# Google QUIC协议：从TCP到UDP的Web平台

QUIC（Quick UDP Internet Connections）协议是一种全新的基于UDP的web开发协议。

QUIC(Quick UDP Internet Connections，快速UDP互联网连接)是Google提出的一种基于UDP改进的通信协议，其目的是降低网络通信的延迟，提供更好的用户互动体验。

    QUIC的主要特点包括：具有SPDY(SPDY是谷歌研制的提升HTTP速度的协议，是HTTP/2.0的基础)所有的优点；0-RTT连接，更少的数据包建立连接；减少丢包；前向纠错，减少丢包导致的重传时延；自适应拥塞控制；在网络切换环境下，通过连接标识UUID不变原理减少重新连接；内部实现了TLS加密。

    有关QUIC的详细资料可以参考http://www.chromium.org/quic里的相关文档和代码。

## 从TCP协议说起

当前，web平台的数据传输都基于TCP协议。TCP协议在创建连接之前需要进行[三次握手](https://en.wikipedia.org/wiki/Transmission_Control_Protocol" \l "Connection_establishment)（图1），如果需要提高数据交互的安全性，既增加传输层安全协议（TLS），还会增加更多的握手次数（图2）。

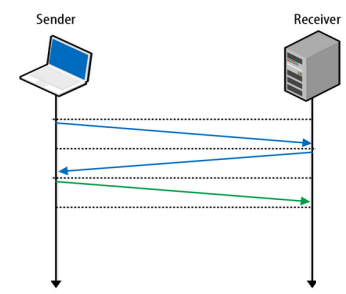


图1，TCP三次握手示意（来源 [Next generation multiplexed transport over UDP (PDF)](https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf)）

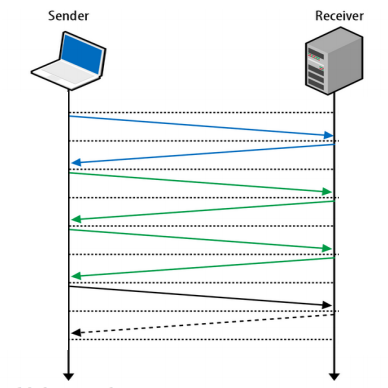


图2，TLS初始化握手示意（来源 [Next generation multiplexed transport over UDP (PDF)](https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf)）

正因为TCP协议连接建立的成本相对较高，可以通过[TCP快速打开](https://zh.wikipedia.org/zh/TCP%E5%BF%AB%E9%80%9F%E6%89%93%E5%BC%80)（TCP Fast Open）来减少建立连接时的握手次数。但是该技术目前应用较少。

和TCP相反，UDP协议是无连接协议。客户端发出UDP数据包后，只能“假设”这个数据包已经被服务端接收。这样的好处是在网络传输层无需对数据包进行确认，但存在的问题就是为了确保数据传输的可靠性，应用层协议需要自己完成包传输情况的确认。

此时，QUIC协议就登场了。**QUIC协议可以在1到2个数据包（取决于连接的服务器是新的还是已知的）内，完成连接的创建（包括TLS）**（图3）。

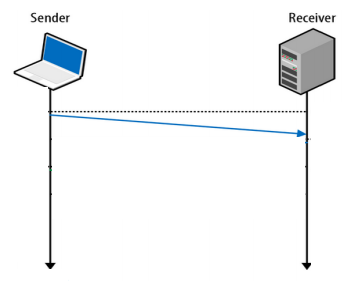


图3，QUIC协议握手示意（来源 [Next generation multiplexed transport over UDP (PDF)](https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf)）

## QUIC协议的目的

从前文对比可以看出，QUIC协议的主要目的，是为了整合TCP协议的可靠性和UDP协议的速度和效率。

[QUIC的维基百科页面](https://en.wikipedia.org/wiki/QUIC)介绍了该协议的主要目的：

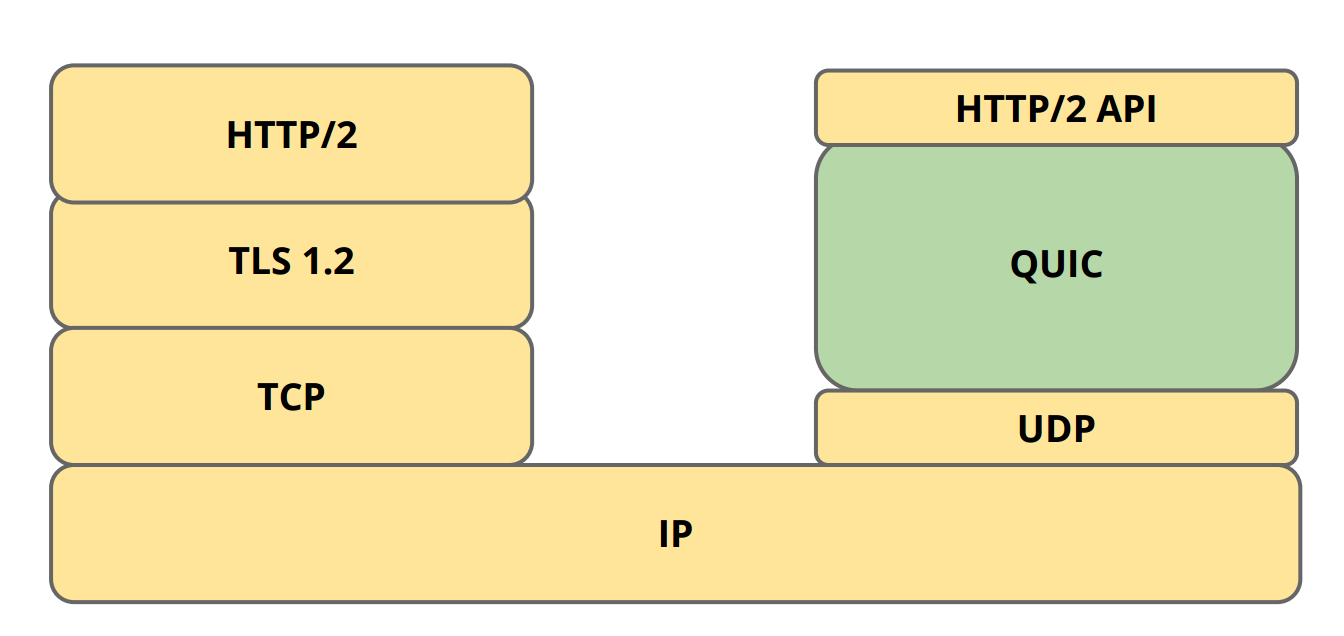
**对于Google来说优化TCP协议是一个长期目标，QUIC旨在创建几乎等同于TCP的独立连接，但有着低延迟，并对类似SPDY的多路复用流协议有更好的支持。 如果QUIC协议的特性被证明是有效的，这些特性以后可能会被迁移入后续版本的TCP和TLS协议（它们都有很长的开发周期）。**

值得注意的是，****如果QUIC的特性被证明是有效的，这些特性以后可能会被迁移到后续版本的TCP协议中****。

TCP协议的实现是高度管制的。TCP协议栈通常由操作系统实现，如Linux、Windows内核或者其他移动设备操作系统。修改TCP协议是一项浩大的工程，因为每种设备、系统的实现都需要更新。

相反的，UDP协议在操作系统层面实现相对简单，基于UDP协议实现新的协议以验证Google对于TCP协议改进的理论，验证成本相对较低。

QUIC协议内置了TLS栈，实现了自己的[传输加密层](https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit)，而没有使用现有的TLS 1.2。同时QUIC还包含了部分HTTP/2的实现，因此QUIC的地位看起来是这样的：

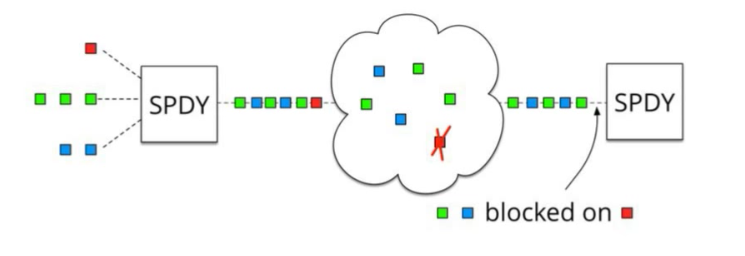


从图上可以看出，**QUIC底层通过UDP协议替代了TCP，上层只需要一层用于和远程服务器交互的HTTP/2 API。这是因为QUIC协议已经包含了多路复用和连接管理，HTTP API只需要完成HTTP协议的解析即可。**

## QUIC特性

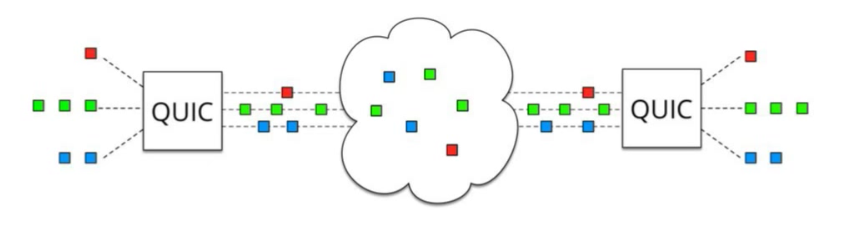
### 避免前序包阻塞

SPDY和HTTP/2协议现在都支持将页面的多个数据（如图片、js等）通过一个数据链接进行传输。该特性能够加快页面组件的传输速度，但是对于TCP协议来说，这会遇到前序包阻塞的问题。这是由于TCP协议在处理包时是有严格顺序的，当其中一个数据包遇到问题，TCP连接需要等待这个包完成重传之后才能继续进行。因此，即使逻辑上一个TCP连接上并行的在进行多路数据传输，其他毫无关联的数据也会因此阻塞。



图片来源 [Next generation multiplexed transport over UDP (PDF)](https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf)

QUIC协议直接通过底层使用UDP协议天然的避免了该问题。由于UDP协议没有严格的顺序，当一个数据包遇到问题需要重传时，只会影响该数据包对应的资源，其他独立的资源（如其他css、js文件）不会受到影响。



图片来源 [Next generation multiplexed transport over UDP (PDF)](https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf)

### 减少数据包

前文已经介绍过QUIC协议在创建连接握手时，只需要1到2个数据包即可。这对于拥有高速互联网连接的网络环境下可能没有太大的感觉，因为此时一个数据包的延时大概在10～50ms之间。

一般来说延迟在50ms之内不会有太大的感觉。但是对于无线网络来说，情况就不太一样了。且不说传统2G/3G网络，即使是4G网络，客户端和服务器之间的延时也通常在100ms以上。传统TCP+TLS协议的传输方式，在创建连接时的4个数据包和QUIC协议的1个数据包相比，连接创建上就会多耗时300ms以上。

### 向前纠错

QUIC协议有一个非常独特的特性，称为向前纠错（Forward Error Correction），每个数据包除了它本身的内容之外，还包括了部分其他数据包的数据，因此少量的丢包可以通过其他包的冗余数据直接组装而无需重传。

这类似网络层的RAID 5！

目前默认的冗余量是10%，既每发送10个数据包，其冗余数据就可以重新构建一个丢失的数据包。

向前纠错牺牲了每个数据包可以发送数据的上限，但是减少了因为丢包导致的数据重传，因为数据重传将会消耗更多的时间（包括确认数据包丢失、请求重传、等待新数据包等步骤的时间消耗）。

### 会话重启和并行下载

底层协议切换到UDP协议之后的另一大好处是，连接不再依赖于来源IP。

对于TCP协议来说，标识一个TCP连接需要4个参数，既来源IP、来源端口、目的IP和目的端口。其中的任一参数改变，TCP连接就需要重新创建。

这对于传统网络来说影响不大，因为来源和目的IP相对固定。但是在无线网络中，情况就大不相同了。设备在移动过程中，可能会因为网络切换（如从WIFI网络切换到4G网络环境），导致TCP连接需要重新创建。

QUIC协议使用了UDP协议，不再需要这四元组参数。同时QUIC协议实现了自己的会话标记方式，称为连接UUID。当设备网络环境切换时，连接UUID不会发生变化，因此无需重新进行握手。

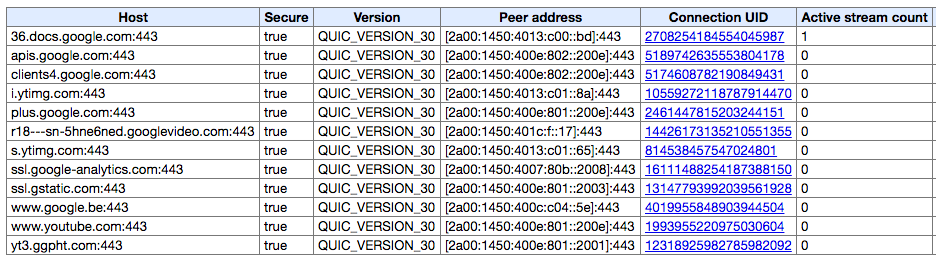
该特性除了可以减少无谓的连接重连之外，还可以充分利用设备的不同网络接口，进行资源的并行下载。因为虽然这些网络接口有不同的IP，但只要他们能够共享连接UUID，就能够并行的从服务器下载数据。

## QUIC协议实践

Chrome浏览器从2014年开始已经实验性的支持了QUIC协议。可以通过在Chrome浏览器中输入chrome://net-internals/#quic查看是否已经支持QUIC协议。如果还未支持，可以在chrome://flags/#enable-quic中进行开启。

开始Chrome浏览器对QUIC协议的支持之后，可以在chrome://net-internals/#quic中查看到当前浏览器的QUIC一些连接。当然目前只有Google服务才支持QUIC协议（如YouTube、 Google.com）。

(点击放大图像)

[](http://cdn4.infoqstatic.com/statics_s2_20170411-0445/resource/articles/quic-google-protocol-web-platform-from-tcp-to-udp/zh/resources/056.png)

### 关于防火墙

通常系统管理员会关注防火墙的TCP规则，而忽略UDP规则。如果要在防火墙之后使用QUIC协议，除了传统web服务需要开放的80/TCP、443/TCP之外，针对QUIC还需要开放443/UDP的访问。

### 服务端使用QUIC协议

目前支持QUIC协议的web服务只有0.9版本以后的[Caddy](https://github.com/mholt/caddy/wiki/QUIC)。其他常用web服务如nginx、apache等都未开始支持。curl表达了对QUIC协议[支持的兴趣](https://daniel.haxx.se/blog/2016/07/20/curl-wants-to-quic/)。

### QUIC性能优势

在2015年的[博文](http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html)中，Google分享了一些关于QUIC协议实现的结果。

这些优势在诸如YouTube的视频服务上更为突出。用户报告通过QUIC协议在观看视频的时候可以减少30%的重新缓冲时间。

如果YouTube收集的报告可靠，可以预见视频服务提供商会更快的采用QUIC协议。

## 总结

QUIC协议开创性的使用了UDP协议作为底层传输协议，通过各种方式减少了网络延迟。

目前QUIC协议已经在运行在最大的网站上，期待QUIC协议规范能够成为终稿，并在其他浏览器和服务器中能够实现。

# 1.重传与恢复

     与TCP类似，QUIC每发送一个包后，都会等待回复一个确认包。当丢包率超过协议的纠错门限时，会显式或隐式地进行重传

    对于某些重要的数据包，如初始密钥协商时的数据包，在建立连接时非常重要，如果这类包丢失会阻塞整体数据流。QUIC对于这一类数据包在确认发生丢失前就会尝试重传，通常是等待较短的时间(如20ms)没收到确认后就马上再次发送。这样在网络中会有若干个相同的包同时传输，只要有一个能成功抵达就完成了连接，这样降低了丢包率。接收方对于关键数据包的多次发送和普通数据包的超时重传，都采用相同的重复包处理机制

     QUIC在拥塞避免**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/this_capslock/article/details/_blank)**的基础上还加入了心跳包，用于减少丢包率

     QUIC使用了FEC(前向纠错码)来恢复数据，FEC采用简单异或的方式。每次发送一组数据，包括若干个数据包后，并对这些数据包依次作异或运算，最后的结果作为一个FEC包再发送出去。接收方收到一组数据后，根据数据包和FEC包即可以进行校验和纠错。

# 2.安全性

    QUIC对每个散装的UDP包都进行了加密和认证的保护，并且避免使用前向依赖的处理方法(如CBC模式)，这样每个UDP包可以独立地根据IV进行加密或认证处理。

    QUIC采用了两级密钥机制：初始密钥和会话密钥。初次连接时不加密，并协商初始密钥。初始密钥协商完毕后会马上再协商会话密钥，这样可以保证密钥的前向安全性，之后可以在通信的过程中就实现对密钥的更新。接收方意识到有新的密钥要更新时，会尝试用新旧两种密钥对数据进行解密，直到成功才会正式更新密钥，否则会一直保留旧密钥有效。

# 3. 0-RTT握手过程

    QUIC握手的过程是需要一次数据交互，0-RTT时延即可完成握手过程中的密钥协商，比TLS相比效率提高了5倍，且具有更高的安全性。

    QUIC在握手过程中使用Diffie-Hellman算法协商初始密钥，初始密钥依赖于服务器存储的一组配置参数，该参数会周期性的更新。初始密钥协商成功后，服务器会提供一个临时随机数，双方根据这个数再生成会话密钥。

    具体握手过程如下：

    (1) 客户端判断本地是否已有服务器的全部配置参数，如果有则直接跳转到(5)，否则继续

    (2) 客户端向服务器发送inchoate client hello(CHLO)消息，请求服务器传输配置参数

    (3) 服务器收到CHLO，回复rejection(REJ)消息，其中包含服务器的部分配置参数

    (4) 客户端收到REJ，提取并存储服务器配置参数，跳回到(1)

    (5) 客户端向服务器发送full client hello消息，开始正式握手，消息中包括客户端选择的公开数。此时客户端根据获取的服务器配置参数和自己选择的公开数，可以计算出初始密钥。

    (6) 服务器收到full client hello，如果不同意连接就回复REJ，同(3)；如果同意连接，根据客户端的公开数计算出初始密钥，回复server hello(SHLO)消息，SHLO用初始密钥加密，并且其中包含服务器选择的一个临时公开数。

    (7) 客户端收到服务器的回复，如果是REJ则情况同(4)；如果是SHLO，则尝试用初始密钥解密，提取出临时公开数

    (8) 客户端和服务器根据临时公开数和初始密钥，各自基于SHA-256算法推导出会话密钥

    (9) 双方更换为使用会话密钥通信，初始密钥此时已无用，QUIC握手过程完毕。之后会话密钥更新的流程与以上过程类似，只是数据包中的某些字段略有不同。

