C/C++Linux服务器开发

高级架构师课程

三年课程沉淀 | 五次精益升级 | 十年行业积累 | 百个实战项目 | 十万内容受众

讲师:darren/326873713

讲师介绍--专业来自专注和实力

King老师

系统架构师，曾供职著名创业公司系统架构师，微软亚洲研究院、创维集团全球研发中心。国内第一代商业Paas平台开发者。著有多个软件专利，参与多个开源软件维护。在全球化，高可用的物联网云平台架构与智能硬件设计方面有丰富的研发与实战经验。

Darren老师

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

QUIC协议

1.认识QUIC协议

2.QUIC报文格式

3.QUIC特点分析

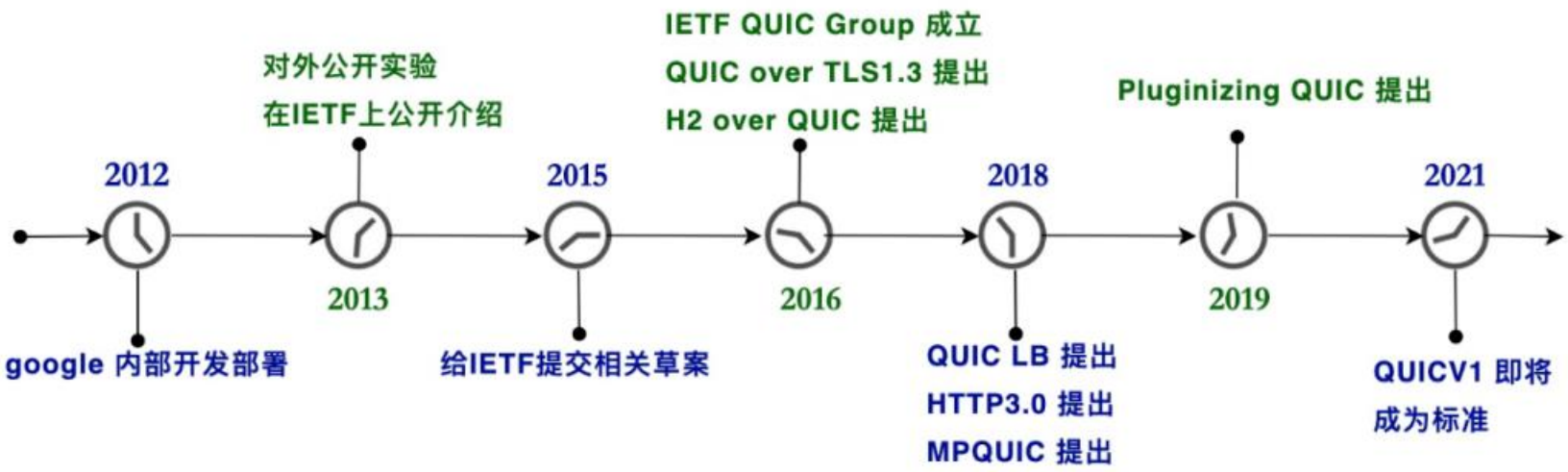
4.QUIC开源库和应用

5.QUIC面临的挑战

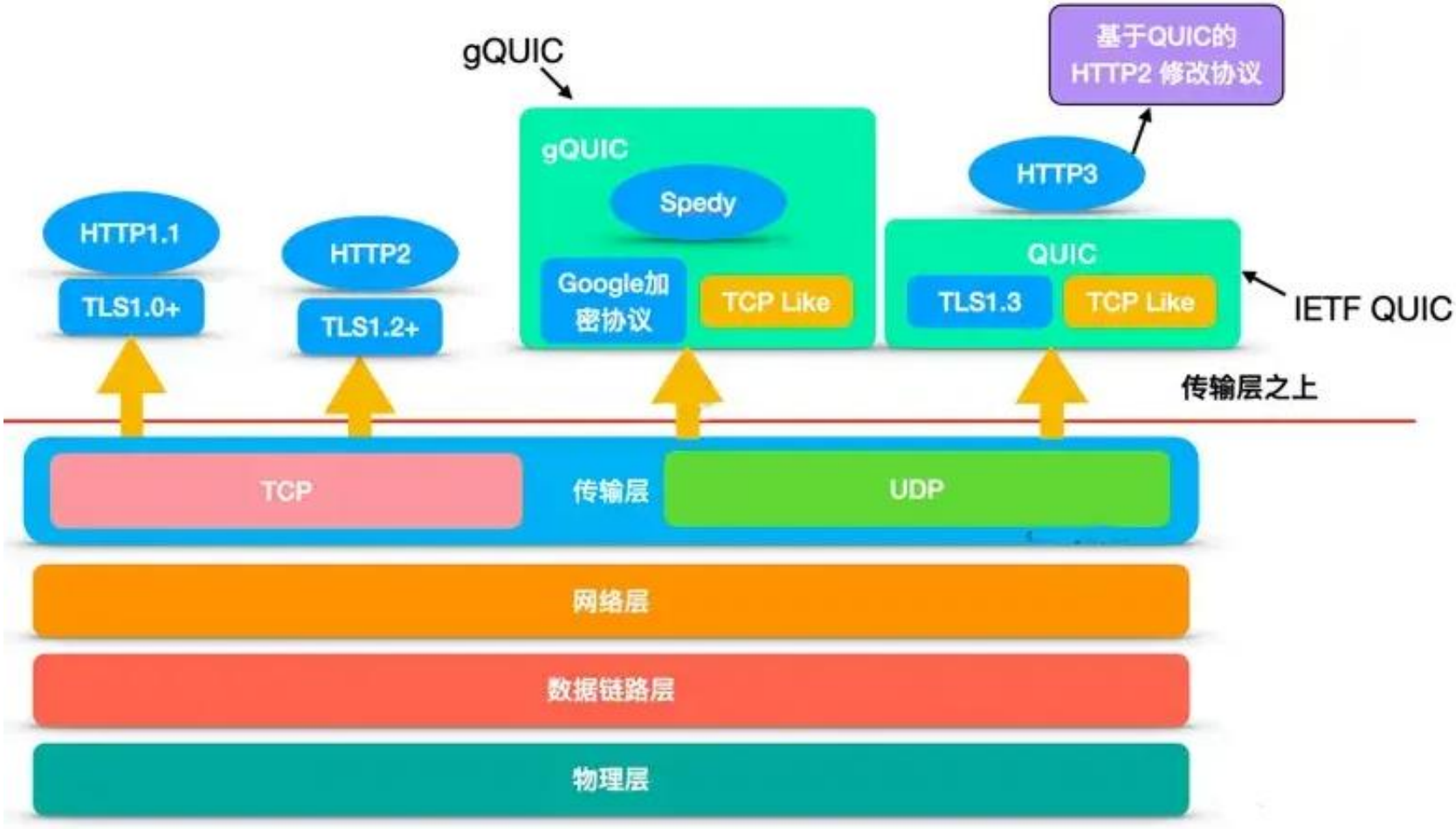
1.1 QUIC前世今生

- QUIC，即 快速UDP网络连接 (Quick UDP Internet Connections)， 是由 Google 提出的实验性网络传输协议，位于 OSI 模型传输层。 QUIC 旨在解决 TCP 协议的缺陷，并最终替代 TCP 协议， 以减少数据传输，降低连接建立延迟时间，加快网页传输速度。
- QUIC诞生于2012年

标准文档地址：https://quicwg.org/base-drafts/rfc9000.html



1.2 QUIC框图


gQUIC
基于QUIC的
HTTP2 修改协议
gQUIC
Spedy
HTTP3
HTTP1.1
TLS1.0+
HTTP2
TLS1.2+
Google加
密协议
TCP Like
QUIC
TLS1.3
TCP Like
IETF QUIC
传输层之上
TCP
传输层
UDP
网络层
数据链路层
物理层

1.3 QUIC为什么在应用层实现

- 新的传输层协议通常会经过严格的设计，分析和评估可重复的结果，证明候选协议对现有协议的正确性和公平性，开发新的传输层协议和它在操作系统进行广泛部署之间通常需要花费数年的时间。

- 再者，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，这些设备很多只认TCP和UDP。如果使用另一种传输层协议，那么就会有可能无法建立连接或者报文无法转发，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。

1.4 QUIC协议术语

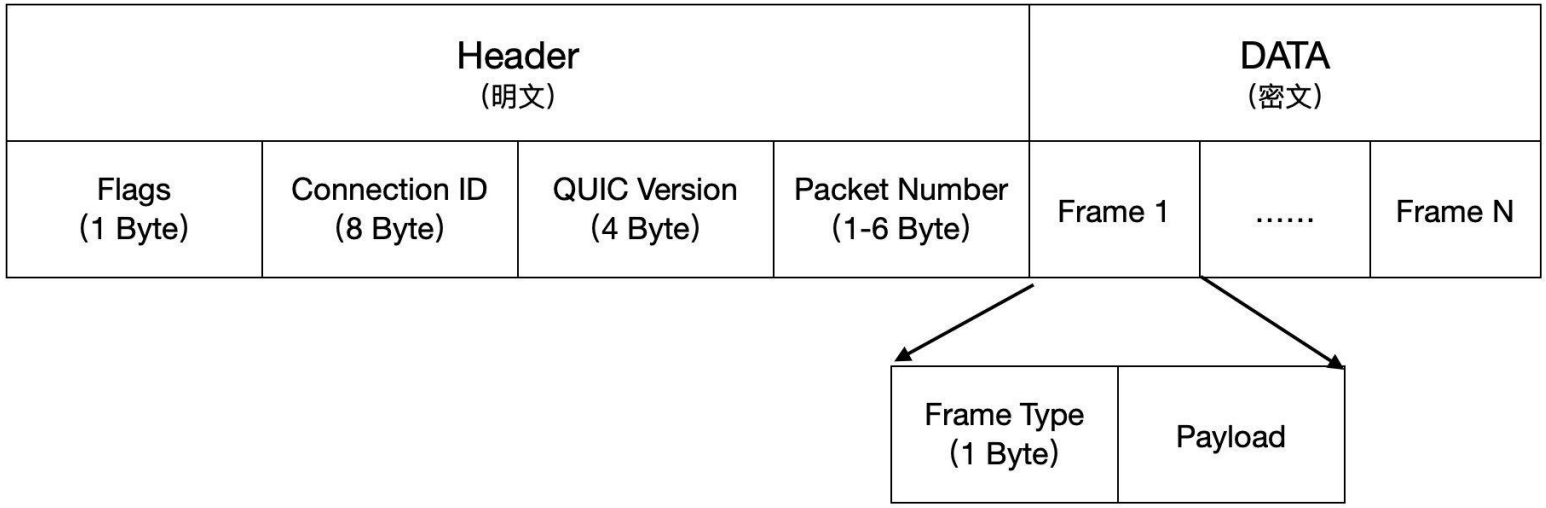
- **QUCI连接**：Client和Server之间的通信关心，Client发起连接，Server接受连接
- **流（Stream）**：一个QUIC连接内，单向或者双向的有序字节流。一个QUIC连接可以同时包含多个Stream
- **帧（Frame）**：QUIC连接内的最小通信单元。一个QUIC数据包（packet）中的数据部分包含一个或多个帧

1.5 QUIC和TCP对比

	TCP	QUIC
握手延迟	TCP需要三次握手+TLS握手	传输层握手和TLS握手放到一起，真正实现0RTT
头阻塞问题	TCP使用连接级别的seq保序，丢失一个报文，影响所有的应用数据包	基于UDP封装传输层Stream，Stream内部保序，Stream之间相互不影响，无头阻塞
连接迁移	内核在五元组维持session，所以五元组改变了需要重建连接	UDP不面向连接，QUIC基于CID标识连接，五元组发生变化，CID不变，Session状态依然维持
特性迭代速度	因为在内核实现，所以迭代很慢	在用户态实现，不用改内核，可快速演进新特性
安全性	TCP Header是完全明文传输	除了一些必要的字段，Header也被加密。

2 QUIC报文格式

一个 QUIC 数据包的格式



- Header 是明文的，包含 4 个字段：Flags、Connection ID、QUIC Version、Packet Number；
- Data 是加密的，可以包含 1 个或多个 frame，每个 frame 又分为 type 和 payload，其中 payload 就是应用数据

2.1 QUIC报文格式-Stream帧1

数据帧有很多类型：Stream、ACK、Padding、Window_Update、Blocked 等，这里重点介绍下用于传输应用数据的 Stream 帧。

<table>
<tr><td colspan="5">Stream Frame
（数据流帧）</td></tr>
<tr><td rowspan="2">Frame Type
(1 Byte)</td><td colspan="4">Payload</td></tr>
<tr><td>Stream ID
(1-4 Byte)</td><td>offset
(0-8 Byte)</td><td>Data Length
(2 Byte)</td><td>Data
（应用数据）</td></tr>
</table>

Frame Type： 帧类型，占用 1 个字节

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

（1）Bit7：必须设置为 1，表示 Stream 帧

（2）Bit6：如果设置为 1，表示发送端在这个 stream 上已经结束发送数据，流将处于半关闭状态

（3）Bit5：如果设置为 1，表示 Stream 头中包含 Data length 字段

（4）Bit432：表示 offset 的长度。000 表示 0 字节，001 表示 2 字节，010 表示 3 字节，以此类推

（5）Bit10：表示 Stream ID 的长度。00 表示 1 字节，01 表示 2 字节，10 表示 3 字节，11 表示 4 字节

2.1 QUIC报文格式-Stream帧2

<table>
<tr><td colspan="5">Stream Frame
(数据流帧)</td></tr>
<tr><td rowspan="2">Frame Type
(1 Byte)</td><td colspan="4">Payload</td></tr>
<tr><td>Stream ID
(1-4 Byte)</td><td>offset
(0-8 Byte)</td><td>Data Length
(2 Byte)</td><td>Data
(应用数据)</td></tr>
</table>

Stream ID：流 ID，用于标识数据包所属的流。后面的流量控制和多路复用会涉及到。

Offset：偏移量，表示该数据包在整个数据中的偏移量，用于数据排序。

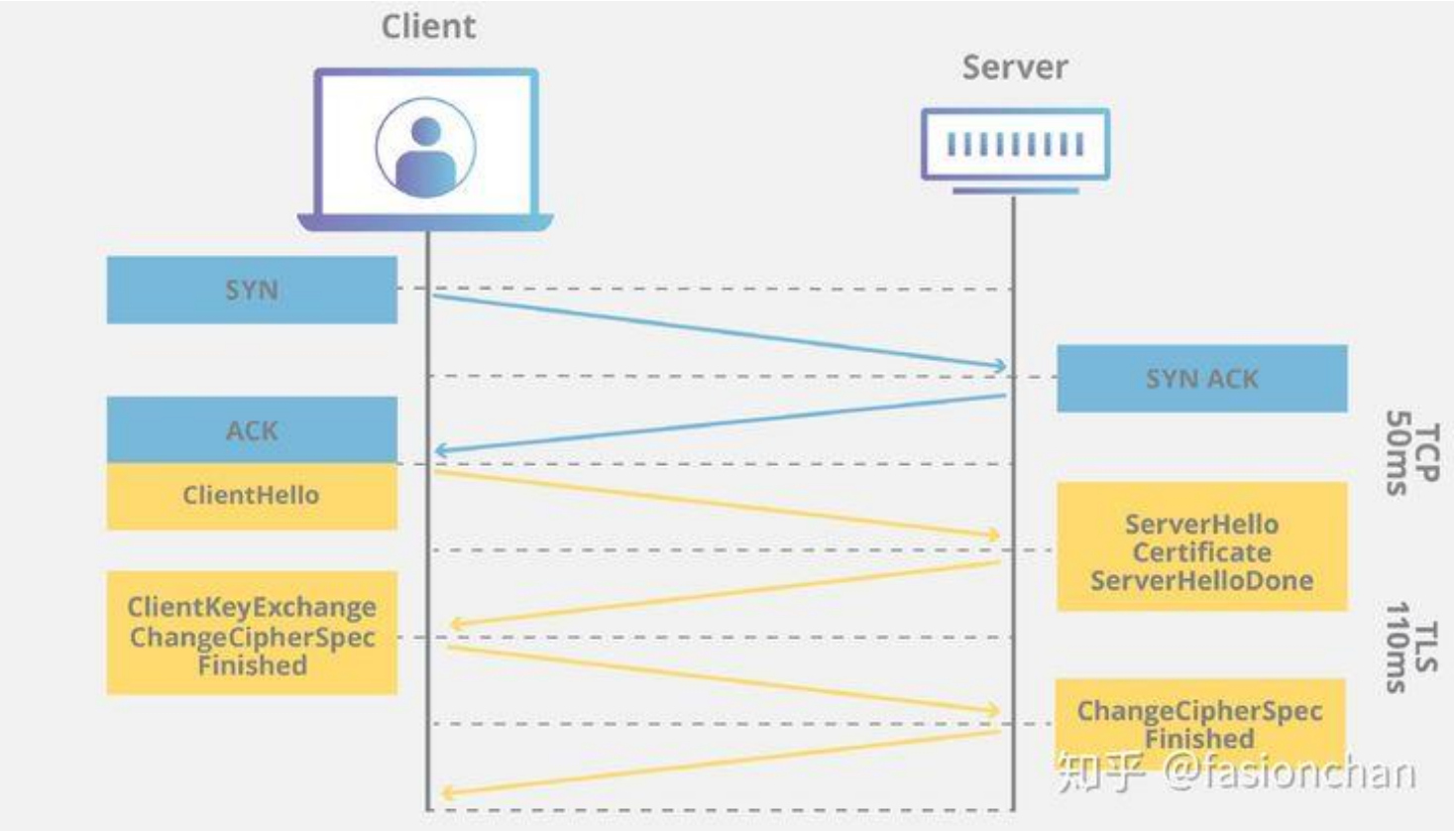
Data Length： 数据长度，占用 2 个字节，表示实际应用数据的长度

Data： 实际的应用数据

3 QUIC的特点

1. 连接建立低时延
2. 多路复用
3. 无队头阻塞
4. 灵活的拥塞控制机制
5. 连接迁移
6. 数据包头和包内数据的身份认证和加密
7. FEC前向纠错
8. 可靠性传输
9. 其他

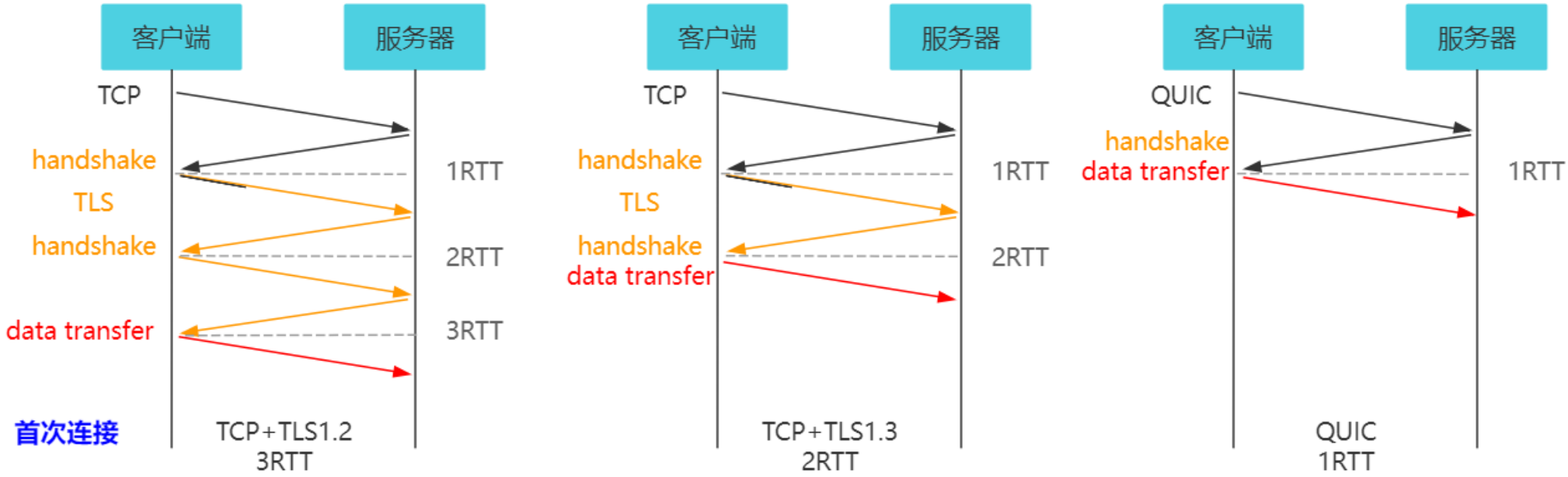
3.1 连接建立低时延-典型TCP+TLS连接



1.首先，执行三次握手，建立 *TCP* 连接(蓝色部分);
2.然后，执行 *TLS* 握手，建立 *TLS* 连接(黄色部分);
3.此后开始传输业务数据;

客户端和服务器之间要进行多轮协议交互，才能建立 *TLS* 连接，延迟相当严重。
平时访问 *https* 网站明显比 *http* 网站慢，三次握手和 *TLS* 握手难辞其咎。

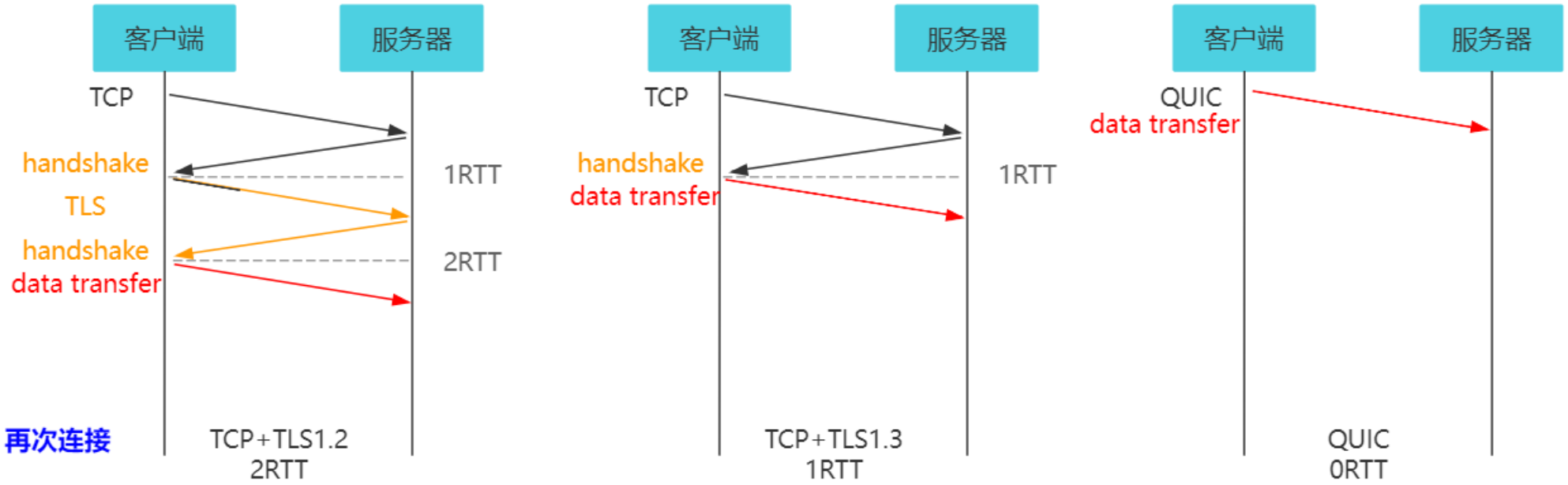
3.1 连接建立低时延-QUIC首次连接



- 共3RTT：TCP+TLS1.2中：TCP三次握手建立连接需要1个RTT；TLS需要2个RTT完成身份验证；传输数据
- 共2RTT：TCP+TLS1.3中：TCP三次握手建立连接需要1个RTT；TLS需要1个RTT完成身份验证；传输数据
- 共1RTT： 首次QUIC连接中，Client 向 Server 发送消息，请求传输配置参数和加密相关参数；Server 回复其配置参数；传输数据

注解:
注意到，三次握手中的 ACK 包与 handshake 合并在一起发送。 这是 TCP 实现中使用的 **延迟确认** 技术，旨在减少协议开销，改善网络性能。

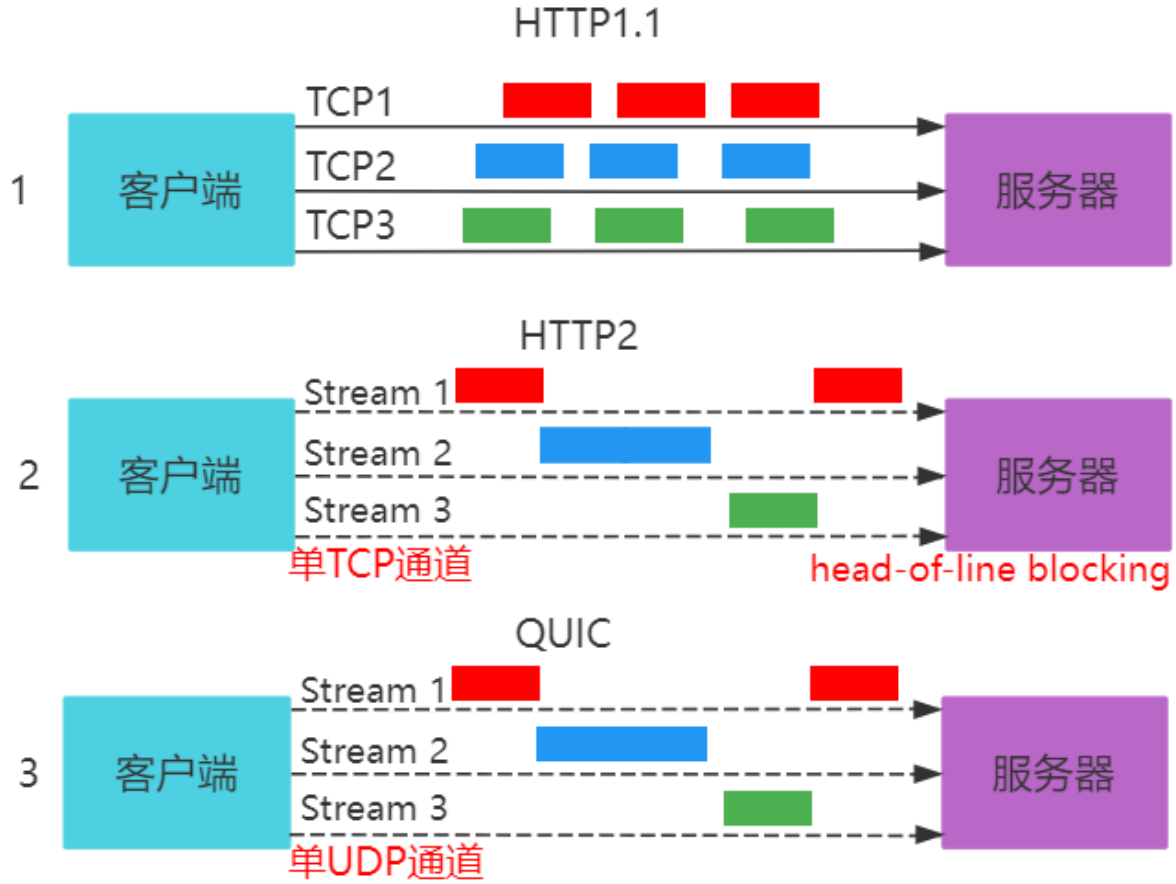
3.1 连接建立低时延-QUIC再次连接



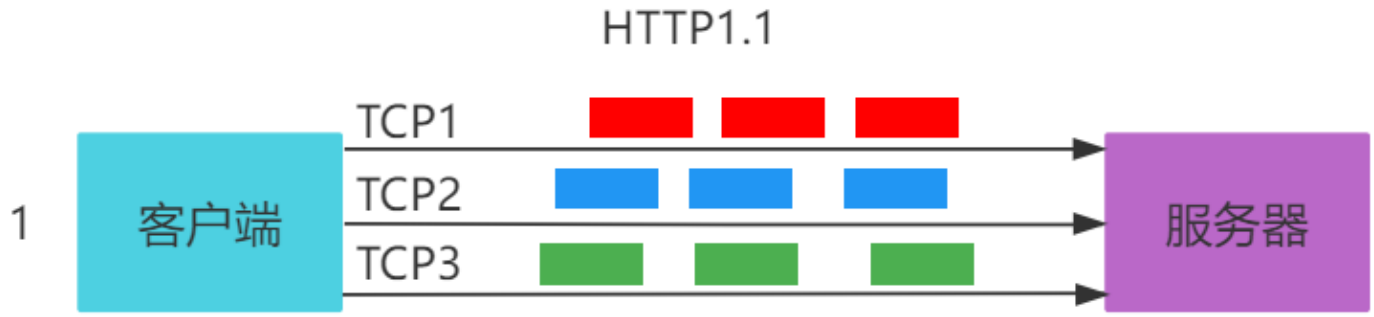
- 再次连接的概念：Client已经访问过Server，在本地存放了Cookie。
- 2RTT：TCP+TLS1.2中：TCP三次握手建立连接需要1个RTT；TLS需要1个RTT完成身份验证（由于缓存的存在，减少1RTT）；传输数据
- 1RTT：TCP+TLS1.3 中: TCP三次握手建立连接需要1个RTT；传输数据
- 0RTT：在客户端与服务端的再次QUIC连接中，Client 本地已有 Server 的全部配置参数（缓存）,据此计算出初始密钥，直接发送加密的数据包。

3.2 多路复用

在一个网页里面总是会有多个数据要传输，总是希望多个数据能够同时传输，以此来提高用户的体验



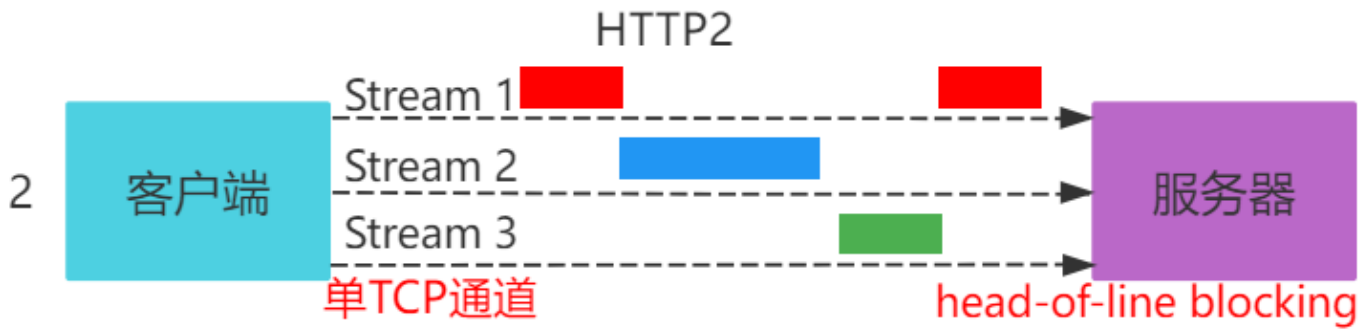
3.2 流复用-HTTP1.1



在HTTP1.1中：

- 每个TCP连接同时只能处理一个请求一响应，为了提高响应速度，需要同时创建多个连接，但是多个连接管理比较复杂

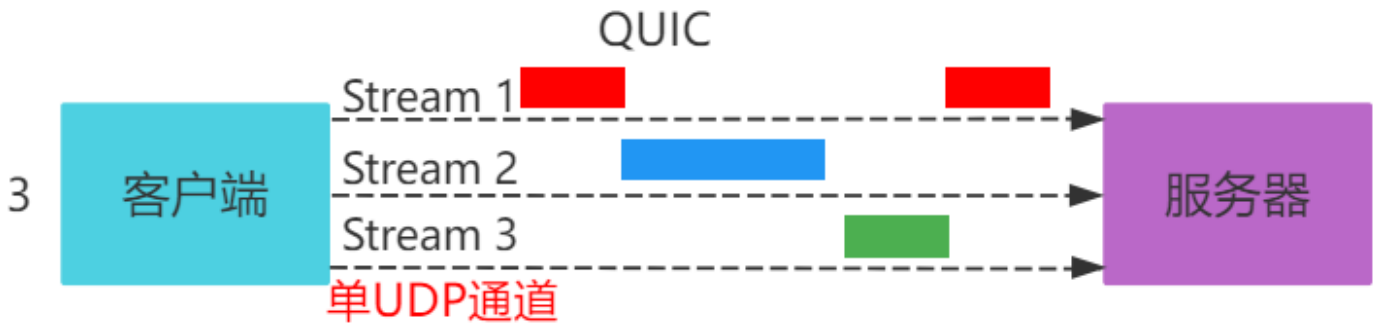
3. 2 流复用-HTTP2



在HTTP2中：

- 每个TCP连接里面有多个逻辑上独立的多个流（stream）
- 每个流可以传输不同的文件数据
- 解决了HTTP1一个连接无法同时传输多个数据的问题
- 缺点：容易出现队头阻塞问题

3.2 流复用-HTTP3(QUIC)

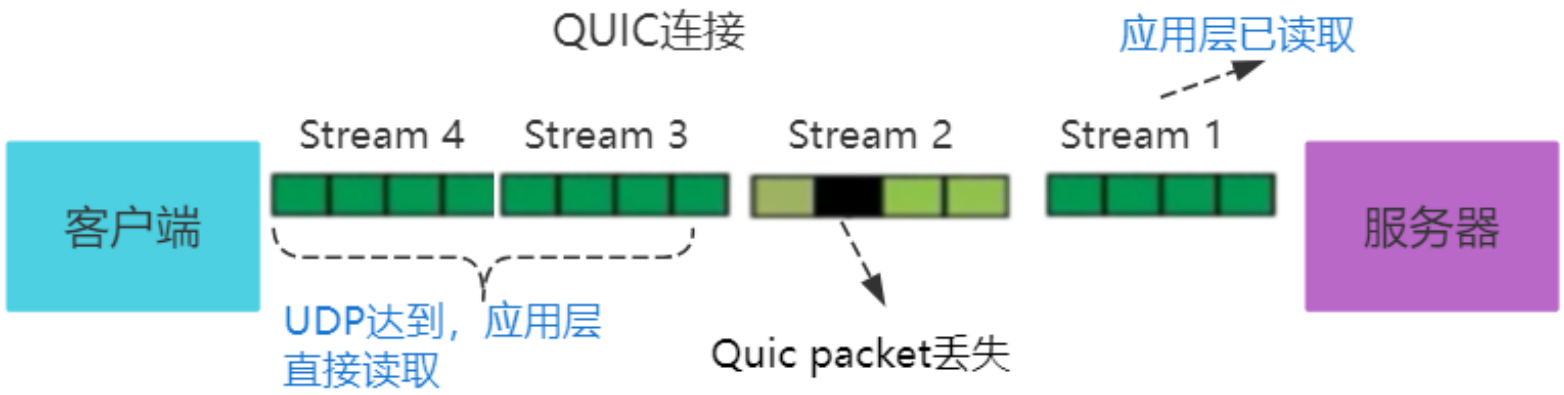


而在基于QUIC的HTTP3中：

- 借鉴了HTTP2中流的概念
- 流之间互相独立，即不同流之间的数据之间交付顺序无关（如果stream2的数据丢失，只会影响排在stream2后面的数据，stream1和stream3的数据不会被影响）
- 建立在UDP之上，没有依赖性

3. 3 无队头阻塞

HTTP/2会出现队头阻塞问题，而在基于QUIC的HTTP/3中则很好地解决这一问题



- UDP中的各个数据报独立，基于UDP的QUIC中的各个stream独立
- 即使stream2里有一个包丢失，因为stream之间互相独立无关联，所以不会阻塞stream3、stream4，依然可以交付给上层

3. 4 灵活的拥塞控制机制

TCP 的拥塞控制实际上包含了四个算法：慢启动，拥塞避免，快速重传，快速恢复

•New Reno：基于丢包检测
•CUBIC：基于丢包检测
•BBR：基于网络带宽

QUIC 协议当前默认使用了 TCP 协议的 Cubic 拥塞控制算法。
这个机制还是可插拔的，什么叫可插拔呢？就是能够非常灵活地生效，变更和停止，可以根据场景来切换不同的方法

QUIC协议在用户空间实现，应用程序不需要停机和升级就能实现拥塞控制的变更，在服务端只需要修改一下配置，reload 一下，完全不需要停止服务就能实现拥塞控制的切换。甚至可以为每一个请求都设置一种拥塞控制算法。

而TCP在内核态，其拥塞控制难以进行修改和升级

3. 5 连接迁移1

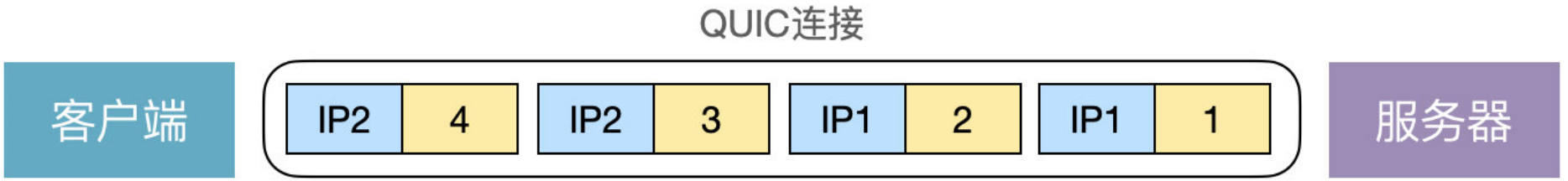
场景：

1. **使用WIFI上网的时候突然网络不好，于是切换成移动数据模式；又或者没有流量上不了网，于是连接WIFI**
2. **户外4G基站的切换，比如在做地铁时正在语音通话，不同的站点会切换不同的基站。对于5G基站则切换更为频繁。**

连接迁移：当客户端切换网络时，和服务器的连接并不会断开，仍然可以正常通信，对于 TCP 协议而言，这是不可能做到的。因为 TCP 的连接基于 4 元组：源 IP、源端口、目的 IP、目的端口，只要其中 1 个发生变化，就需要重新建立连接。但 QUIC 的连接是基于 64 位的 Connection ID，网络切换并不会影响 Connection ID 的变化，连接在逻辑上仍然是通的。

IP地址或者端口号发生变化时，只要ID不变，依然能够维持原有连接，上层业务逻辑感知不到变化，不会中断

3.5 连接迁移2



假设客户端先使用 IP1 发送了 1 和 2 数据包，之后切换网络，IP 变更为 IP2，发送了 3 和 4 数据包，服务器根据数据包头部的 Connection ID 字段可以判断这 4 个包是来自于同一个客户端。QUIC 能实现连接迁移的根本原因是底层使用 UDP 协议就是面向无连接的。

3.6 数据包头和包内数据的身份认证和加密

相比于TCP，QUIC的安全性是内置的，也就是说是必须的。

在恶意环境下性能与TLS类似，友好环境下优于TLS，因为TLS使用一个会话密钥，如果这个密钥被截获的话就不能保证之前数据的安全性

而QUIC使用两个密钥——初始密钥和会话密钥，并且QUIC提供密码保护，TLS不提供

除此之外，QUIC的包头经过身份认证，包内数据是加密的。这样如果QUIC数据被恶意修改的话接收端是可以发现的，降低了安全风险

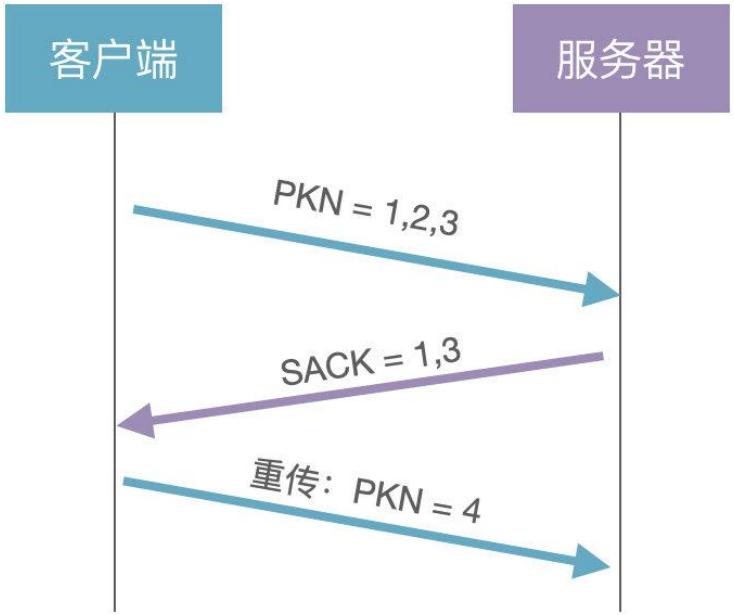
而且数据加密之后，可以通过像防火墙或者nat这些中间件

3.7 FEC前向纠错

FEC是Forward Error Correction前向错误纠正的意思，就是通过多发一些冗余的包，当有些包丢失时，可以通过冗余的包恢复出来，而不用重传。这个算法在多媒体网关拥塞控制有重要的地位。QUIC的FEC是使用的XOR的方式，即发N + 1个包，多发一个冗余的包，在正常数据的N个包里面任意一个包丢了，可以通过这个冗余的包恢复出来，使用异或可以做到切换网络操持连接。

3.8 可靠性传输-应答

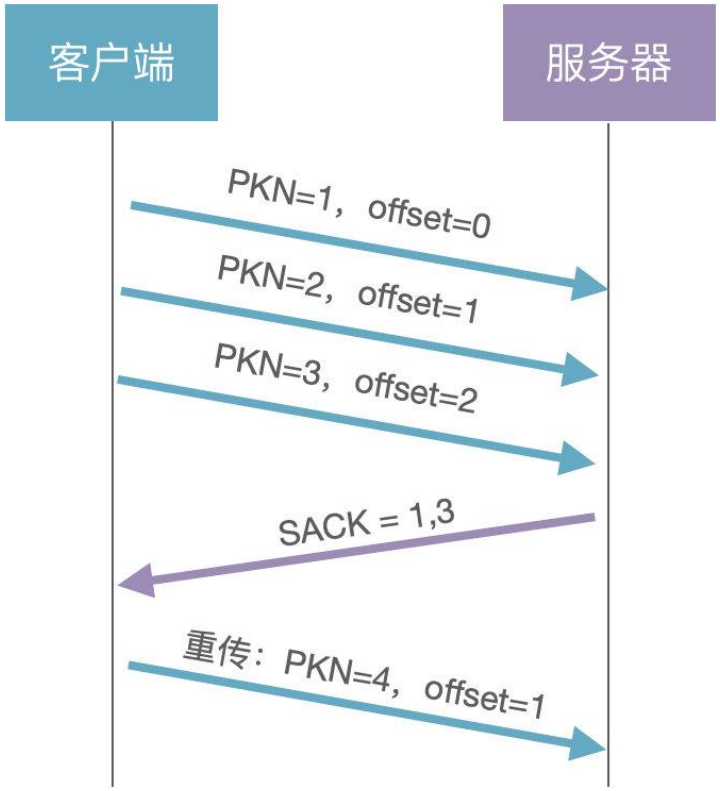
QUIC 是基于 UDP 协议的，而 UDP 是不可靠传输协议，那 QUIC 是如何实现可靠传输的呢？
可靠传输有 2 个重要特点：
（1）完整性：发送端发出的数据包，接收端都能收到
（2）有序性：接收端能按序组装数据包，解码得到有效的数据
问题 1：发送端怎么知道发出的包是否被接收端收到了？
解决方案：通过包号（PKN）和确认应答（SACK）



（1）客户端：发送 3 个数据包给服务器（PKN = 1，2，3）
（2）服务器：通过 SACK 告知客户端已经收到了 1 和 3，没有收到 2
（3）客户端：重传第 2 个数据包（PKN=4）
由此可以看出，QUIC 的数据包号是单调递增的。也就是说，之前发送的数据包（PKN=2）和重传的数据包（PKN=4），虽然数据一样，但包号不同。

3.8 可靠性传输-有序

问题 2：既然包号是单调递增的，那接收端怎么保证数据的有序性呢？
解决方案：通过数据偏移量 offset
每个数据包都有一个 offset 字段，表示在整个数据中的偏移量。



接收端根据 offset 字段就可以对异步到达的数据包进行排序了。为什么 QUIC 要将 PKN 设计为单调递增？解决 TCP 的重传歧义问题：
由于原始包和重传包的序列号是一样的，客户端不知道服务器返回的 ACK 包到底是原始包的，还是重传包的。但 QUIC 的原始包和重传包的序列号是不同的，也就可以判断 ACK 包的归属。

4 QUIC开源库和应用

1. google的gquic 起源最早, 不过它不是单独项目, 代码在chromium项目里边, 用的是c++写的, 可能不是很适合.
2. 微软的msquic, 用c写的, 跨平台, 不过开始得比较晚（好像2020才开始, 不是很成熟).
3. facebook的quic 用的是c++写的. 暂不考虑.
4. nginx的quic 没有自带client, 但它可与ngtcp2联调.
5. litespeed的 lsquic 是基于MIT的, 开始于2017年, 还算比较稳定, 用c语言编写, 各主流平台都有通过测试, 有server/client/lib, 它用于自家的各种产品，暂时看上去是最合适的.
6. ngtcp2, 它是一个实验性质的quic client, 很简洁, 实现了几乎每一版ietf draft. 从代码简洁性上来看, 它无疑是最好的。目前srs流媒体服务器、curl等开源项目有基于ngtcp2做二次开发。

	全接管	半接管	全开放
msquic	Y		
mvfst	Y		
google-quiche		Y	
cloudflare-quiche		Y	
lsquic		Y	
ngtcp2			Y

全开放，开放更多的接口给到外部，方便

4 QUIC开源库和应用-ngtcp2

1. https://github.com/ngtcp2/ngtcp2
2. 采用c语言实现
3. 范例client和server使用了c++17的特性，我们需要升级编译器(比如, clang >= 8.0, 或 gcc >= 8.0).
4. 编译文档见《QUIC开源库安装和实践.pdf》

5 QUIC面临的挑战

1. 路由封杀UDP 443端口（这正是QUIC部署的端口）
2. UDP包过多，由于QoS（Quality of Service，服务质量）限定，会被服务器商认为是攻击，UDP包被丢弃
3. 无论是路由器还是防火墙目前对QUIC都还没做好准备
4. QUIC有较多的开源库，标准尚未真正落地，对应的开源库没有经过充分的验证，不建议中小厂商在QUIC上完全跟进。