技术扫盲：新一代基于UDP的低延时网络传输层协议——QUIC详解

JackJiang Lv.9 7 个月前

本文来自腾讯资深研发工程师罗成的技术分享，主要介绍 QUIC 协议产生的背景和核心特性等。

1、写在前面

如果你的 App，在不需要任何修改的情况下就能提升 15% 以上的访问速度。特别是弱网络的时候能够提升 20% 以上的访问速度。

如果你的 App，在频繁切换 4G 和 WIFI 网络的情况下，不会断线，不需要重连，用户无任何感知。如果你的 App，既需要 TLS 的安全，也想实现 HTTP2 多路复用的强大。

如果你刚刚才听说 HTTP2 是下一代互联网协议，如果你刚刚才关注到 TLS1.3 是一个革命性具有里程碑意义的协议，但是这两个协议却一直在被另一个更新兴的协议所影响和挑战。

如果这个新兴的协议，它的名字就叫做“快”，并且正在标准化为新一代的互联网传输协议。

你愿意花一点点时间了解这个协议吗？你愿意投入精力去研究这个协议吗？你愿意全力推动业务来使用这个协议吗？

2、相关文章

《技术扫盲：新一代基于UDP的低延时网络传输层协议——QUIC详解》
《让互联网更快：新一代QUIC协议在腾讯的技术实践分享》
《七牛云技术分享：使用QUIC协议实现实时视频直播0卡顿！》

3、本文作者

罗成： 腾讯资深研发工程师。目前主要负责腾讯 stgw(腾讯安全云网关）的相关工作，整体推进腾讯内部及腾讯公有云，混合云的七层负载均衡及全站 HTTPS 接入。对 HTTPS，SPDY，HTTP2，QUIC 等应用层协议、高性能服务器技术、云网络技术、用户访问速度、分布式文件传输等有较深的理解。（作者微博：点此进入）

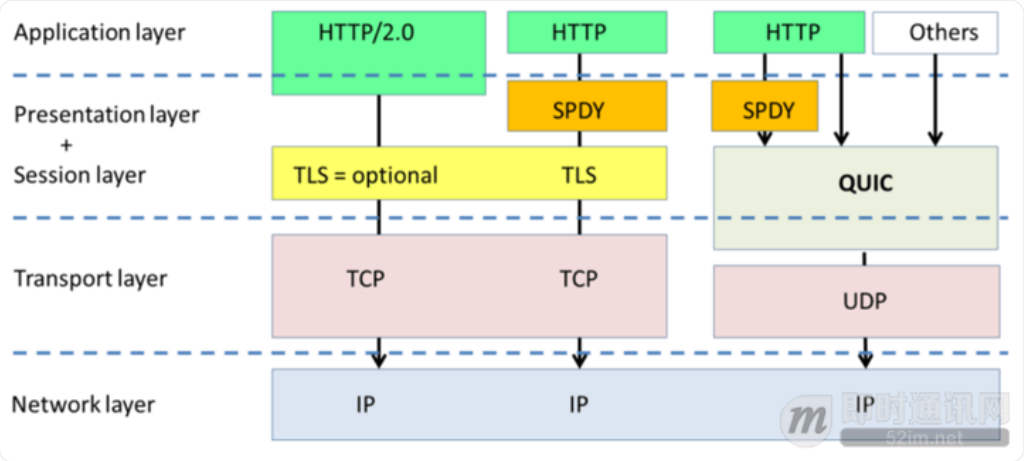
4、QUIC 协议概述

Quic 全称 quick udp internet connection [1]，“快速 UDP 互联网连接”，（和英文 quick 谐音，简称“快”）是由 Google 提出的使用 udp 进行多路并发传输的协议。

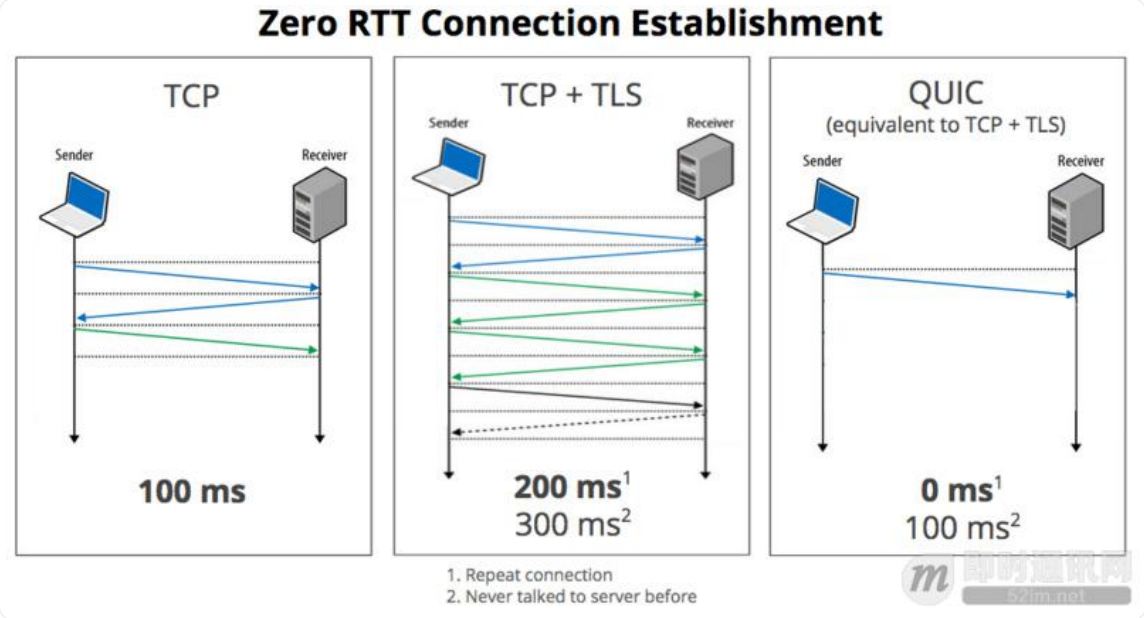
Quic 相比现在广泛应用的 http2+tcp+tls 协议有如下优势 [2]：

- 减少了 TCP 三次握手及 TLS 握手时间；
- 改进的拥塞控制；
- 避免队头阻塞的多路复用；
- 连接迁移；
- 前向冗余纠错。

下图是网络层对比图：



下图是通信延迟对比图：



5、为什么需要 QUIC

5.1 概述

从上个世纪 90 年代互联网开始兴起一直到现在，大部分的互联网流量传输只使用了几个网络协议。使用 IPv4 进行路由，使用 TCP 进行连接层面的流量控制，使用 SSL/TLS 协议实现传输安全，使用 DNS 进行域名解析，使用 HTTP 进行应用数据的传输。

而且近三十年来，这几个协议的发展都非常缓慢。TCP 主要是拥塞控制算法的改进，SSL/TLS 基本上停留在原地，几个小版本的改动主要是密码套件的升级，TLS1.3[3] 是一个飞跃式的变化，但截止到今天，还没有正式发布。IPv4 虽然有一个大的进步，实现了 IPv6，DNS 也增加了一个安全的 DNSSEC，但和 IPv6 一样，部署进度较慢。

随着移动互联网快速发展以及物联网的逐步兴起，网络交互的场景越来越丰富，网络传输的内容也越来越庞大，用户对网络传输效率和 WEB 响应速度的要求也越来越高。

一方面是历史悠久使用广泛的古老协议，另外一方面用户的使用场景对传输性能的要求又越来越高。

如下几个由来已久的问题和矛盾就变得越来越突出：

- 协议历史悠久导致中间设备僵化；
- 依赖于操作系统的实现导致协议本身僵化；
- 建立连接的握手延迟大；
- 队头阻塞。

这里分小节简单说明一下。

5.2 中间设备的僵化

可能是 TCP 协议使用得太久，也非常可靠。所以我们很多中间设备，包括防火墙、NAT 网关，整流器等出现了一些约定俗成的动作。

比如有些防火墙只允许通过 80 和 443，不放通其他端口。NAT 网关在转换网络地址时重写传输层的头部，有可能导致双方无法使用新的传输格式。整流器和中间代理有时候出于安全的需要，会删除一些它们不认识的选项字段。

TCP 协议本来是支持端口、选项及特性的增加和修改。但是由于 TCP 协议和知名端口及选项使用的历史太悠久，中间设备已经依赖于这些潜规则，所以对这些内容的修改很容易遭到中间环节的干扰而失败。

而这些干扰，也导致很多在 TCP 协议上的优化变得小心谨慎，步履维艰。

5.3 依赖于操作系统的实现导致协议僵化

TCP 是由操作系统在内核西方栈层面实现的，应用程序只能使用，不能直接修改。虽然应用程序的更新迭代非常快速和简单。但是 TCP 的迭代却非常缓慢，原因就是操作系统升级很麻烦。

现在移动终端更加流行，但是移动端部分用户的操作系统升级依然可能滞后数年时间。PC 端的系统升级滞后得更加严重，windows xp 现在还有大量用户在使用，尽管它已经存在快 20 年。

服务端系统不依赖用户升级，但是由于操作系统升级涉及到底层软件和运行库的更新，所以也比较保守和缓慢。

这也就意味着即使 TCP 有比较好的特性更新，也很难快速推广。比如 TCP Fast Open。它虽然 2013 年就被提出了，但是 Windows 很多系统版本依然不支持它。

5.4 建立连接的握手延迟大

不管是 HTTP1.0/1.1 还是 HTTPS，HTTP2，都使用了 TCP 进行传输。HTTPS 和 HTTP2 还需要使用 TLS 协议来进行安全传输。

这就出现了两个握手延迟：

- 1) TCP 三次握手导致的 TCP 连接建立的延迟；
- 2) TLS 完全握手需要至少 2 个 RTT 才能建立，简化握手需要 1 个 RTT 的握手延迟。

对于很多短连接场景，这样的握手延迟影响很大，且无法消除。

5.5 队头阻塞

队头阻塞主要是 TCP 协议的可靠性机制引入的。TCP 使用序列号来标识数据的顺序，数据必须按照顺序处理，如果前面的数据丢失，后面的数据就算到达了也不会通知应用层来处理。

另外 TLS 协议层面也有一个队头阻塞，因为 TLS 协议都是按照 record 来处理数据的，如果一个 record 中丢失了数据，也会导致整个 record 无法正确处理。

概括来讲，TCP 和 TLS1.2 之前的协议存在着结构性的问题，如果继续在现有的 TCP、TLS 协议之上实现一个全新的应用层协议，依赖于操作系统、中间设备还有用户的支持。部署成本非常高，阻力非常大。

所以 QUIC 协议选择了 UDP，因为 UDP 本身没有连接的概念，不需要三次握手，优化了连接建立的握手延迟，同时在应用程序层面实现了 TCP 的可靠性，TLS 的安全性和 HTTP2 的并发性，只需要用户端和服务端的应用程序支持 QUIC 协议，完全避开了操作系统和中间设备的限制。

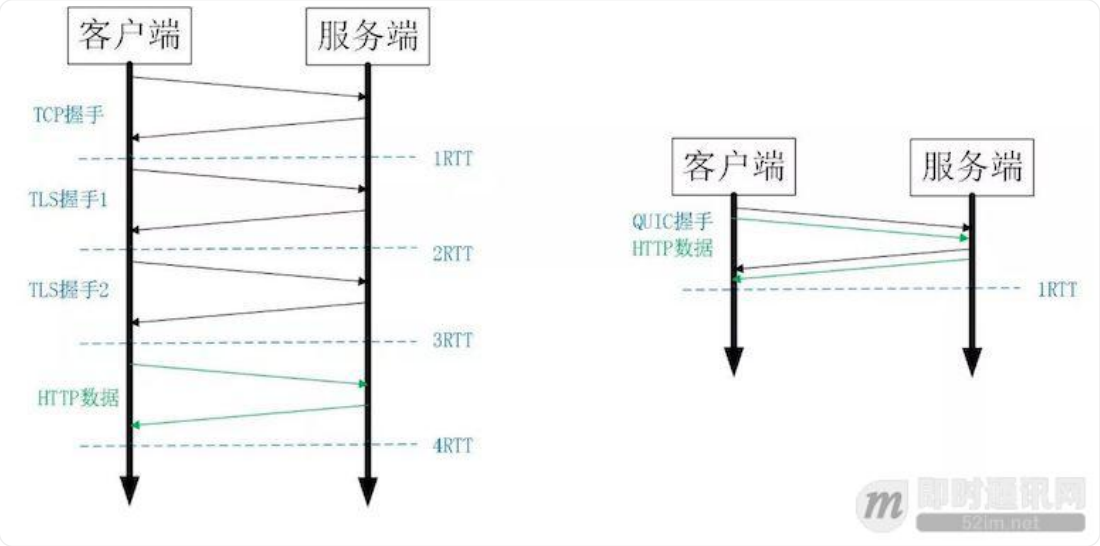
6、QUIC 核心特性

6.1 连接建立延时低

0RTT 建连可以说是 QUIC 相比 HTTP2 最大的性能优势。那什么是 0RTT 建连呢？

这里面有两层含义：

- 传输层 0RTT 就能建立连接；
- 加密层 0RTT 就能建立加密连接。



▲ 图 1 HTTPS 及 QUIC 建连过程

比如上图左边是 HTTPS 的一次完全握手的建连过程，需要 3 个 RTT。就算是 Session Resumption[14]，也需要至少 2 个 RTT。

而 QUIC 呢？由于建立在 UDP 的基础上，同时又实现了 0RTT 的安全握手，所以在大部分情况下，只需要 0 个 RTT 就能实现数据发送，在实现前向加密 [15] 的基础上，并且 0RTT 的成功率相比 TLS 的 Sesison Ticket[13] 要高很多。

6.2 改进的拥塞控制

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 [22]。

QUIC 协议当前默认使用了 TCP 协议的 Cubic 拥塞控制算法 [6]，同时也支持 CubicBytes, Reno, RenoBytes, BBR, PCC 等拥塞控制算法。

从拥塞算法本身来看，QUIC 只是按照 TCP 协议重新实现了一遍，那么 QUIC 协议到底改进在哪些方面呢？主要有如下几点。

【可插拔】：

什么叫可插拔呢？就是能够非常灵活地生效，变更和停止。体现在如下方面：

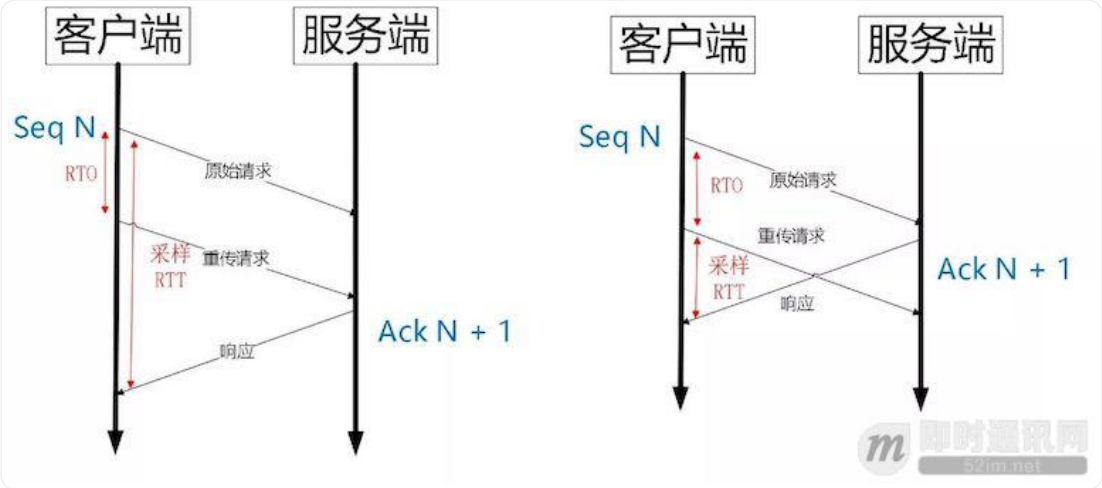
- 1）应用程序层面就能实现不同的拥塞控制算法，不需要操作系统，不需要内核支持。这是一个飞跃，因为传统的 TCP 拥塞控制，必须要端到端的网络协议栈支持，才能实现控制效果。而内核和操作系统的部署成本非常高，升级周期很长，这在产品快速迭代，网络爆炸式增长的今天，显然有点满足不了需求；
- 2）即使是单个应用程序的不同连接也能支持配置不同的拥塞控制。就算是一台服务器，接入的用户网络环境也千差万别，结合大数据及人工智能处理，我们能为各个用户提供不同的但又更加精准更加有效的拥塞控制。比如 BBR 适合，Cubic 适合；
- 3）应用程序不需要停机和升级就能实现拥塞控制的变更，我们在服务端只需要修改一下配置，reload 一下，完全不需要停止服务就能实现拥塞控制的切换。

STGW 在配置层面进行了优化，我们可以针对不同业务，不同网络制式，甚至不同的 RTT，使用不同的拥塞控制算法。

【单调递增的 Packet Number】：

TCP 为了保证可靠性，使用了基于字节序号的 Sequence Number 及 Ack 来确认消息的有序到达。

QUIC 同样是一个可靠的协议，它使用 Packet Number 代替了 TCP 的 sequence number，并且每个 Packet Number 都严格递增，也就是说就算 Packet N 丢失了，重传的 Packet N 的 Packet Number 已经不是 N，而是一个比 N 大的值。而 TCP 呢，重传 segment 的 sequence number 和原始的 segment 的 Sequence Number 保持不变，也正是由于这个特性，引入了 Tcp 重传的歧义问题。

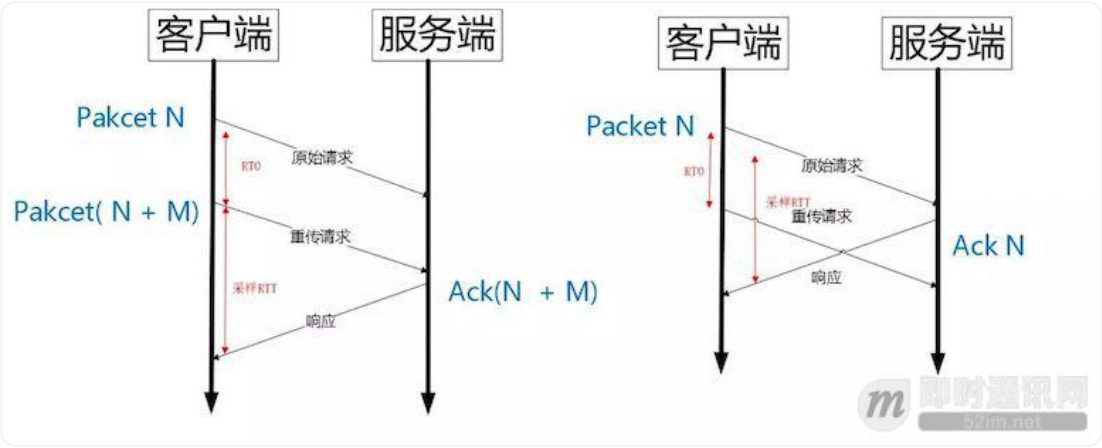


▲ 图 2 Tcp 重传歧义性

如上图所示，超时事件 RTO 发生后，客户端发起重传，然后接收到了 Ack 数据。由于序列号一样，这个 Ack 数据到底是原始请求的响应还是重传请求的响应呢？不好判断。

如果算成原始请求的响应，但实际上是重传请求的响应（上图左），会导致采样 RTT 变大。如果算成重传请求的响应，但实际上是原始请求的响应，又很容易导致采样 RTT 过小。

由于 Quic 重传的 Packet 和原始 Packet 的 Pakcet Number 是严格递增的，所以很容易就解决了这个问题。



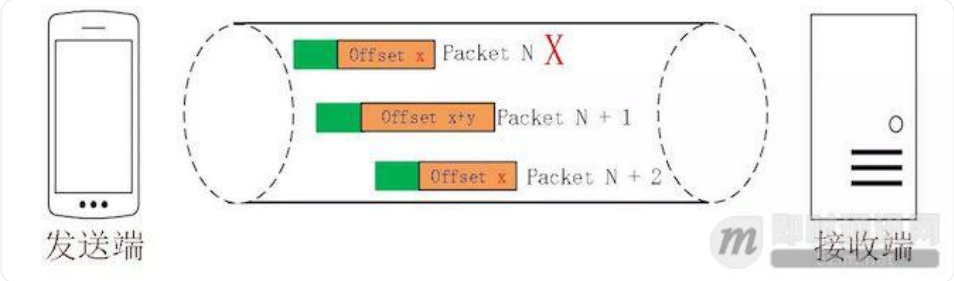
▲ 图 3 Quic 重传没有歧义性

如上图所示，RTO 发生后，根据重传的 Packet Number 就能确定精确的 RTT 计算。如果 Ack 的 Packet Number 是 N+M，就根据重传请求计算采样 RTT。如果 Ack 的 Pakcet Number 是 N，就根据原始请求的时间计算采样 RTT，没有歧义性。

但是单纯依靠严格递增的 Packet Number 肯定是无法保证数据的顺序性和可靠性。QUIC 又引入了一个 Stream Offset 的概念。

即一个 Stream 可以经过多个 Packet 传输，Packet Number 严格递增，没有依赖。但是 Packet 里的 Payload 如果是 Stream 的话，就需要依靠 Stream 的 Offset 来保证应用数据的顺序。如错误! 未找到引用源。所示，发送端先后发送了 Pakcet N 和 Pakcet N+1，Stream 的 Offset 分别是 x 和 x+y。

假设 Packet N 丢失了，发起重传，重传的 Packet Number 是 N+2，但是它的 Stream 的 Offset 依然是 x，这样就算 Packet N + 2 是后到的，依然可以将 Stream x 和 Stream x+y 按照顺序组织起来，交给应用程序处理。



▲ 图 4 Stream Offset 保证有序性

【不允许 Reneging】：

什么叫 Reneging 呢？就是接收方丢弃已经接收并且上报给 SACK 选项的内容 [8]。TCP 协议不鼓励这种行为，但是协议层面允许这样的行为。主要是考虑到服务器资源有限，比如 Buffer 溢出，内存不够等情况。

Reneging 对数据重传会产生很大的干扰。因为 Sack 都已经表明接收到了，但是接收端事实上丢弃了该数据。

QUIC 在协议层面禁止 Reneging，一个 Packet 只要被 Ack，就认为它一定被正确接收，减少了这种干扰。

【更多的 Ack 块】：

TCP 的 Sack 选项能够告诉发送方已经接收到的连续 Segment 的范围，方便发送方进行选择性重传。

由于 TCP 头部最大只有 60 个字节，标准头部占用了 20 字节，所以 Tcp Option 最大长度只有 40 字节，再加上 Tcp Timestamp option 占用了 10 个字节 [25]，所以留给 Sack 选项的只有 30 个字节。

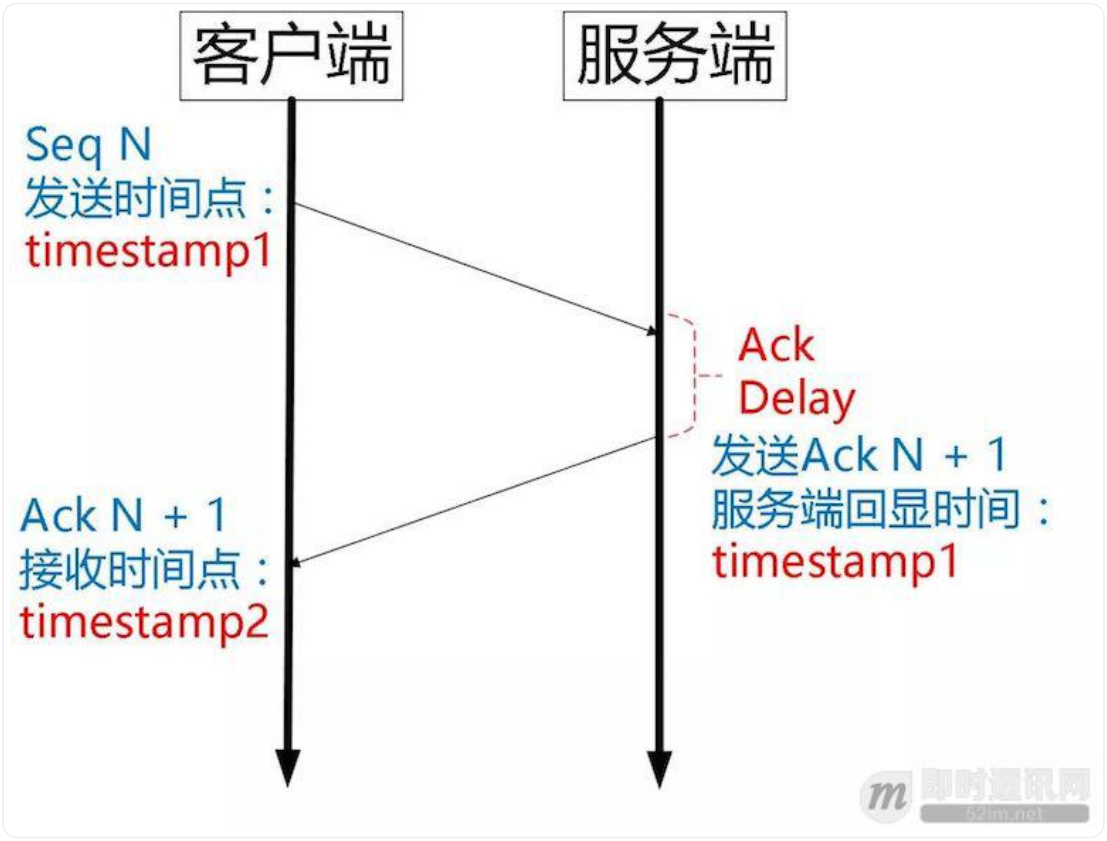
每一个 Sack Block 的长度是 8 个，加上 Sack Option 头部 2 个字节，也就意味着 Tcp Sack Option 最大只能提供 3 个 Block。

但是 Quic Ack Frame 可以同时提供 256 个 Ack Block，在丢包率比较高的网络下，更多的 Sack Block 可以提升网络的恢复速度，减少重传量。

【Ack Delay 时间】：

Tcp 的 Timestamp 选项存在一个问题 [25]，它只是回显了发送方的时间戳，但是没有计算接收端接收到 segment 到发送 Ack 该 segment 的时间。这个时间可以简称为 Ack Delay。

这样就会导致 RTT 计算误差。如下图：



可以认为 TCP 的 RTT 计算：

$$RTT = timestamp2 - timestamp1$$

而 Quic 计算如下：

$$RTT = timestamp2 - timestamp1 - Ack\ Delay$$

当然 RTT 的具体计算没有这么简单，需要采样，参考历史数值进行平滑计算，参考如下公式 [9]：

$$SRTT = SRTT + \alpha (RTT - SRTT)$$
$$RTO = \mu * SRTT + \partial * DevRTT$$

6.3 基于 stream 和 connecton 级别的流量控制

QUIC 的流量控制 [22] 类似 HTTP2，即在 Connection 和 Stream 级别提供了两种流量控制。为什么需要两类流量控制呢？主要是因为 QUIC 支持多路复用。

Stream 可以认为就是一条 HTTP 请求。

Connection 可以类比一条 TCP 连接。多路复用意味着在一条 Connetion 上会同时存在多条 Stream。既需要对单个 Stream 进行控制，又需要针对所有 Stream 进行总体控制。

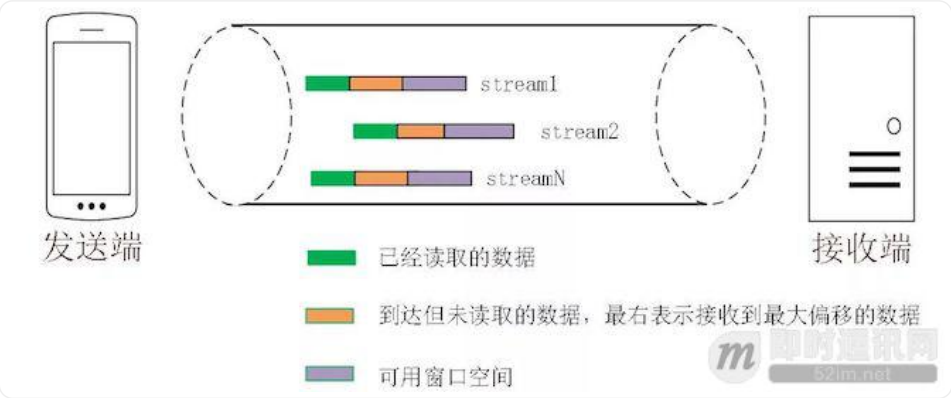
QUIC 实现流量控制的原理比较简单：

通过 window_update 帧告诉对端自己可以接收的字节数，这样发送方就不会发送超过这个数量的数据。

通过 BlockFrame 告诉对端由于流量控制被阻塞了，无法发送数据。

QUIC 的流量控制和 TCP 有点区别，TCP 为了保证可靠性，窗口左边沿向右滑动时的长度取决于已经确认的字节数。如果中间出现丢包，就算接收到了更大序号的 Segment，窗口也无法超过这个序列号。

但 QUIC 不同，就算此前有些 packet 没有接收到，它的滑动只取决于接收到的最大偏移字节数。



▲ 图 5 Quic Flow Control

针对 Stream：

$$可用窗口 = 最大窗口数 - 接收到的最大偏移数$$

针对 Connection：

$$可用窗口 = stream1\ 可用窗口 + Stream2\ 可用窗口 + Stream\ N\ 可用窗口$$

同样地，STGW 也在连接和 Stream 级别设置了不同的窗口数。

最重要的是，我们可以在内存不足或者上游处理性能出现问题时，通过流量控制来限制传输速率，保障服务可用性。

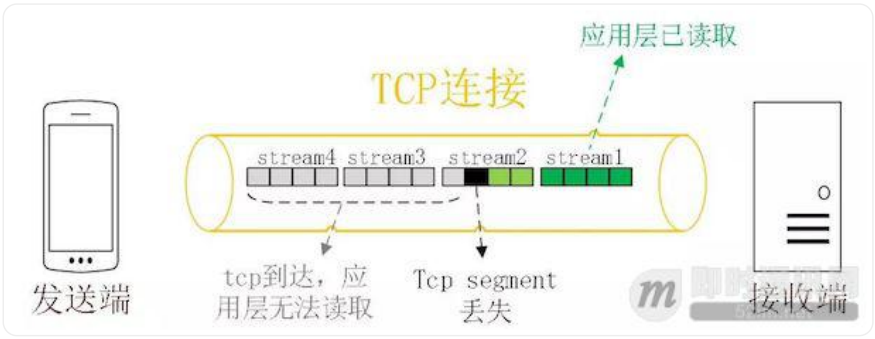
6.4 没有队头阻塞的多路复用

QUIC 的多路复用和 HTTP2 类似。在一条 QUIC 连接上可以并发发送多个 HTTP 请求 (stream)。但是 QUIC 的多路复用相比 HTTP2 有一个很大的优势。

QUIC 一个连接上的多个 stream 之间没有依赖。这样假如 stream2 丢了一个 udp packet，也只会影响 stream2 的处理。不会影响 stream2 之前及之后的 stream 的处理。

这也就在很大程度上缓解甚至消除了队头阻塞的影响。

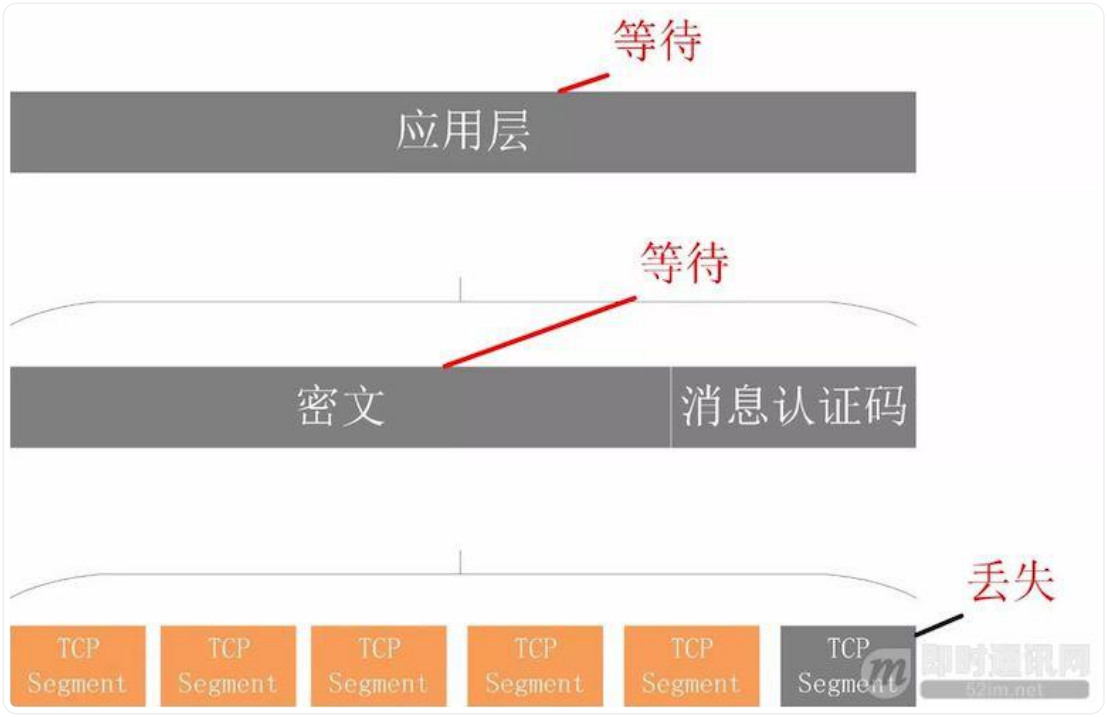
多路复用是 HTTP2 最强大的特性 [7]，能够将多条请求在一条 TCP 连接上同时发出去。但也恶化了 TCP 的一个问题，队头阻塞 [11]，如下图示：



▲ 图 6 HTTP2 队头阻塞

HTTP2 在一个 TCP 连接上同时发送 4 个 Stream。其中 Stream1 已经正确到达，并被应用层读取。但是 Stream2 的第三个 tcp segment 丢失了，TCP 为了保证数据的可靠性，需要发送端重传第 3 个 segment 才能通知应用层读取接下去的数据，虽然这个时候 Stream3 和 Stream4 的全部数据已经到达了接收端，但都被阻塞住了。

不仅如此，由于 HTTP2 强制使用 TLS，还存在一个 TLS 协议层面的队头阻塞 [12]。

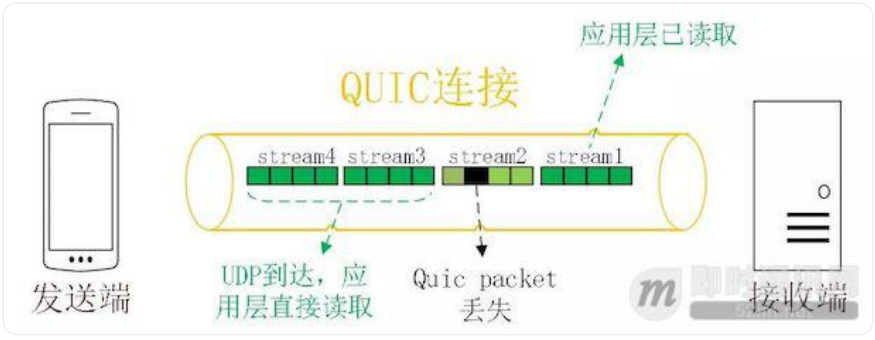


▲ 图 7 TLS 队头阻塞

Record 是 TLS 协议处理的最小单位，最大不能超过 16K，一些服务器比如 Nginx 默认的大小就是 16K。由于一个 record 必须经过数据一致性校验才能进行加解密，所以一个 16K 的 record，就算丢了一个字节，也会导致已经接收到的 15.99K 数据无法处理，因为它不完整。

那 QUIC 多路复用为什么能避免上述问题呢？

- 1）QUIC 最基本的传输单元是 Packet，不会超过 MTU 的大小，整个加密和认证过程都是基于 Packet 的，不会跨越多个 Packet。这样就能避免 TLS 协议存在的队头阻塞；
- 2）Stream 之间相互独立，比如 Stream2 丢了一个 Pakcet，不会影响 Stream3 和 Stream4。不存在 TCP 队头阻塞。



▲ 图 8 QUIC 多路复用时没有队头阻塞的问题

当然，并不是所有的 QUIC 数据都不会受到队头阻塞的影响，比如 QUIC 当前也是使用 Hpack 压缩算法 [10]，由于算法的限制，丢失一个头部数据时，可能遇到队头阻塞。

总体来说，QUIC 在传输大量数据时，比如视频，受到队头阻塞的影响很小。

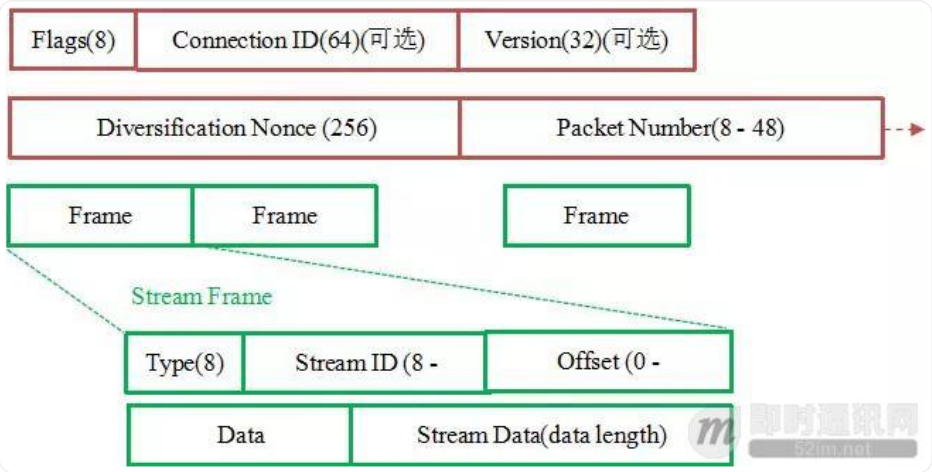
6.5 加密认证的报文

TCP 协议头部没有经过任何加密和认证，所以在传输过程中很容易被中间网络设备篡改，注入和窃听。比如修改序列号、滑动窗口。这些行为有可能是出于性能优化，也有可能是主动攻击。

但是 QUIC 的 packet 可以说是武装到了牙齿。除了个别报文比如 PUBLIC_RESET 和 CHLO，所有报文头部都是经过认证的，报文 Body 都是经过加密的。

这样只要对 QUIC 报文任何修改，接收端都能够及时发现，有效地降低了安全风险。

如下图所示，红色部分是 Stream Frame 的报文头部，有认证。绿色部分是报文内容，全部经过加密。



6.6 连接迁移

一条 TCP 连接 [17] 是由四元组标识的（源 IP，源端口，目的 IP，目的端口）。什么叫连接迁移呢？就是当其中任何一个元素发生变化时，这条连接依然维持着，能够保持业务逻辑不中断。当然这里面主要关注的是客户端的变化，因为客户端不可控并且网络环境经常发生变化，而服务端的 IP 和端口一般都是固定的。

比如大家使用手机在 WIFI 和 4G 移动网络切换时，客户端的 IP 肯定会发生变化，需要重新建立和服务端的 TCP 连接。

又比如大家使用公共 NAT 出口时，有些连接竞争时需要重新绑定端口，导致客户端的端口发生变化，同样需要重新建立 TCP 连接。

针对 TCP 的连接变化，MPTCP[5] 其实已经有了解决方案，但是由于 MPTCP 需要操作系统及网络协议栈支持，部署阻力非常大，目前并不适用。

所以从 TCP 连接的角度来讲，这个问题是无解的。

那 QUIC 是如何做到连接迁移呢？很简单，任何一条 QUIC 连接不再以 IP 及端口四元组标识，而是以一个 64 位的随机数作为 ID 来标识，这样就算 IP 或者端口发生变化时，只要 ID 不变，这条连接依然维持着，上层业务逻辑感知不到变化，不会中断，也就不需要重连。

由于这个 ID 是客户端随机产生的，并且长度有 64 位，所以冲突概率非常低。

6.7 其他亮点

此外，QUIC 还能实现前向冗余纠错，在重要的包比如握手消息发生丢失时，能够根据冗余信息还原出握手消息。

QUIC 还能实现证书压缩，减少证书传输量，针对包头进行验证等。

限于篇幅，本文不再详细介绍，有兴趣的可以参考文档 [23] 和文档 [4] 和文档 [26]。

7、参考文献

- [1]. https://www.chromium.org/quic

- [2]. https://docs.google.com/document/d/1gY9-YNDNAB1eip-RTPbqphgySwSNSDHLq9D5Bty4FSU/edit
- [3]. E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, draft-ietf-tls-tls13-21, https://tools.ietf.org/html/draft-ietf-tls-tls13-21, July 03, 2017
- [4]. Adam Langley,Wan-Teh Chang, “QUIC Crypto”，https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit, 20161206
- [5]. https://www.multipath-tcp.org/
- [6]. Ha, S., Rhee, I., and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review , 2008.
- [7]. M. Belshe,BitGo, R. Peon, “Hypertext Transfer Protocol Version 2 (HTTP/2)”, RFC 7540, May 2015
- [8]. M. Mathis,J. Mahdavi,S. Floyd,A. Romanow,“TCP Selective Acknowledgment Options”, rfc2018, https://tools.ietf.org/html/rfc2018, October 1996
- [9]. V. Paxson，M. Allman，J. Chu，M. Sargent，“Computing TCP's Retransmission Timer”， rfc6298, https://tools.ietf.org/html/rfc6298, June 2011
- [10]. R. Peon,H. Ruellan,“HPACK: Header Compression for HTTP/2”,RFC7541,May 2015
- [11]. M. Scharf, Alcatel-Lucent Bell Labs, S. Kiesel, “Quantifying Head-of-Line Blocking in TCP and SCTP”, https://tools.ietf.org/id/draft-scharf-tcpm-reordering-00.html, July 15, 2013
- [12]. Ilya Grigorik，“Optimizing TLS Record Size & Buffering Latency”, https://www.igvita.com/2013/10/24/optimizing-tls-record-size-and-buffering-latency/, October 24, 2013
- [13]. J. Salowey,H. Zhou,P. Eronen,H. Tschofenig, “Transport Layer Security (TLS) Session Resumption without Server-Side State”, RFC5077, January 2008
- [14]. Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.
- [15]. Shirey, R., "Internet Security Glossary, Version 2", FYI , RFC 4949, August 2007
- [16]. 罗成，“HTTPS性能优化”， http://www.infoq.com/cn/presentations/performance-optimization-of-https，February.2017
- [17]. Postel, J., "Transmission Control Protocol", STD 7, RFC793, September 1981.
- [18]. J. Postel,“User Datagram Protocol”, RFC768,August 1980
- [19]. Q. Dang, S. Santesson,K. Moriarty,D. Brown.T. Polk, “Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA”,RFC5758, January 2010
- [20]. Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002
- [21]. D.Cooper,S.Santesson, S.Farrell,S. Boeyen,R. Housley,W.Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, RFC5280, May 2008
- [22]. M. Allman,V. Paxson,E. Blanton, "TCP Congestion Control",RFC5681, September 2009
- [23]. Robbie Shade, “Flow control in QUIC”, https://docs.google.com/document/d/1F2YfdDXKpy20WVKJueEf4abn_LVZHhMUMS5gX6Pgjl4/edit#, May, 2016,
- [24]. ianswett , “QUIC fec v1”, https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit#heading=h.xgjl2srtytjt, 2016-02-19
- [25]. D.Borman,B.Braden,V.Jacobson,R.Scheffenegger, Ed. “TCP Extensions for High Performance”,rfc7323, https://tools.ietf.org/html/rfc7323,September 2014
- [26]. 罗成，“WEB加速，协议先行”， https://zhuanlan.zhihu.com/p/27938635，july, 2017

（原文链接：点此进入）

附录：相关技术资料

全站即时通讯技术资料分类

[1] 网络编程基础资料：
《TCP/IP详解 - 第11章·UDP：用户数据报协议》
《TCP/IP详解 - 第17章·TCP：传输控制协议》
《TCP/IP详解 - 第18章·TCP连接的建立与终止》
《TCP/IP详解 - 第21章·TCP的超时与重传》
《技术往事：改变世界的TCP/IP协议（珍贵多图、手机慎点）》
《通俗易懂-深入理解TCP协议（上）：理论基础》
《通俗易懂-深入理解TCP协议（下）：RTT、滑动窗口、拥塞处理》
《理论经典：TCP协议的3次握手与4次挥手过程详解》
《理论联系实际：Wireshark抓包分析TCP 3次握手、4次挥手过程》
《计算机网络通讯协议关系图（中文珍藏版）》
《UDP中一个包的大小最大能多大？》
《P2P技术详解(一)：NAT详解——详细原理、P2P简介》
《P2P技术详解(二)：P2P中的NAT穿越(打洞)方案详解》
《P2P技术详解(三)：P2P技术之STUN、TURN、ICE详解》

《通俗易懂：快速理解P2P技术中的NAT穿透原理》
《高性能网络编程(一)：单台服务器并发TCP连接数到底可以有多少》
《高性能网络编程(二)：上一个10年，著名的C10K并发连接问题》
《高性能网络编程(三)：下一个10年，是时候考虑C10M并发问题了》
《高性能网络编程(四)：从C10K到C10M高性能网络应用的理论探索》
《不为人知的网络编程(一)：浅析TCP协议中的疑难杂症(上篇)》
《不为人知的网络编程(二)：浅析TCP协议中的疑难杂症(下篇)》
《不为人知的网络编程(三)：关闭TCP连接时为什么会TIME_WAIT、CLOSE_WAIT》
《不为人知的网络编程(四)：深入研究分析TCP的异常关闭》
《不为人知的网络编程(五)：UDP的连接性和负载均衡》
《不为人知的网络编程(六)：深入地理解UDP协议并用好它》
《不为人知的网络编程(七)：如何让不可靠的UDP变的可靠？》
《网络编程懒人入门(一)：快速理解网络通信协议（上篇）》
《网络编程懒人入门(二)：快速理解网络通信协议（下篇）》
《网络编程懒人入门(三)：快速理解TCP协议一篇就够》
《网络编程懒人入门(四)：快速理解TCP和UDP的差异》
《网络编程懒人入门(五)：快速理解为什么说UDP有时比TCP更有优势》
《技术扫盲：新一代基于UDP的低延时网络传输层协议——QUIC详解》
>> 更多同类文章 ……

[2] NIO异步网络编程资料：
《Java新一代网络编程模型AIO原理及Linux系统AIO介绍》
《有关“为何选择Netty”的11个疑问及解答》
《开源NIO框架八卦——到底是先有MINA还是先有Netty?》
《选Netty还是Mina：深入研究与对比（一）》
《选Netty还是Mina：深入研究与对比（二）》
《NIO框架入门(一)：服务端基于Netty4的UDP双向通信Demo演示》
《NIO框架入门(二)：服务端基于MINA2的UDP双向通信Demo演示》
《NIO框架入门(三)：iOS与MINA2、Netty4的跨平台UDP双向通信实战》
《NIO框架入门(四)：Android与MINA2、Netty4的跨平台UDP双向通信实战》
《Netty 4.x学习（一）：ByteBuf详解》
《Netty 4.x学习（二）：Channel和Pipeline详解》
《Netty 4.x学习（三）：线程模型详解》
《Apache Mina框架高级篇（一）：IoFilter详解》
《Apache Mina框架高级篇（二）：IoHandler详解》
《MINA2 线程原理总结（含简单测试实例）》
《Apache MINA2.0 开发指南（中文版）[附件下载]》
《MINA、Netty的源代码（在线阅读版）已整理发布》
《解决MINA数据传输中TCP的粘包、缺包问题（有源码）》
《解决Mina中多个同类型Filter实例共存的问题》
《实践总结：Netty3.x升级Netty4.x遇到的那些坑（线程篇）》
《实践总结：Netty3.x VS Netty4.x的线程模型》
《详解Netty的安全性：原理介绍、代码演示（上篇）》
《详解Netty的安全性：原理介绍、代码演示（下篇）》
《详解Netty的优雅退出机制和原理》
《NIO框架详解：Netty的高性能之道》
《Twitter：如何使用Netty 4来减少JVM的GC开销（译文）》
《绝对干货：基于Netty实现海量接入的推送服务技术要点》
《Netty干货分享：京东京麦的生产级TCP网关技术实践总结》
>> 更多同类文章 ……

[3] 有关IM/推送的通信格式、协议的选择：
《简述传输层协议TCP和UDP的区别》
《为什么QQ用的是UDP协议而不是TCP协议？》
《移动端即时通讯协议选择：UDP还是TCP？》
《如何选择即时通讯应用的数据传输格式》
《强列建议将Protobuf作为你的即时通讯应用数据传输格式》
《全方位评测：Protobuf性能到底有没有比JSON快5倍？》
《移动端IM开发需要面对的技术问题（含通信协议选择）》
《简述移动端IM开发的那些坑：架构设计、通信协议和客户端》
《理论联系实际：一套典型的IM通信协议设计详解》
《58到家实时消息系统的协议设计等技术实践分享》
《详解如何在NodeJS中使用Google的Protobuf》
《技术扫盲：新一代基于UDP的低延时网络传输层协议——QUIC详解》
>> 更多同类文章 ……

来源：即时通讯网 - 即时通讯开发者社区！

标签： 网络编程 QUIC

本主题由 JackJiang 于 7 个月前 加入精华

上一篇：不为人知的网络编程(七)：如何让不可靠的UDP变的可靠？ · 下一篇： 新人网络编程技术咨询贴

本帖已收录至以下技术专辑

网络编程基础 | 主题 67 · 关注 14　　IM/推送的通信格式、协议篇 | 主题 21 · 关注 10

相关文章

- 网络编程懒人入门(六)：史上最通俗的集线器、交换机、路由器功能原理入门
- 网络编程懒人入门(七)：深入浅出，全面理解HTTP协议
- 微信小程序中如何使用WebSocket实现长连接(含完整源码)
- 从HTTP/0.9到HTTP/2：一文读懂HTTP协议的历史演变和设计思路
- 网络编程懒人入门(八)：手把手教你写基于TCP的Socket长连接
- 脑残式网络编程入门(一)：跟着动画来学TCP三次握手和四次挥手
- 脑残式网络编程入门(二)：我们在读写Socket时，究竟在读写什么？
- 跪求解答！不太理解这段Python解码WebSocket报文
- 脑残式网络编程入门(三)：HTTP协议必知必会的一些知识
- 脑残式网络编程入门(四)：快速理解HTTP/2的服务器推送(Server Push)

推荐方案

评论 7

2 楼: 我就叫这个 *Lv.1*　7 个月前　引用此评论
感谢分享

3 楼: rzc666 *Lv.4*　5 个月前　引用此评论
写的不错, 赞!

4 楼: x931609201 *Lv.2*　5 个月前　引用此评论
这个协议实现只要hook操作系统的tcp相关的api就可以了吧

5 楼: JackJiang *Lv.9* 楼主　5 个月前　引用此评论

> **引用：** x931609201 发表于 2018-02-28 18:52
> 这个协议实现只要hook操作系统的tcp相关的api就可以了吧

QUIC是google基于UDP开发的新协议，并不是操作系统这一层，因为主流操作系统并不受google控制，而且暂时还不属于通用的标准协议

签名: 《最火移动端跨平台方案盘点：React Native、weex、Flutter》http://www.52im.net/thread-1870-1-1.html

6 楼: malt93 *Lv.1*　17 天前　引用此评论
非常好，看了论文回来看这篇文章有新的启发和理解。
不过对于标题有一些质疑，严格来说QUIC本身应该算是应用层协议，网络传输层是基于UDP，但是在应用层实现了类似TCP的功能

7 楼: JackJiang *Lv.9* 楼主　17 天前　引用此评论

> **引用：** malt93 发表于 2018-07-28 16:52
> 非常好，看了论文回来看这篇文章有新的启发和理解。
> 不过对于标题有一些质疑，严格来说QUIC本身应该算是应 ...

你觉得标题该怎么写？

签名: 《最火移动端跨平台方案盘点：React Native、weex、Flutter》http://www.52im.net/thread-1870-1-1.html

8 楼: malt93 *Lv.1*　15 天前　引用此评论

> **引用：** JackJiang 发表于 2018-07-28 17:43
> 你觉得标题该怎么写？

最简单的改法是把“层”去掉

✚ 发新帖 发表评论 返回列表

即时通讯网

实时推送、IM等即时通讯相关技术的学习、交流与分享的平台。专业的资料、专业的人、专业的社区！让即时通讯技术能更好传播与分享。

平等 开放 分享 传承

商务/合作：business@52im.net
投稿/报道：contact@52im.net

友情链接 [友链交换]

WebRTC中文网
JackJiang的Git
一起开源网
OpenSNS
容联云通讯
网易云信

关于

关于我们
活跃QQ群
在线文档
网址导航
广告投放 new

手机访问本站

微信公众号 new