。后者在2018年11月被重命名为HTTP/3。

尽管这种分层结构看似人畜无害，但这实质上造成IETF-QUIC与Google版QUIC有着诸多不同。

工作组很快发现了这一点，为保持适当关注和能按时交付第一版QUIC，工作组的重心转移到了先交付“基于QUIC的HTTP”传输部分，非HTTP传输部分将留待今后研究。

2018年3月，当我们开始写这本书的时候，第一版QUIC最终的规范计划于2018年11月发布。不过发布时间在这之后被推迟到了2019年7月。

在IETF-QUIC取得进展的同时，Google团队已经整合了IETF版本的细节并逐渐推进他们的协议版本，以期最终可能符合IETF定义的规范。尽管如此，Google在他们的浏览器和服务中继续使用自己的QUIC版本。

正在开发的大多数新实现 已经决定着眼于IETF版本，与Google版本并不兼容。

HTTP/2的经验

HTTP/2规范RFC 7540发布于2015年5月，只比QUIC首次进入IETF早一个月。

HTTP/2的发布为HTTP进化奠定了基础，同时这让HTTP/2工作组认识到迭代更新的HTTP版本会比从第一版到第二版（花费约16年）快得多。

随着HTTP/2的发布，用户和软件堆栈不再假设HTTP是一个序列化的、基于文本的协议。

HTTP-over-QUIC(HTTP/3)建立在HTTP/2的基础上，并从中借鉴了许多概念。由于某些具体的概念已被QUIC所涵盖，所以从HTTP层中移除。

标准化进展情况

QUIC工作组自2016年底以来在积极标准化该协议，现计划于2019年7月之前完成。

到2018年11月为止，还没有过大型的HTTP/3互通性测试。目前来说，只有2个实现进行过测试，且这两个实现不基于浏览器和主流的开源服务器软件。

QUIC工作组的wiki页面目前列出了大约15种QUIC实现（QUIC实现列表），但距离它们都能与最新版的规范草案互操作仍有很长的路。

实现QUIC并不容易，且到目前为止，协议本身还在不断演变。

服务器

Apache和Nginx还没有对QUIC支持的公开声明。

客户端

还没有任何主流浏览器的任何状态的任何版本支持IETF版本的QUIC或者HTTP/3协议。

Google Chrome在数年前已经支持Google版的QUIC，但是该版本不能与官方的QUIC版本互操作，且它的HTTP实现与HTTP/3不同。

实现的障碍

为了避免重复发明轮子，以及依靠可信赖的现有协议，QUIC决定使用TLS 1.3作为它的加密和安全协议层。不过工作组决定大幅精简QUIC中TLS的使用，只使用“TLS信息”（TLS Messages）而不是协议中的“TLS记录”（TLS Records）。

这听上去可能人畜无害，但也事实上成为了很多QUIC堆栈实现者的重大障碍。现存的支持TLS 1.3的TLS库都没有提供此功能的API并允许QUIC访问它。有一些QUIC的实现由大型机构完成，这些机构可能有自己的TLS协议栈，但并不是所有实现都能如此。

例如，主流的重量级开源软件OpenSSL就没有这些API，且到目前（2018年11月）为止，没有表达过在任何时间点提供这些API的意愿。

这最终将成为QUIC协议栈部署的障碍。因为QUIC要么基于其他TLS库，要么使用补丁版OpenSSL或者等待OpenSSL版本更新。

操作系统内核、CPU负载

据Google和Facebook称，与基于TLS的HTTP/2相比，它们大规模部署的QUIC需要近2倍的CPU使用量。

对此的进一步解释包括：

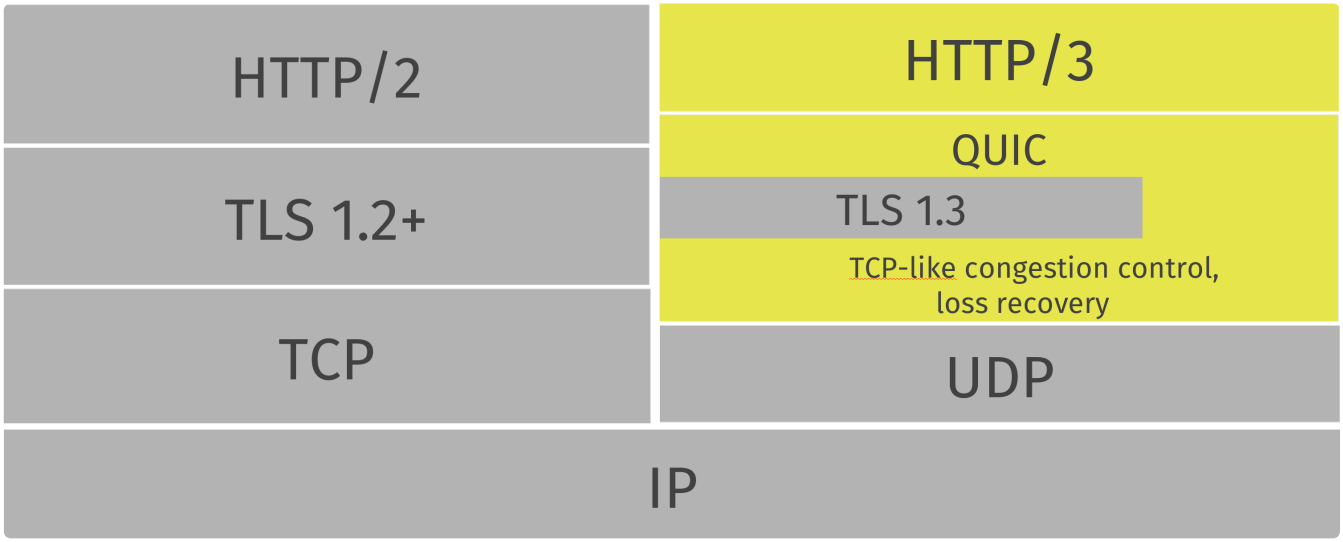
- Linux内核的UDP部分没有得到像TCP堆栈那样的优化，因为传统上没有使用UDP进行如此高速的信息传输。
- TCP和TLS有硬件加速（负载卸载到硬件，offload），而这对于UDP很罕见，对于QUIC则基本不存在。

就上述理由，我们可以相信QUIC的CPU使用量能随着时间的推移得到改善。

协议特点

本章从高层次出发概述QUIC协议。

下图是使用HTTP传输时，HTTP/2（左）和HTTP/3（右）的协议栈对比。



QUIC logo

基于UDP

基于UDP的传输层协议

QUIC是基于UDP在用户空间实现的传输协议。如果不观察细节，你会觉得QUIC跟UDP报文差不多。

基于UDP意味着它使用UDP端口号来识别指定机器上的特定服务器。

目前已知的所有QUIC实现都位于用户空间，这使它能得到更快速的迭代（相较于内核空间中的实现）。

跑得起来吗？

有一些网络上的中间设备会拦截端口53（用于DNS）以外的UDP流量。还有一些网络会节流（throttle）UDP流量，使得QUIC的表现慢于基于TCP的协议。更多网络的表现未知。

在可预见的未来，所有基于QUIC的传输可能会配有一个优雅地回退到另一个基于TCP的后备方案的替代机制。据Google多位工程师早前的报告，协议的失败率不足10%。

会好起来吗？

如果QUIC被证明确实是互联网世界的一个有益补充，用户会希望能正常使用QUIC，网络公司就可能重新解决上述的拦截。多年以来，随着QUIC取得进展，在互联网上建立和使用QUIC的成功率有所提高。

可靠性

虽然UDP不提供可靠的传输，但QUIC在基于UDP之时增加了一层带来可靠性的层。它提供了数据包重传、拥塞控制、调整传输节奏（pacing）以及其他一些TCP中存在的特性。

只要连接没有中断，从QUIC一端传输的数据迟早会出现在另一端。

数据流

类似SCTP、SSH和HTTP/2，QUIC在同一物理连接上可以有多个独立的逻辑数据流。这些数据流并行在同一个连接上传输，不影响其他流。

连接在两个端点之间经过类似TCP连接的方式协商建立。QUIC连接基于UDP端口和IP地址建立，而一旦建立，连接通过其“连接ID”（connection ID）关联。

在已建立的连接上，双方均可以建立传输给对方的数据流。单一数据流的传输是可靠、有序的，但不同的数据流间可能无序传送。

QUIC可对连接和数据流分别进行流量控制（flow control）。

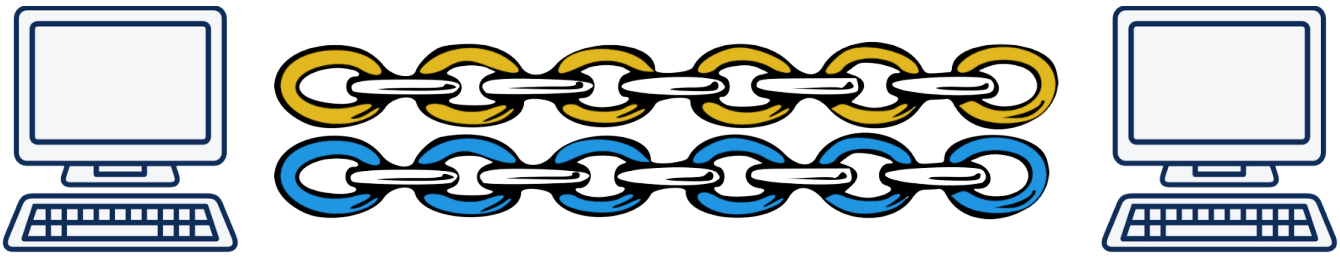
进一步细节参见连接和数据流。

有序交付

QUIC的单个数据流可以保证有序交付，但多个数据流之间可能乱序。这意味着单个数据流的传输是按序的，但是多个数据流中接收方收到的顺序可能与发送方的发送顺序不同！

举个例子：服务器传送流A和B到客户端。流A先启动，然后是流B。在QUIC中，丢包只会影响该包所处的流。如果流A发生了一次丢包，而流B没有，流B将继续传输直到结束，而流A将会进行丢包重传过程。而在HTTP/2中这不可能发生。

下图展示了连通两个QUIC端点的单一连接中的黄色与蓝色的数据流。它们互相独立，所以可能乱序到达，但是每个流内的信息将按序可靠到达。



两台电脑间的两个QUIC数据流

快速握手

QUIC提供0-RTT和1-RTT的连接建立，这意味着QUIC在最佳情况下不需要任何的额外往返时间便可建立新连接。其中更快的0-RTT仅在两个主机之间建立过连接且缓存了该连接的“秘密”（secret）时可以使用。

早期数据（Early data）

QUIC允许客户端在0-RTT的情况下直接捎带数据。这使得客户端能尽早向对方传送数据，当然也使得服务器能更快地发回数据响应。

TLS 1.3

QUIC使用TLS 1.3传输层安全协议（RFC 8446）。QUIC没有非加密的版本。

与更早的TLS版本相比，TLS 1.3有着很多优点，但使用它的最主要原因是其握手所花费的往返次数更低，从而能降低协议的延迟。

Google的传统QUIC使用一个自行定制的加密法。

传输层与应用层协议

IETF版QUIC是一个传输层协议，在该协议之上可以运行其他应用层协议。初始的应用层协议是HTTP/3（h3）。

传输层协议负责连接和数据流处理。

在Google的传统QUIC中，传输层与HTTP融在一起，为包揽一切的全功能设计，它是一个更有指向性的“基于UDP传输HTTP/2帧”（send-http/2-frames-over-udp）的协议。

QUIC之上的HTTP协议

HTTP层以HTTP风格传输内容，包括使用QPACK进行HTTP头部压缩——这和HTTP/2中使用HPACK压缩头部类似。

HPACK算法依赖于数据流的有序交付，由于HTTP/3的数据流之间可能乱序，所以该算法需要修改才能使用。QPACK可被视作适用于QUIC版本的HPACK。

QUIC之上的非HTTP协议

基于QUIC传输非HTTP协议的相关工作已被推迟到第一版QUIC发布之后实现。

QUIC工作原理

本章节将解释QUIC传输层协议各基本模块的功能，而不会逐比特逐字节解释协议的报文。如果你想自己实现QUIC，这些介绍会加强你对协议的理解，但有关具体细节，请参考IETF的互联网草案（Internet Draft）和RFC。

1. 建立一个连接
2. 协商安全的TLS连接
3. 使用数据流

连接

QUIC连接是两个QUIC端点之间的单次会话（conversation）过程。QUIC建立连接时，加密算法的版本协商与传输层握手合并完成，以减小延迟。

在连接上实际传输数据时需要建立并使用一个或多个数据流。

连接ID（Connection ID）

每个连接过程都有一组连接标识符，或称连接ID，该ID用以识别该连接。每个端点各自选择连接ID。每个端点选择对方使用的连接ID。

连接ID的基本功能是确保底层协议（UDP、IP及其底层协议）的寻址变更不会使QUIC连接传输数据到错误的端点。

利用连接ID的优势，连接可以在IP地址和网络接口迁移的情况下得到保持——而这TCP永远做不到。举例来说，当用户的设备连接到一个Wi-Fi网络时，将进行中的下载从蜂窝网络连接转移到更快速的Wi-Fi连接。与此类似，当Wi-Fi连接不再可用时，将连接转移到蜂窝网络连接。

端口号

QUIC基于UDP建立，因此使用16比特的UDP端口号字段来区分传入的不同连接。

版本协商

客户端的QUIC连接请求会告知服务器所希望使用的QUIC协议版本，服务器端会回复一系列支持的版本供客户端选择。

使用TLS的连接

在初始的数据包建立连接之后，连接发起者会马上发一个加密的帧以开始安全层握手。安全层使用TLS 1.3协议。

在QUIC中，没有方法或机制避免使用TLS连接。该设计旨在使中间设备难以篡改数据包，防止协议僵化。

数据流

数据流（Streams）在QUIC中提供了一个轻量级、有序的字节流的抽象化。

QUIC中有两种基本的数据流类型：

- 从发起者到对等端（Peer）的单向数据流。
- 双向均可发出数据的双向数据流。

连接端点的任意一方都可以建立这两种数据流，数据流之间可并行、交错地传输，并且可以被取消。

通过QUIC发送数据需要建立一个或多个数据流。

流量控制（Flow control）

每个数据流都有独立的流量控制，端点可以通过此实现内存控制和反压（back pressure）。数据流的创建本身也有流量控制，连接双方可以声明最多愿意创建几个流ID。

流标识符

数据流通过一个无符号的62比特整数标识，也称流ID。流ID的最低2位比特用于识别流的类型（单向或双向）和流的发起者。

流ID的最低1位比特（0x1）用于识别流的发起者。客户端发起双数（最低位置0）流，服务器发起单数（最低位置1）流。

第2个比特（0x2）识别单/双向流。单向流始终置1，双向流则置0。

流并发

QUIC允许任意数量的并发流。端点通过闲置最大流ID来控制并发活动的传入流数量。

每个端点指定自己的最大流ID数，并只对对等端端点有效。

收发数据

端点使用流来收发数据，这是流的最终用途。QUIC数据流是有序的字节流抽象。但是，不同流之间是无序的。

流优先度

如果正确设置了各流的优先度，流复用机制可以显著提升应用的效率。使用其他多路复用协议（如HTTP/2）的经验表明，有效的优先度划分策略对效率具有显著的正面影响。

QUIC本身没有提供交换优先度信息的报文。接收优先度信息依赖于使用QUIC的应用层。应用层可以定义所有符合其语义的优先度方案。

基于QUIC使用HTTP/3时，优先度信息在HTTP层完成。

0-RTT

先前已连接过一个服务器的客户端可能缓存来自该连接的某些参数，并在之后与该服务器建立一个无需等待握手完成就可以立即传输信息的**0-RTT**连接，从而减少建立新连接所必需的时间。

旋转比特位

在QUIC工作组的设计讨论中，最长的主题之一就是旋转比特位，人们花费了数百封邮件和数百个小时来讨论它。

旋转比特位的支持者认为，两个QUIC端点之间路径上的运营商和人员需要有办法来测量延迟。

反对者则反感此功能潜在的信息泄露。

旋转一个比特

QUIC连接的客户端、服务器这两个端点各为每一个QUIC连接维护一个旋转的值——0或1，在传送时候它们在报文中设置该值。

然后，在每一次往返时，连接双方都翻转这一比特的值。效果是观察者可以检测该比特字段的0与1脉冲。

这一观测只在发送方未被应用层或流量控制限制的情况下有效，并且网络上经过重新排序的数据包也会给数据带来噪声。

用户空间实现

在用户空间中实现一个传输层协议有助于协议的快速迭代，协议的演进更为容易，不需要客户端和服务器更新其操作系统内核才能部署新的版本。

QUIC本身没有固有的东西阻碍未来在操作系统内核中实现和提供QUIC协议。

众多的实现

在用户空间中实现一个新的传输层协议时，一个显而易见的效果是我们会看到很多独立的实现。

在可预见的未来，不同的应用程序可能包含（或基于）不同的HTTP/3和QUIC实现。

API

常规TCP与程序最成功的因素之一便是标准化的套接字（socket）API。其API有着定义良好的功能，使用它能让你轻松地在各操作系统之间移植程序，因为TCP采用同样的方式运作。

但QUIC不是如此。QUIC目前没有标准化的API。

使用QUIC时，你需要选择一个现有的库实现，并坚持使用它的API。这在某种程度上把应用“绑定”到了单一的库上。换库意味着使用另外一套API，这可能带来相当的工作量。

另外，由于QUIC一般在用户空间中实现，所以它不像现有的TCP和UDP套接字API那样能轻松扩展。使用QUIC意味着选择了套接字API之外的另一套API。

HTTP/3

上文提到过，基于QUIC传输的第一个也是最基础的协议是HTTP。

就像HTTP/2是通过网络传输HTTP流量的一种新方式，HTTP/3是另一种通过网络传输HTTP的新方法。

HTTP的范例和概念没有改变。它含有头部（header）和正文（body），请求和回复，还有动词（verb）、Cookie和缓存。HTTP/3的主要改变是将这些报文比特传送到另一端的方式。

为了使HTTP可以通过QUIC传输，协议的某些方面要进行修改，修改的结果便是HTTP/3。这些必要修改是因QUIC与TCP在某些性质上的不同所致，修改包括：

- 在QUIC中，数据流由传输层本身提供，而在HTTP/2中，流由HTTP层完成。
- 由于数据流互相独立，HTTP/2中使用的头部压缩算法如果不做改动，会造成队头阻塞。
- QUIC流与HTTP/2略有不同。本书的HTTP/3章节会做详细介绍。

HTTPS:// URL

HTTP/3将使用`HTTPS://` URL履行。我们的世界里充斥着HTTPS URL，并且为新协议引入另一种URL方案被认为不切实际且完全不合理。如同HTTP/2一样，HTTP/3不会引入新的URL方案。

HTTP/2是传输HTTP的一种新方式，但是它还是基于TLS和TCP，这和HTTP/1一样。而在基于QUIC的HTTP/3中，情况更加复杂，它在一些重要的地方做了一些改变。

历史遗留下来的明文`HTTP://` URL的处理方式将保持原样，随着我们迈入安全传输更加普及的未来，它的使用可能会越来越少。对HTTP URL的请求不会升级为使用HTTP/3。在实践中，因为其他一些理由，它们也很少升级到HTTP/2。

初始连接

当尝试连接到一个全新的、未访问过的网站时，到HTTPS:// URL的连接可能必须通过TCP（也许会有并行的一个QUIC连接）。因为主机可能是一个不支持QUIC的传统服务器，或者链路之间可能有阻碍QUIC成功连接的中间设备。

现代的浏览器和服务器可能在首次握手时协商HTTP/2协议。在连接建立并且服务器响应客户端的HTTP请求时，服务器可以通告客户端它对HTTP/3的支持与偏好。

使用Alt-svc自举

替代服务（alternative service, Alt-svc:）头部和它相对应的 `ALT-SVC` HTTP/2帧并不是特别为QUIC和HTTP/3设计的。它是为了让服务器可以告诉客户端 *“看，我在这个主机的这个端口用这个协议提供相同的服务”* 而设计的。详见RFC 7838。

如果初始连接使用的是HTTP/2（甚至HTTP/1），服务器可以响应并告诉客户端它可以再试试HTTP/3。连接可以指向相同主机或者不同但提供相同服务的主机。Alt-svc回复中有一个到期计时器，让客户端可以在指定的时间内使用建议的替代协议将后续的连接和请求直接发送给替代主机。

例子

一个HTTP服务器的响应中包含了如下的一个 `Alt-Svc:` 头部：

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

这指示了同一名称的主机在UDP端口50781提供HTTP/3服务。

然后，客户端可以尝试与该端口建立QUIC连接。如果成功，后续将通过该连接继续通信，代替初始的HTTP版本。

QUIC流与HTTP/3

HTTP/3针对QUIC设计，所以它可以利用QUIC流的所有好处。而HTTP/2不得不在TCP之上构建它的数据流和复用概念。

通过HTTP/3传输的HTTP请求使用一系列的数据流完成。

HTTP/3帧（frame）

HTTP/3意味着建立QUIC数据流，并将一系列帧发送给对方。HTTP/3中的数据帧种类不多且固定（截至2018年12月18日有九种）。最关键的帧可能是：

- HEADERS：发送压缩的HTTP头部
- DATA：发送二进制数据内容
- GOAWAY：请关闭此连接

HTTP请求

客户端通过其发起的 双向 QUIC流来发送HTTP请求。

一个请求包括一个HEADERS帧，之后可能有一两种其他的帧：一系列的DATA帧，以及可能有一个作为末尾的HEADERS帧。

发送一个请求后，客户端会关闭该数据流以进行发出。

HTTP响应

服务器在双向流上发回其HTTP响应。其中含有一个HEADERS帧，一系列DATA帧，末尾可能有一个HEADERS帧。

QPACK头部

HEADERS含有用QPACK算法压缩的HTTP头部。QPACK与HTTP/2中的HPACK（RFC 7541）类似，并针对乱序流做了相应修改。

QPACK本身在两个端点间使用两个额外的单向QUIC流，用于在两个方向上传递动态表信息。

优先度

HTTP/3中有一种帧是优先度（`PRIORITY`）。与HTTP/2中的类似，它用于设定一个流的优先度和依赖关系。

该帧可以设定一个流依赖于另一个流，也可以设定特定流的“权重”。

服务器应该只在一个流所依赖的所有流都被关闭，或者都无法取得进展时为该流分配资源。

一个流的权重是介于1到256之间的值，有着相同父系流的流**应该**按照权重的比例分配资源。

服务器推送

HTTP/3的服务器推送与HTTP/2（RFC 7540）类似，但机制上有所不同。

服务器推送实际上就是对一个未曾发出的客户端请求做出响应！

服务器推送仅在客户端同意的前提下才允许发出。在HTTP/3中，客户端甚至能通过通告给服务器的最大推送流ID来设置所接受推送的次数限制。超出限制将导致连接错误。

如果服务器端认为客户端可能需要某个并未要求但应该有的额外资源，服务器可以通过请求流发送一个`PUSH_PROMISE`帧，使该推送请求看上去像是一个响应，然后通过新的流发送实际响应。

虽然客户端之前已经表示过推送可接受，但如果客户端认为适合，每个推送流仍可以随时取消，然后发送一个`CANCEL_PUSH`帧到服务器。

问题重重

自从推送这一特性在HTTP/2中讨论、开发、部署以来，它就备受争议、讨论和抨击。为了让它有用，人们付出了许多努力。

推送从来不是没有代价的，尽管它省了半个往返的延迟，它还是会消耗带宽。服务器也很难或不可能从高层面确定一个资源是否应该被推送过去。

与HTTP/2的比较

HTTP/3面向QUIC设计，QUIC是一个自己处理数据流的传输层协议。

HTTP/2面向TCP设计，因此数据流在HTTP层处理。

相似之处

这两个协议为客户端提供了几乎相同的功能集。

- 两者都提供数据流
- 两者都提供服务器推送
- 两者都有头部压缩，QPACK与HPACK的设计非常类似
- 两者都通过单一连接上的数据流提供复用
- 两者都提供数据流的优先度设置

不同之处

两个协议的主要不同点在于细节，不同之处主要由HTTP/3使用的QUIC带来。

- 得益于QUIC的0-RTT握手，HTTP/3可以提供更好的早期数据支持，而TCP快速打开和TLS通常只能传输更少的数据，且经常存在问题。
- 得益于QUIC，HTTP/3的握手速度比TCP+TLS快得多。
- HTTP/3不存在明文的不安全版本。尽管在互联网上很少见，HTTP/2还是可以不配合HTTPS来实现和使用。
- 通过ALPN拓展，HTTP/2可以直接在TLS握手时进行协商。HTTP/3基于QUIC，所以需要凭借响应中的 `Alt-Svc:` 头部来向客户端宣告。

常见批评

UDP永远不会通

很多企业、运营商和组织对53端口（DNS）以外的UDP流量进行拦截或者限流，因为这些流量近来常被滥用于攻击。特别是一些现有的UDP协议和实现易受放大攻击（amplification attack）威胁，攻击者可以控制无辜的主机向受害者投放发送大量的流量。

QUIC内置了对放大攻击的缓解处理。它要求初始数据包不小于1200字节，并且协议中限制，服务器在未收到客户端回复的情况下，不能发送超过请求大小三倍的响应内容。

内核处理UDP很慢

我们必须承认这在2018年的今天是一个事实。当然，UDP技术会发展，这些年开发者对UDP的重视程度也不够，这些东西都自不必说了。

对于大多数客户端来说，这个程度的“缓慢”从未被觉察到。

QUIC太吃CPU

类似上文的“UDP很慢”，一部分原因是TCP和TLS长期以来的成熟发展、改进，以及得到硬件协助，造成UDP看上去比较慢。

我们有理由期望这会随着时间得到改善。问题在于，这额外的CPU占用会对部署者带来多大的影响。

只有Google在弄

并非如此。Google通过大规模的部署证明，通过UDP部署这种协议可以正常运行且表现良好，这为IETF带来了初始的规范。

在那之后，很多公司和组织的人员都在这个利益方中立的IETF组织下推进标准化。在这个阶段，虽然Google的雇员也有参与，但Mozilla、Fastly、Cloudflare、Akamai、微软、Facebook、苹果等等很多公司的员工也参与进来，共同推进互联网的传输层协议。

进步太小

这个是一个观点，而不是批评。也许进步是很小，这可能与相距HTTP/2的发布很近有着关系，时间太短了。

HTTP/3在高丢包的网络中可能表现更好，它提供了更快的握手，所以能改善可感知和实际的延迟。这些进步足够推动人们在服务器和服务上部署HTTP/3的支持吗？时间以及未来的性能测试会给我们答案！

技术标准

以下是QUIC和HTTP/3各个部分的最新官方IETF草案列表。

不变性

Version-Independent Properties of QUIC

传输层

QUIC: A UDP-Based Multiplexed and Secure Transport

自动恢复

QUIC Loss Detection and Congestion Control

TLS

Using Transport Layer Security (TLS) to Secure QUIC

HTTP

Hypertext Transfer Protocol (HTTP) over QUIC

QPACK

QPACK: Header Compression for HTTP over QUIC

QUIC v2

为了尽力专注于QUIC的核心特性，以及为了赶上发布进度，最初计划为核心协议一部分的几个特性已被推迟，计划到QUIC第二版或以后的版本中完成。

本文档的作者买的水晶球是山寨的，所以我们不能知道哪些特性会或者不会出现在第二版中。但我们可以谈谈那些在第一版中被标记为“之后做”的可能出现在第二版中的特性。

前向纠错（Forward Error Correction）

前向纠错（FEC）是一种通过向接收方发送冗余数据，使得接收方能快速识别出含有不明显错误的一种错误控制方式。

Google在它们的原版QUIC成果中实验过该特性，但由于实验结果不理想而已经去除。此特性可供QUIC v2讨论，但可能需要有人证明它在代价不大的情况下确实有用。

多路径（Multipath）

多路径意味着通信可以通过多个网络路径传输，以期最大化利用资源并增加冗余。

SCTP支持者可能有话说了，SCTP和现代TCP早就已经支持这一点。

不可靠数据

通过QUIC传输不可靠的数据流也得到过广泛讨论，这样QUIC就能在传统UDP风格的应用程序中作为代用品。

非HTTP适应

基于QUIC的DNS是早期被提出的非HTTP协议之一，这可能在QUIC v1和HTTP/3发布后得到更多的关注。但我想，如果我们拥有了这样的新型传输层协议，DNS远不是终点。