**简介：** XQUIC是阿里巴巴淘系架构团队自研的IETF QUIC标准化协议库实现，在手机淘宝上进行了广泛的应用，并在多个不同类型的业务场景下取得明显的效果提升，为手机淘宝APP的用户带来丝般顺滑的网络体验： 在RPC请求场景，网络耗时降低15% 在直播高峰期场景，卡顿率降低30%、秒开率提升2% 在短视频场景，卡顿率降低20%

XQUIC是阿里巴巴淘系架构团队自研的IETF QUIC标准化协议库实现，在手机淘宝上进行了广泛的应用，并在多个不同类型的业务场景下取得明显的效果提升，为手机淘宝APP的用户带来丝般顺滑的网络体验：

* 在RPC请求场景，网络耗时降低15%
* 在直播高峰期场景，卡顿率降低30%、秒开率提升2%
* 在短视频场景，卡顿率降低20%

从以上提升效果可以看出，对QUIC的一个常见认知谬误：“QUIC只对弱网场景有优化提升”是不准确的。实际上QUIC对于整体网络体验有普遍提升，弱网场景由于基线较低、提升空间更显著。此外，在5G推广初期，基站部署不够密集的情况下，如何保证稳定有效带宽速率，是未来2-3年内手机视频应用将面临的重大挑战，而我们研发的MPQUIC将为这些挑战提供有效的解决方案。

本文将会重点介绍XQUIC的设计原理，面向业务场景的网络传输优化，以及面向5G的Multipath QUIC技术（多路径QUIC）。

# **QUIC**

▐ 网络分层模型及QUIC进化史

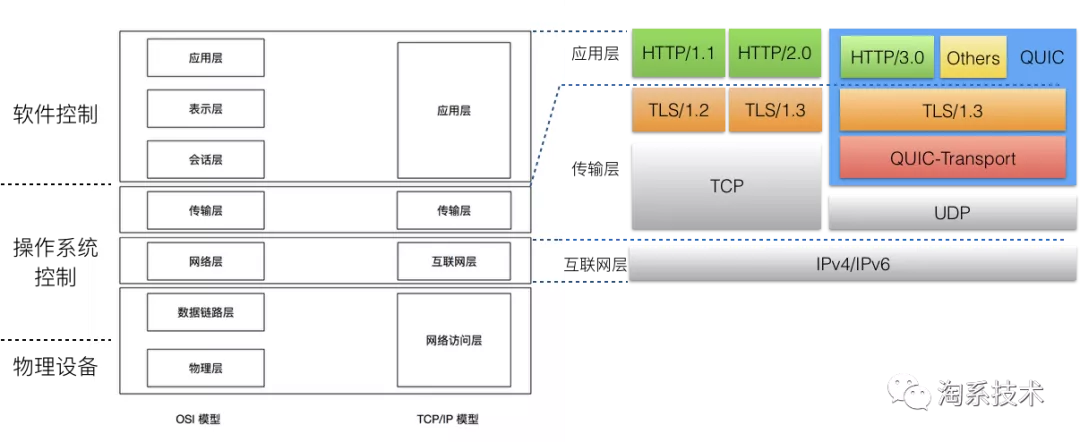


图1. 网络七层/四层模型 和 QUIC分层设计

为了方便说明QUIC在网络通信协议栈中所处的位置及职能，我们简单回顾一下网络OSI模型（七层模型）和TCP/IP模型（四层模型）。从两套网络模型中可以看出，网络传输行为和策略主要由传输层来控制，而TCP作为过去30年最为流行和广泛使用的传输层协议，是由操作系统控制和实现的。

QUIC是由Google从2013年开始研究的基于UDP的可靠传输协议，它最早的原型是SPDY + QUIC-Crypto + Reliable UDP，后来经历了SPDY[1]转型为2015年5月IETF正式发布的HTTP/2.0[2]，以及2016年TLS/1.3[3]的正式发布。QUIC在IETF的标准化工作组自2016年成立，考虑到HTTP/2.0和TLS/1.3的发布，它的核心协议族逐步进化为现在的HTTP/3.0 + TLS/1.3 + QUIC-Transport的组合。

▐ QUIC带来的核心收益是什么

众所周知，QUIC具备多路复用/0-RTT握手/连接迁移等多种优点，然而在这些优势中，最关键的核心收益，当属QUIC将四/七层网络模型中控制传输行为的传输层，从内核态实现迁移到了用户态实现，由应用软件控制。这将带来2个巨大的优势：

****(1) 迭代优化效率大大提升。****以服务端角度而言，大型在线系统的内核升级成本往往是非常高的，考虑到稳定性等因素，升级周期从月到年为单位不等。以客户端角度而言，手机操作系统版本升级同样由厂商控制，升级周期同样难以把控。调整为用户态实现后，端到端的升级都非常方便，版本迭代周期以周为计（甚至更快）。

****(2) 灵活适应不同业务场景的网络需求。****在过去4G的飞速发展中，短视频、直播等新的业务场景随着基建提供的下行带宽增长开始出现，在流媒体传输对于稳定高带宽和低延迟的诉求下，TCP纷纷被各类标准/私有UDP解决方案逐步替代，难以争得一席之地。背后的原因是，实现在内核态的TCP，难以用一套拥塞控制算法/参数适应快速发展的各类业务场景。这一缺陷将在5G下变得更加显著。QUIC则可将拥塞控制算法/参数调控到连接的粒度，针对同一个APP内的不同业务场景（例如RPC/短视频/直播等）具备灵活适配/升级的能力。

在众多增强型UDP的选择中，QUIC相较于其他的方案最为通用，不仅具备对于HTTP系列的良好兼容性，同时其优秀的的分层设计，也使得它可以将传输层单独剥离作为TCP的替代方案，为其他应用层协议提供可靠/非可靠传输能力（是的，QUIC也有非可靠传输草案设计）。

# **XQUIC是什么、为什么选择自研 + 标准**

XQUIC是阿里自研的IETF QUIC标准化实现，这个项目由淘系架构网关与基础网络团队发起和主导，当前有阿里云CDN、达摩院XG实验室与AIS网络研究团队等多个团队参与其中。

现今QUIC有多家开源实现，为什么选择标准协议 + 自研实现的道路？我们从14年开始关注Google在QUIC上的实践（手机淘宝在16年全面应用HTTP/2），从17年底开始跟进并尝试在电商场景落地GQUIC[4]，在18年底在手淘图片、短视频等场景落地GQUIC并拿到了一定的网络体验收益。然而在使用开源方案的过程中或多或少碰到了一些问题，Google的实现是所有开源实现中最为成熟优秀的，然而由于Chromium复杂的运行环境和C++实现的缘故，GQUIC包大小在优化后仍然有2.4M左右，这使得我们在集成手淘时面临困难。在不影响互通性的前提下，我们进行了大量裁剪才勉强能够达到手淘集成的包大小要求，然而在版本升级的情况下难以持续迭代。其他的开源实现也有类似或其他的问题（例如依赖过多、无服务端实现、无稳定性保障等）。最终促使我们走上自研实现的道路。

为什么要选择IETF QUIC[5]标准化草案的协议版本？过去我们也尝试过自研私有协议，在端到端都由内部控制的场景下，私有协议的确是很方便的，但私有协议方案很难走出去建立一个生态圈 / 或者与其他的应用生态圈结合（遵循相同的标准化协议实现互联互通）；从阿里作为云厂商的角度，私有协议也很难与外部客户打通；同时由于IETF开放讨论的工作模式，协议在安全性、扩展性上会有更全面充分的考量。

因此我们选择IETF QUIC标准化草案版本来落地。截止目前，IETF工作组草案已经演化到draft-29版本（2020.6.10发布），XQUIC已经支持该版本，并能够与其他开源实现基于draft-29互通。

# **XQUIC整体架构和传输框架设计**

XQUIC是IETF QUIC草案版本的一个C协议库实现，端到端的整体链路架构设计如下图所示。XQUIC内部包含了QUIC-Transport（传输层）、QUIC-TLS（加密层、与TLS/1.3对接）和HTTP/3.0（应用层）的实现。在外部依赖方面，TLS/1.3依赖了开源boringssl或openssl实现（两者XQUIC都做了支持、可用编译选项控制），除此之外无其他外部依赖。

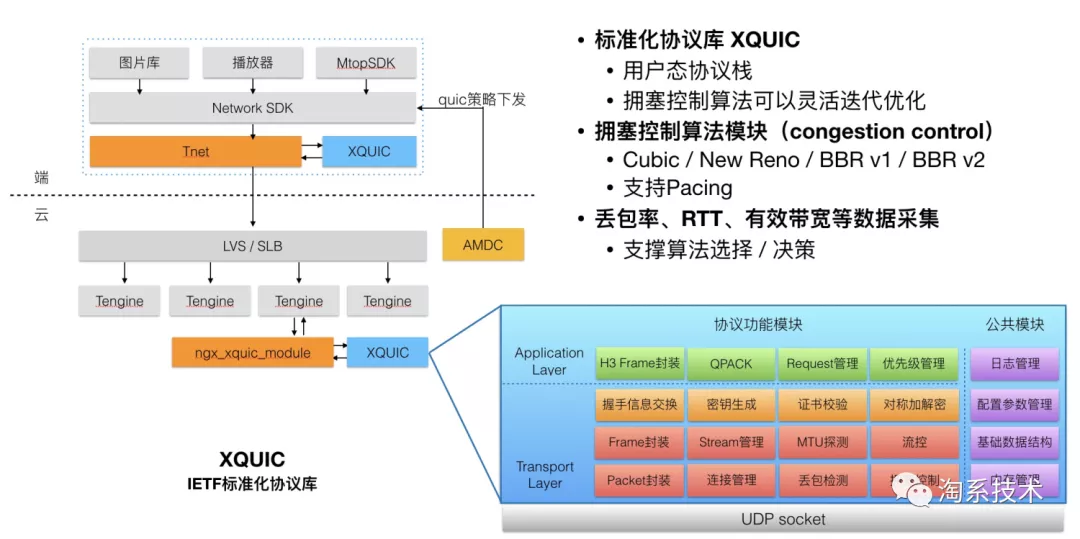


图2. XQUIC端到端架构设计 和 内部分层模块

XQUIC整体包大小在900KB左右（包含boringssl的情况下），对于客户端集成是较为轻量的（支持Android/iOS）。服务端方面，由于阿里内部网关体系广泛使用Tengine（Nginx开源分支），我们开发了一个ngx\_xquic\_module用于适配Tengine服务端。协议的调度方面，由客户端网络库与调度服务AMDC配合完成，可以根据版本/地域/运营商/设备百分比进行协议调度。

XQUIC传输层内部流程设计如下图，可以看到XQUIC内部的读写事件主流程。考虑到跨平台兼容性，UDP收发接口由外部实现并注册回调接口。XQUIC内部维护了每条连接的状态机、Stream状态机，在Stream级别实现可靠传输（这也是根本上解决TCP头部阻塞的关键），并通过读事件通知的方式将数据投递给应用层。传输层Stream与应用层HTTP/3的Request Stream有一一映射关系，通过这样的方式解决HTTP/2 over TCP的头部阻塞问题。

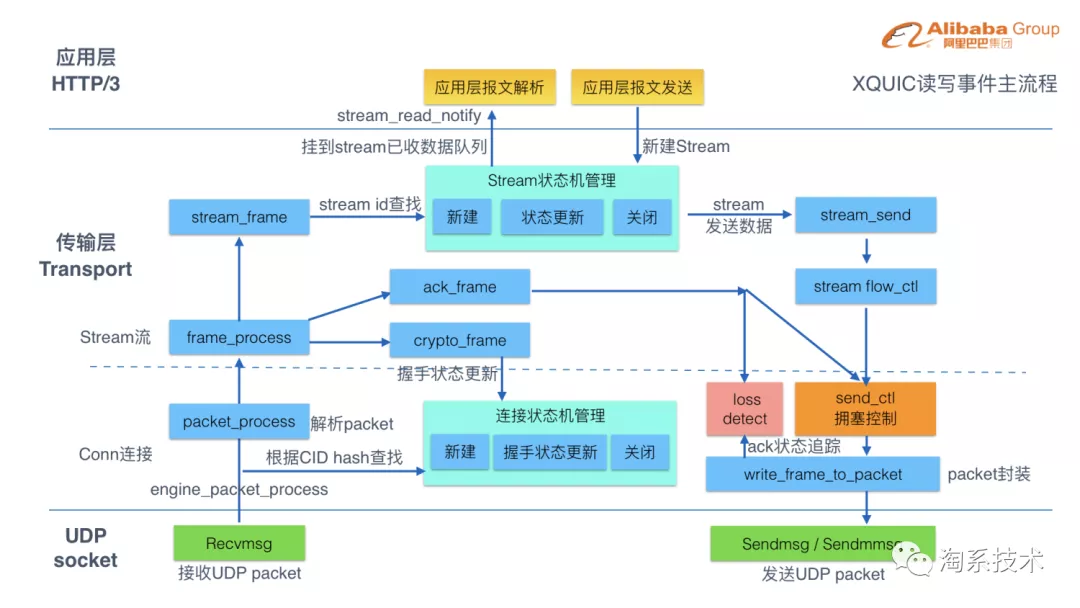


图3. XQUIC读写事件主流程设计

考虑到IETF QUIC传输层的设计可以独立剥离，并作为TCP的替代方案对接其他应用层协议，XQUIC内部实现同样基于这样的分层方式，并对外提供两套原生接口：HTTP/3请求应答接口 和 传输层独立接口（类似TCP），使得例如RTMP、上传协议等可以较为轻松地接入。

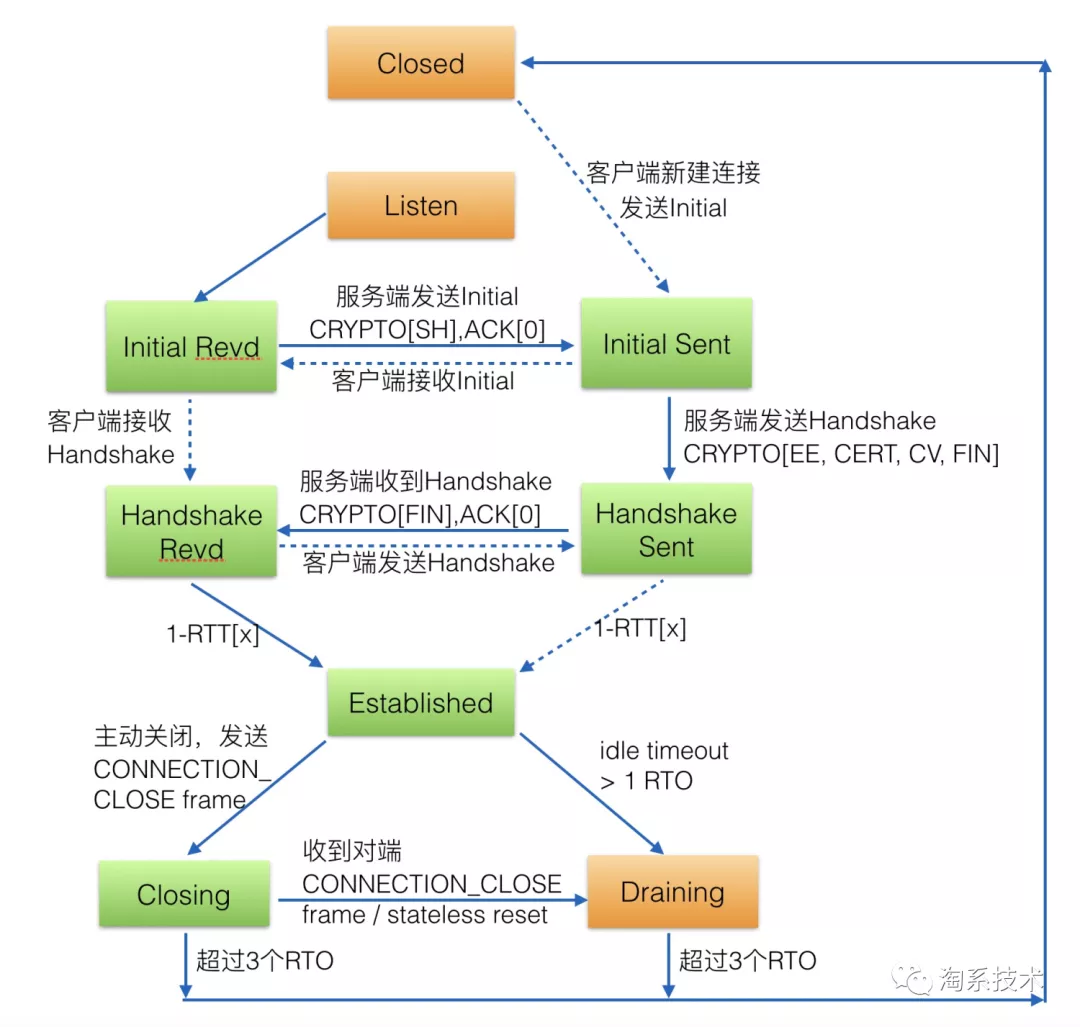


图4. XQUIC内部的连接状态机设计（参考TCP）

# **XQUIC拥塞控制算法模块**

我们将XQUIC传输层的内部设计放大，其中拥塞控制算法模块，是决定传输行为和效率的核心模块之一。

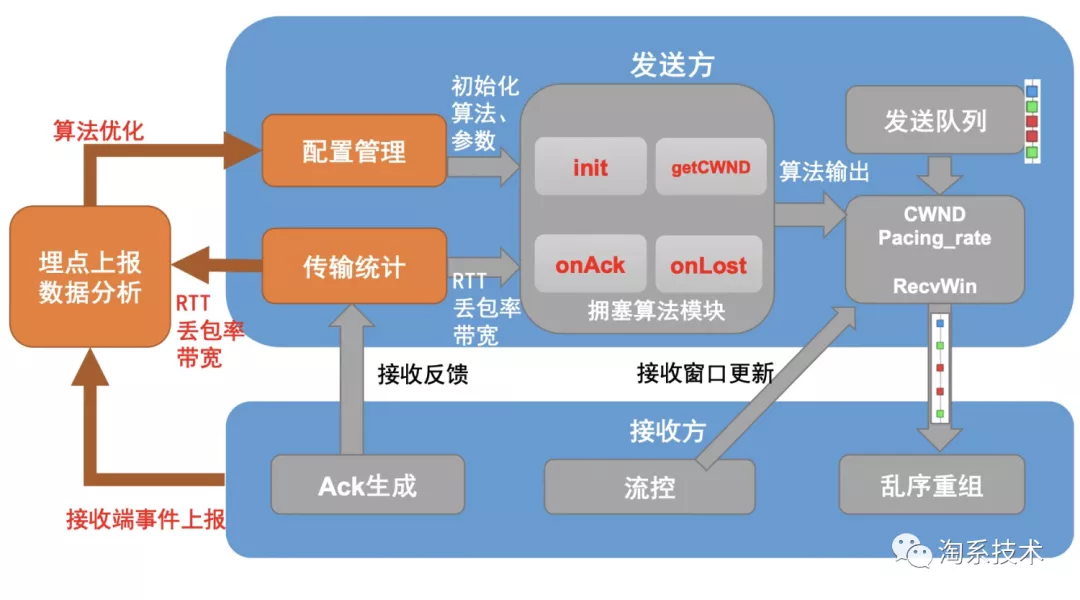


图5. XQUIC拥塞控制算法模块设计

为了能够方便地实现多套拥塞控制算法，我们将拥塞控制算法流程抽象成7个回调接口，其中最核心的两个接口onAck和onLost用于让算法实现收到报文ack和检测到丢包时的处理逻辑。XQUIC内部实现了多套拥塞控制算法，包括最常见的Cubic、New Reno，以及音视频场景下比较流行的BBR v1和v2，每种算法都只需要实现这7个回调接口即可实现完整算法逻辑。

为了方便用数据驱动网络体验优化，我们将连接的丢包率、RTT、带宽等信息通过埋点数据采样和分析的方式，结合每个版本的算法调整进行效果分析。同时在实验环境下模拟真实用户的网络环境分布，更好地预先评估算法调整对于网络体验的改进效果。

# **面向业务场景的传输优化**

XQUIC在RPC请求场景降低网络耗时15%，在短视频场景下降低20%卡顿率，在直播场景高峰期降低30%卡顿率、提升2%秒开率（相对于TCP）。以下基于当下非常火热的直播场景，介绍XQUIC如何面向业务场景优化网络体验。

▐ 优化背景

部分用户网络环境比较差，存在直播拉流打开慢、卡顿问题。

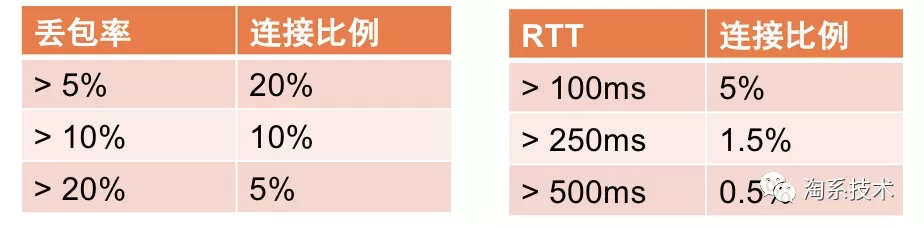


图6. 某节点丢包率和RTT统计分布

这是CDN某节点上统计的丢包率和RTT分布数据，可以看到，有5%的连接丢包率超过20%，0.5%的连接RTT超过500ms，如何优化网络较差用户的流媒体观看体验成为关键。

▐ 秒开卡顿模型

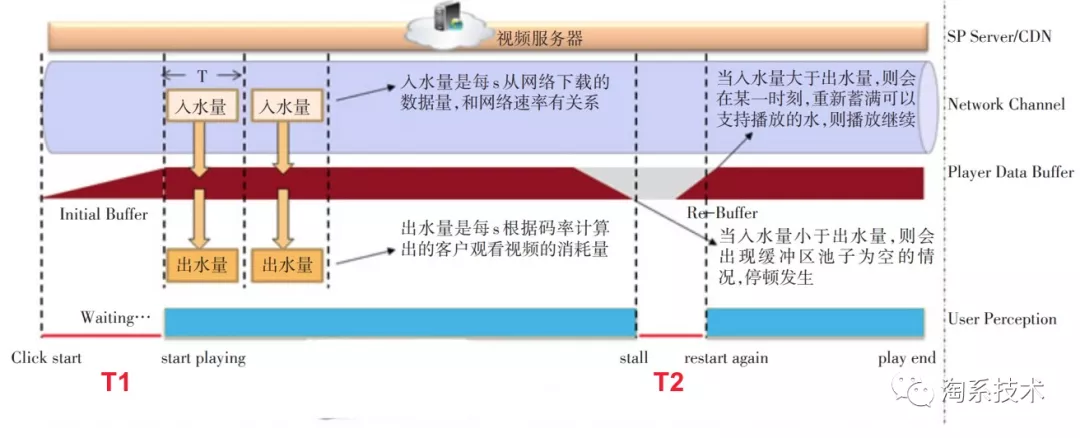


图7. 直播拉流模型

直播拉流可以理解为一个注水模型，上面是CDN服务器，中间是播放器缓冲区，可以理解成一个管道，下面是用户的体感，用户点击播放时，CDN不断向管道里注水，当水量达到播放器初始buffer时，首帧画面出现，然后播放器以一定速率排水，当水被排完时，播放器画面出现停顿，当重新蓄满支持播放的水后，继续播放。

我们假设Initial Buffer（首帧）为100K（实际调整以真实情况为准），起播时间 T1 < 1s 记为秒开，停顿时间 T2 > 100ms 记为卡顿。

****优化目标****

* 提升秒开：1s内下载完100K
* 降低卡顿：保持下载速率稳定，从而保持管道内始终有水

****核心思路****

* 提升秒开核心--快  
  高丢包率用户：加快重传

高延迟用户：减少往返次数

* 降低卡顿核心--稳  
  优化拥塞算法机制，稳定高效地利用带宽

▐ 拥塞算法选型

常见的拥塞算法可分为三类：

* 基于路径时延（如Vegas、Westwood）

将路径时延上升作为发生拥塞的信号，在单一的网络环境下（所有连接都使用基于路径时延的拥塞算法）是可行的，但是在复杂的网络环境下，带宽容易被其他算法抢占，带宽利用率最低。

* 基于丢包（如Cubic、NewReno）

将丢包作为发生拥塞的信号，其背后的逻辑是路由器、交换机的缓存都是有限的，拥塞会导致缓存用尽，进而队列中的一些报文会被丢弃。

拥塞会导致丢包，但是丢包却不一定拥塞导致的。事实上，丢包可以分为两类，一类是拥塞丢包，另一类是噪声丢包，特别是在无线网络环境中，数据以无线电的方式进行传递，无线路由器信号干扰、蜂窝信号不稳定等都会导致信号失真，最终数据链路层CRC校验失败将报文丢弃。

基于丢包的拥塞算法容易被噪声丢包干扰，在高丢包率高延迟的环境中带宽利用率较低。

* 基于带宽时延探测（如BBR）

既然无法区分拥塞丢包和噪声丢包，那么就不以丢包作为拥塞信号，而是通过探测最大带宽和最小路径时延来确定路径的容量。抗丢包能力强，带宽利用率高。

三种类型的拥塞算法没有谁好谁坏，都是顺应当时的网络环境的产物，随着路由器、交换机缓存越来越大，无线网络的比例越来越高，基于路径时延和基于丢包的的拥塞算法就显得不合时宜了。对于流媒体、文件上传等对带宽需求比较大的场景，BBR成为更优的选择。

▐ 秒开率优化

****✎ 加快握手包重传****

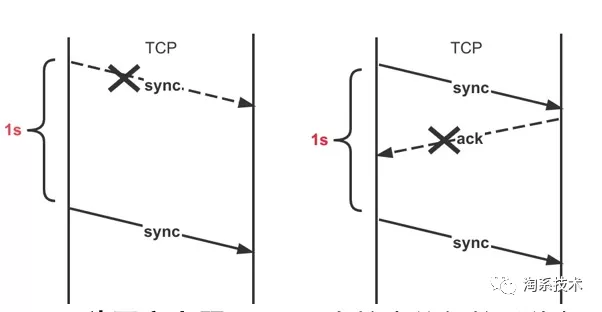


图8. TCP握手阶段出现丢包

如图，TCP在握手时，由于尚未收到对端的ACK，无法计算路径RTT，因此，RFC定义了初始重传超时，当超过这个超时时间还未收到对端ACK，就重发sync报文。

TCP秒开率上限：Linux内核中的初始重传超时为1s (RFC6298, June 2011)，3%的丢包率意味着TCP秒开率理论上限为97%，调低初始重传时间可以有效提升秒开率。

同理，如果你有一个RPC接口超时时间为1s，那么在3%丢包率的环境下，接口成功率不会超过97%。

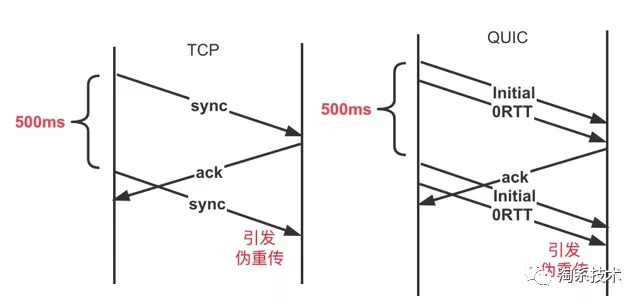


图9. TCP和QUIC握手阶段 - 伪重传模拟

另一方面，调低初始重传超时会引发伪重传，需要根据用户RTT分布进行取舍，比如初始重传超时调低到500ms，那么RTT大于500ms的用户在握手期间将会多发一个sync报文。

****✎ 减少往返次数****

慢启动阶段 N 个 RTT 内的吞吐量（不考虑丢包）:

T = init\_cwnd *(2^N-1)*MSS, N = ⌊t / RTT ⌋  
Linux内核初始拥塞窗口=10（RFC 6928，April 2013）

首帧100KB，需要4个RTT，如果RTT>250ms，意味着必然无法秒开。在我们举的这个例子中，如果调整为32，那么只需要2个RTT。

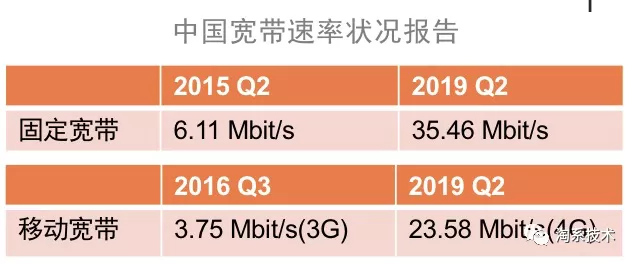


图10. 宽带速率报告

从2015到2019，固定带宽翻了4.8倍。从2016到2019，移动宽带翻了5.3倍。初始拥塞窗口从10调整为32在合理范围内。

▐ 卡顿率优化

****✎ BBR RTT探测优化****

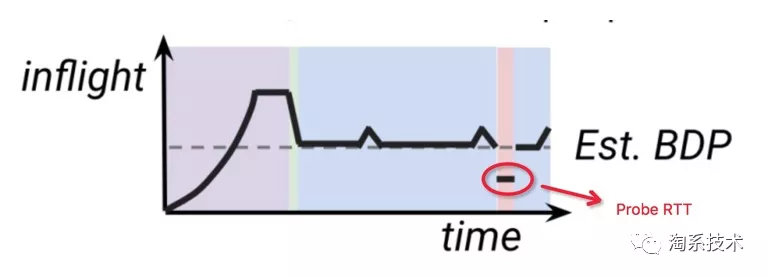


图11. BBR v1示意图：ProbeRTT阶段

****问题****

* BBR v1的ProbeRTT阶段会把inflight降到4\*packet并保持至少200ms。会导致传输速率断崖式下跌，引起卡顿
* 10s进入一次ProbeRTT，无法适应RTT频繁变化的场景

****优化方案****

* 减少带宽突降：inflight 降到 4*packet 改为降到 0.75*Estimated\_BDP
* 加快探测频率：ProbeRTT进入频率 10s 改为 2.5s

****推导过程****

* 为什么是 0.75x?

Max Estimated\_BDP = 1.25\*realBDP  
0.75 *Estimated\_BDP = 0.75*1.25 *realBDP = 0.9375* realBDP

保证inflight < realBDP, 确保RTT准确性。

* 为什么是 2.5s?

优化后BBR带宽利用率：(0.2s *75% + 2.5s*100%) / (0.2s+ 2.5s) = 98.1%

原生BBR带宽利用率：(0.2s *0% + 10s*100%) / (0.2s + 10s) = 98.0%

在整体带宽利用率不降低的情况下，调整到2.5s能达到更快感知网络变化的效果。

****优化效果****

保证带宽利用率不低于原生BBR的前提下，使得发送更平滑，更快探测到RTT的变化。

****✎ BBR带宽探测优化****

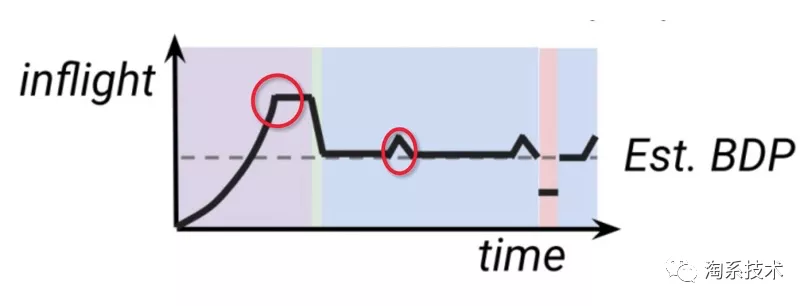


图12. BBR v1示意图：StartUp 和 ProbeBW阶段

****问题分析****

StartUp阶段2.89和ProbeBW阶段1.25的增益系数导致拥塞丢包，引发卡顿和重传率升高（Cubic重传率3%， BBR重传率4%）重传导致带宽成本增加1%。

****优化策略****

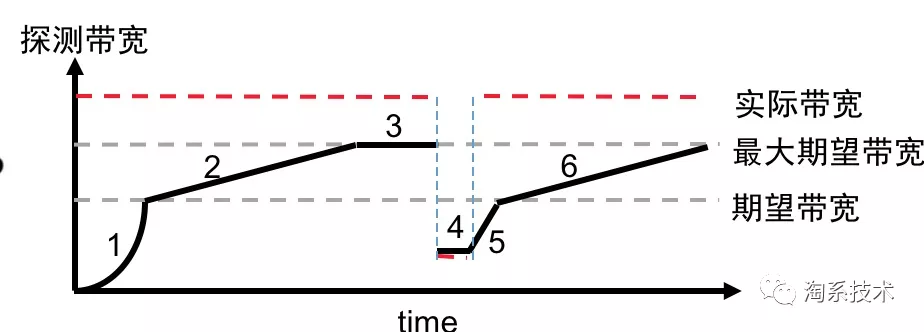


图13. 带宽变化示意图

定义两个参数：

期望带宽：满足业务需要的最小带宽  
最大期望带宽：能跑到的最大带宽

未达到期望带宽时采用较大的增益系数，较激进探测带宽。

达到期望带宽后采用较小的增益系数，较保守探测带宽。

****优化效果****

成本角度：重传率由4%降到3%，与Cubic一致，在不增加成本的前提下降低卡顿率。

另外，该策略可以推广到限速场景，如5G视频下载限速，避免浪费过多用户流量。

▐ 卡顿、秒开优化效果

在直播拉流场景下与TCP相比较，高峰期卡顿率降低30%+，秒开率提升2%。

▐ 写在优化最后的思考

最后，我们看看TCP初始重传超时和初始拥塞窗口的发展历程：

****初始重传时间****

* RFC1122 (October 1989) 为3s
* RFC6298 (June 2011) 改为1s，理由为97.5%的连接RTT<1s

****初始拥塞窗口****

* RFC 3390 (October 2002) 为 min(4 *MSS, max(2*MSS, 4380 bytes))（MSS=1460时为3）
* RFC 6928 (April 2013) 改为 min (10 *MSS, max (2*MSS, 14600))（MSS=1460时为10）—— 由google提出，理由为 90% of Google's search responses can fit in 10 segments (15KB).

首先IETF RFC是国际标准，需要考虑各个国家的网络情况，总会有一些网络较慢的地区。在TCP的RFC标准中，由于内核态实现不得不面临一刀切的参数选取方案，需要在考虑大盘分布的情况下兼顾长尾地区。

对比来看，QUIC作为用户态协议栈，其灵活性相比内核态实现的TCP有很大优势。未来我们甚至有机会为每个用户训练出所在网络环境最合适的一套最优算法和参数，也许可以称之为千人千面的网络体验优化：）

# **Multipath QUIC技术**

Multipath QUIC（多路径QUIC）是当前XQUIC内部正在研究和尝试落地的一项新技术。

MPQUIC可以同时利用cellular和wifi双通道进行数据传输，不仅提升了数据的下载和上传速度，同时也加强了应用对抗弱网的能力，从而进一步提高用户的端到端体验。此外，由于5G比4G的无线信号频率更高，5G的信道衰落问题也会更严重。所以在5G部署初期，基站不够密集的情况下如何保证良好信号覆盖是未来2-3年内手机视频应用的重大挑战，而我们研发的MPQUIC将为这些挑战提供有效的解决方案。

Multipath QUIC 的前身是MPTCP[6]。MPTCP在IETF有相对成熟的一整套RFC标准，但同样由于其实现在内核态，导致落地成本高，规模化推广相对困难。业界也有对MPTCP的应用先例，例如苹果在iOS内核态实现了MPTCP，并将其应用在Siri、Apple Push Notification Service和Apple Music中，用来保障消息的送达率、降低音乐播放的卡顿次数和卡顿时间。

我们在18年曾与手机厂商合作（MPTCP同样需要厂商支持），并尝试搭建demo服务器验证端到端的优化效果，实验环境下测试对「收藏夹商品展示耗时」与「直播间首帧播放耗时」降低12-50%不等，然而最终由于落地成本太高并未规模化使用。由于XQUIC的用户态协议栈能够大大降低规模化落地的成本，现在我们重新尝试在QUIC的传输层实现多路径技术。

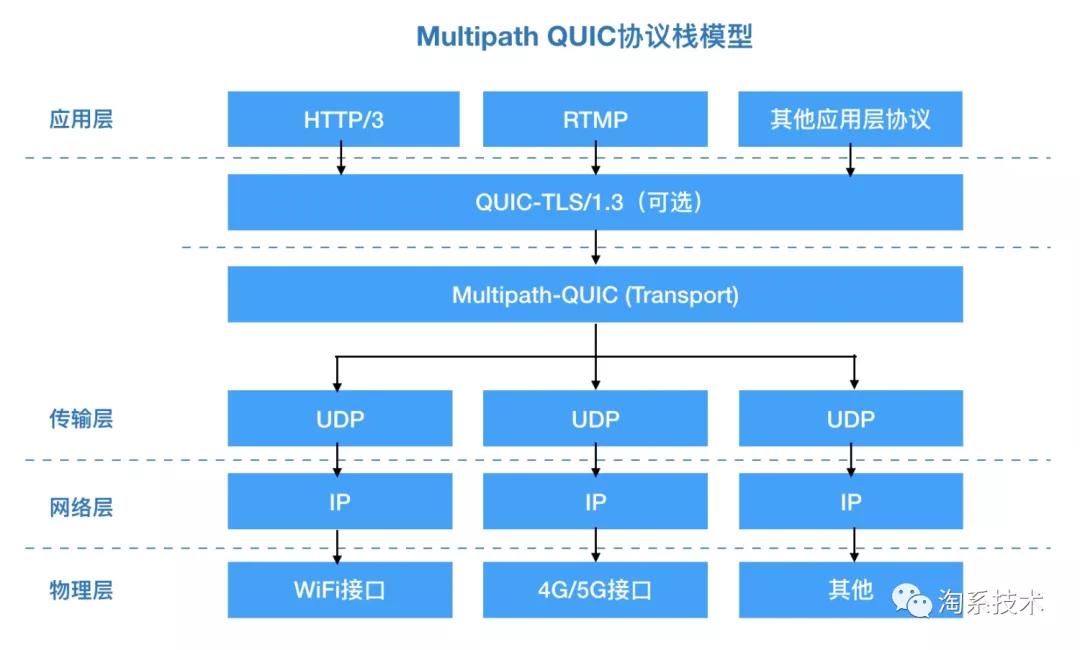


图15. Multipath QUIC协议栈模型

Multipath QUIC对于移动端用户最核心的提升在于，通过在协议栈层面实现多通道技术，能够在移动端同时复用Wi-Fi和蜂窝移动网络，来达到突破单条物理链路带宽上限的效果；在单边网络信号强度弱的情况下，可以通过另一条通道补偿。适用的业务场景包括上传、短视频点播和直播，可以提升网络传输速率、降低文件传输耗时、提升视频的秒开和卡顿率。在3GPP Release 17标准中也有可能将MPQUIC引入作为5G标准的一部分[7]。

技术层面上，和MPTCP不同，我们自研的MPQUIC采取了全新的算法设计，这使得MPQUIC相比于MPTCP性能更加优化，解决了slow path blocking问题。在弱网中的性能比以往提升30%以上。

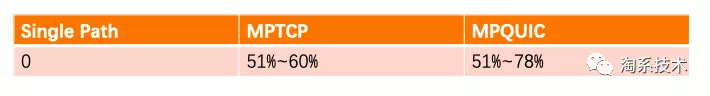


图16. 弱网下相对单路径的文件平均下载时间降低比例（实验数据）



图17. 弱网下相对单路径的视频播放卡顿率降低比例（实验数据）

在推进MPQUIC技术落地的过程中，我们将会尝试在IETF工作组推进我们的方案作为MPQUIC草案的部分内容[8]。期望能够为MPQUIC的RFC标准制定和落地贡献一份力量。

# **下一步的未来**

我们论证了QUIC的核心优势在于用户态的传输层实现（面向业务场景具备灵活调优的能力），而非单一针对弱网的优化。在业务场景的扩展方面，除了RPC、短视频、直播等场景外，XQUIC还会对其他场景例如上传链路等进行优化。

在5G逐步开始普及的时代背景下，IETF QUIC工作组预计也将在2020年底左右将QUIC草案发布为RFC标准，我们推测在5G大背景下QUIC的重要性将会进一步凸显。

▐ QUIC/MPQUIC对于5G

对于5G下eMBB（Enhanced Mobile Broadband）和URLLC（Ultra Reliable Low Latency）带来的不同高带宽/低延迟业务场景，QUIC将能够更好地发挥优势，贴合场景需求调整传输策略（拥塞控制算法、ACK/重传策略）。对于5G运营商提供的切片能力，QUIC同样可以针对不同的切片适配合适的算法组合，使得基础设施提供的传输能力能够尽量达到最大化利用的效果。在XQUIC的传输层实现设计中，同样预留了所需的适配能力。

在5G推广期间，在基站部署不够密集的情况下，保障稳定的有效带宽将会是音视频类的应用场景面临的巨大挑战。Multipath QUIC技术能够在用户态协议栈提供有效的解决方案，然而这项新技术仍然有很多难点需要攻克，同时3GPP标准化组织也在关注这一技术的发展情况。

▐ XQUIC开源计划

阿里淘系技术架构团队计划在2020年底开源XQUIC，期望能够帮助加速IETF标准化QUIC的推广，并期待更多的开源社区开发者参与到这个项目中来。

****附录：参考文献****

[1] SPDY - HTTP/2.0原型，由Google主导的支持双工通信的应用层协议

[2] [HTTP/2.0](https://developer.aliyun.com/article/770062)

[3] [TLS/1.3](https://developer.aliyun.com/article/770062)

[4] GQUIC - 指Google QUIC版本，与IETF QUIC草案版本有一定差异

[5] [IETF QUIC - 指IETF QUIC工作组正在推进的QUIC系列草案：](https://developer.aliyun.com/article/770062)包括QUIC-Transport、QUIC-TLS、QUIC-recovery、HTTP/3.0、QPACK等一系列草案内容

[6] [MPTCP](https://developer.aliyun.com/article/770062)

[7] [3GPP向IETF提出的需求说明](https://developer.aliyun.com/article/770062)

[8] [MPQUIC - 我们正在尝试推进的草案](https://developer.aliyun.com/article/770062)

****淘系架构与基础服务团队****

致力于为淘系和阿里提供基础核心能力、产品与解决方案：

• 下一代网络协议QUIC的实现与落地 - XQUIC  
• 自适应高可用解决方案与核心能力 - Noah  
• 新一代业务研发模式平台 - Gaia  
• 支撑整个阿里的移动中间件体系（千万级QPS接入网关、API网关、推送/消息、移动配置中心等）  
团队大牛云集~~ 想要加入我们，请邮件简历至：miaoji.lym@alibaba-inc.com