首先介绍一下携程App的网络服务架构。由于携程业务众多，开发资源导致无法全部使用Native来实现业务逻辑，因此有相当一部分频道基于Hybrid实现。网络通讯属于基础&业务框架层中基础设施的一部分，为App提供统一的网络服务：

Native端的网络服务

Native模块是携程的核心业务模块（酒店、机票、火车票、攻略等），Native模块的网络服务主要通过TCP连接实现，而非常见的Restful HTTP API那种HTTP连接，只有少数轻量级服务使用HTTP接口作为补充。

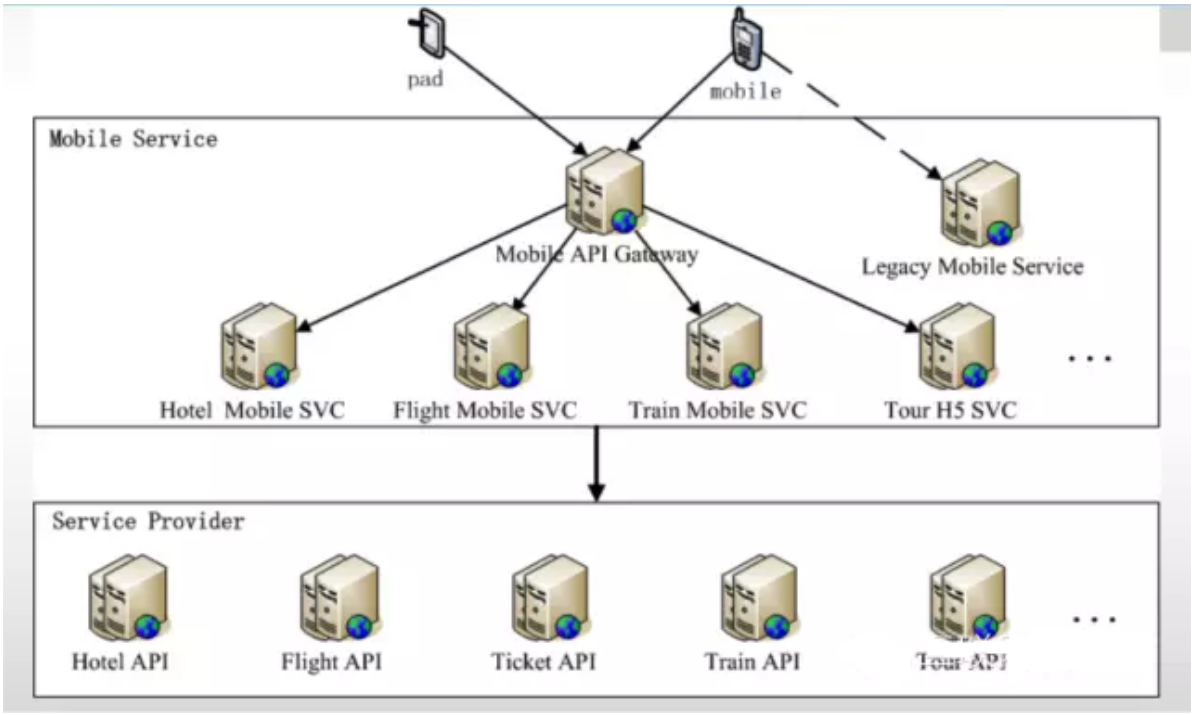
TCP连接网络服务模块使用了长连接+短连接机制，即有一个长连接池保持一定数目长连接，用于减少每次服务额外的连接，服务完成后会将该连接Socket放回长连接池，继续保持连接状态（一段时间空闲后会被回收）；短连接作为补充，每次服务完成后便会主动关闭连接。

TCP网络服务的Payload使用的是自定义的数据及序列化协议；HTTP服务的Payload比较简单，即常见的JSON数据格式。

Hybrid端的网络服务

Hybrid模块由于是在WebView中展示本地或者直连的H5页面，页面逻辑发起的网络服务都是通过系统WebView的HTTP请求实现的。少量业务场景（需要加密和支付等）以Hybrid桥接接口形式的Native TCP通道来完成网络服务。

下图是网络服务的部署架构图：



携程App所有网络服务，无论是TCP还是HTTP都会先连接到一个API Gateway服务器。如果是TCP服务，会先连接上TCP Gateway，TCP Gateway会负责将请求通过HTTP转发到后端的SOA服务接口。HTTP Gateway的原理与之类似。TCP Gateway和HTTP Gateway的区别仅仅在客户端到服务端的连接方式不同而已。Gateway的作用除了业务请求还有流量控制和熔断。

要发现常见网络性能问题，先来看看一个网络服务做了哪些事情：

1.DNS Lookup

2.TCP Handshake

3.TLS Handshake

4.TCP/HTTP Request/Response

首先会是DNS解析，然后TCP连接握手，TLS连接握手（如果有的话），连接成功后再发送TCP或HTTP请求以及收到响应。如果能够将这些过程逐一梳理并确保不会存在明显的性能问题，那么基本可以确保获得不错的网络性能。网络服务里有一个重要的性能标准，即RTT(Round-Trip Time)，往返时延，它表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认）所间隔的时间。理想情况下可以假设4G网络RTT为100ms，3G网络RTT为200ms，很容易就能计算出我们的App网络服务耗时的下限，当然还要加上服务器处理时间。

常见的网络性能问题有如下几种：

- **问题一：DNS问题**

DNS出问题的概率其实比大家感觉的要大，首先是DNS被劫持或者失效，2015年初业内比较知名的就有Apple内部DNS问题导致App Store、iTunes Connect账户无法登录；京东因为CDN域名付费问题导致服务停摆。携程在去年11月也遇到过DNS问题，主域名被国外服务商误列入黑名单，导致主站和H5等所有站点无法访问，但是App客户端的Native服务都正常，原因后面介绍。

另一个常见问题就是DNS解析慢或者失败，例如国内中国运营商网络的DNS就很慢，一次DNS查询的耗时甚至都能赶上一次连接的耗时，尤其2G网络情况下，DNS解析失败是很常见的。因此如果直接使用DNS，对于首次网络服务请求耗时和整体服务成功率都有非常大的影响。

- **问题二：TCP连接问题**

DNS成功后拿到IP，便可以发起TCP连接。HTTP协议的网络层也是TCP连接，因此TCP连接的成功和耗时也成为网络性能的一个因素。我们发现常见的问题有TCP端口被封（例如上海长宽对非HTTP常见端口80、8080、443的封锁），以及TCP连接超时时长问题。端口被封，直接导致无法连接；连接超时时长过短，在低速网络上可能总是无法连接成果；连接超时过长，又有可能导致用户长时间等待，用户体验差。很多时候尽快失败重新发起一次连接会很快，这也是移动网络带宽不稳定情况下的一个常见情况。

- **问题三：Write/Read问题**

DNS Lookup和TCP连接成功后，就会开始发送Request，服务端处理后返回Response，如果是HTTP连接，业内大部分App是使用第三方SDK或者系统提供的API来实现，那么只能设置些缓存策略和超时时间。iOS上的NSURLConnection超时时间在不同版本上还有不同的定义，很多时候需要自己设置Timer来实现；如果是直接使用TCP连接实现网络服务，就要自己对读写超时时间负责，与网络连接超时时长参数类似，太小了在低速网络很容易读写失败，太大了又可能影响用户体验，因此需要非常小心地处理。

我们还遇到另一类问题，某些酒店Wi-Fi对使用非80、8080和443等常见HTTP端口的服务进行了限制，即使发送Request是正常的，服务端能够正常收到，但是Response却被酒店网络proxy或防火墙拦截，客户端最终会等待读取超时。

移动网络和传统网络另一个很大的区别是Connection Migration问题。定义一个Socket连接是四元组（客户端IP，客户端Port，服务端IP，服务端Port），当用户的网络在WIFI/4G/3G/2G类型中切换时，其客户端IP会发生变化，如果此时正在进行网络服务通讯，那么Socket连接自身已经失效，最终也会导致网络服务失败。

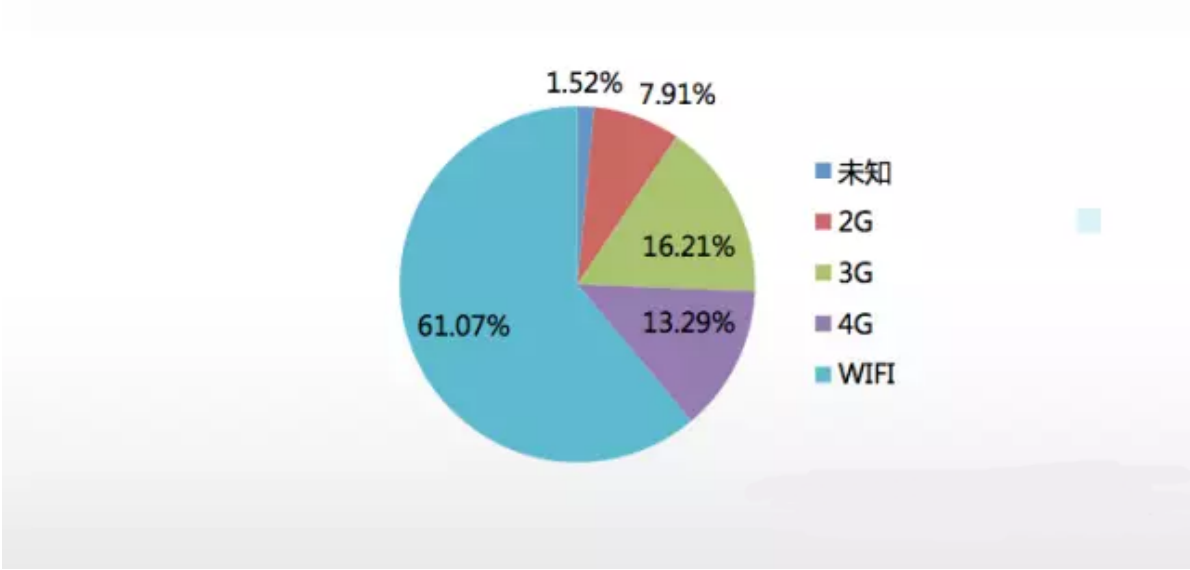
- **问题四：传输Payload过大**

传的多就传的慢，如果没做过特别优化，传输Payload可能会比实际所需要的大很多，那么对于整体网络服务耗时影响非常大。

- **问题五：复杂的国内外网络情况**

国内运营商互联和海外访问国内带宽低传输慢的问题也令人难非常头疼。

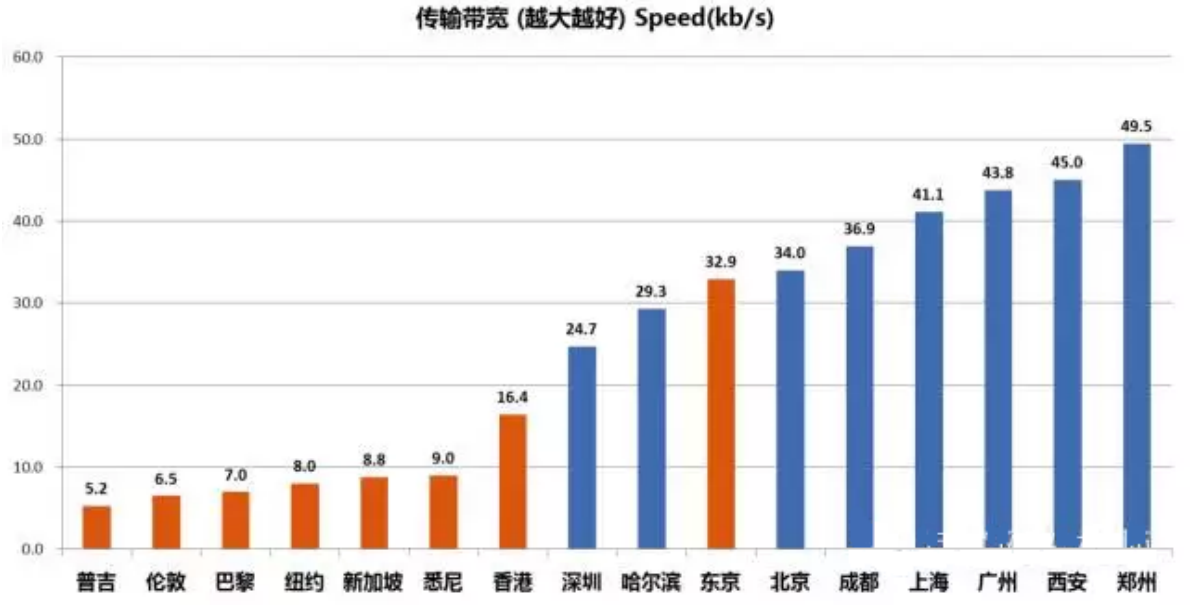
看下携程App用户的网络类型分布：

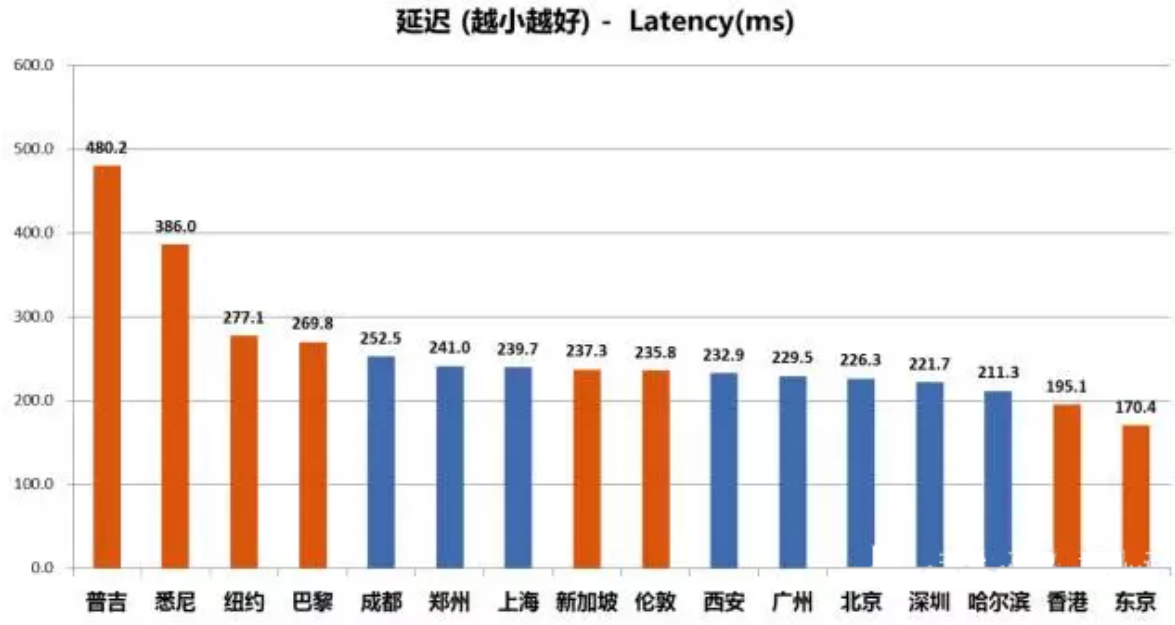


Wi-Fi用户占比已超过60%，4G用户量正接近3G用户量，2G用户在逐步减少，用户的网络越来越好。4G/3G/2G网络的带宽和延迟差别很大，而带宽和延迟是网络性能的重要指标：

网络制式	下行带宽	上行带宽	延迟
4G	400-600KB/S	200-500KB/S	80-200ms
3G	100-200KB/S	10-100KB/S	100-400ms
2G	10KB/S	1KB/S	400+ms

针对携程App用户的网络带宽和延迟，我们采样了海内外各8个城市的数据：





注意网络带宽和延迟并没有直接相关性，带宽高不意味着延迟低，延迟再低也不可能快过光速。从上图我们可以看到海内外带宽相差很大，但是延迟基本一致。

针对上面这些问题，在网络复杂环境和国内运营商互通状况无能为力的情况下，就针对性地逐一优化，以期达到目标：连得上、连得快、传输时间短。

- **优化实践一：优化DNS解析和缓存**

由于我们的App网络服务主要基于TCP连接，为了将DNS时间降至最低，我们内置了Server IP列表，该列表可以在App启动服务中下发更新。App启动后的首次网络服务会从Server IP列表中取一个IP地址进行TCP连接，同时DNS解析会并行进行，DNS成功后，会返回最适合用户网络的Server IP，那么这个Server IP会被加入到Server IP列表中被优先使用。

Server IP列表是有权重机制的，DNS解析返回的IP很明显具有最高的权重，每次从Server IP列表中取IP会取权重最高的IP。列表中IP权重也是动态更新的，根据连接或者服务的成功失败来动态调整，这样即使DNS解析失败，用户在使用一段时间后也会选取到适合的Server IP。

- **优化实践二：网络质量检测（根据网络质量来改变策略）**

针对网络连接和读写操作的超时时间，我们提出了网络质量检测机制。目前做到的是根据用户是在2G/3G/4G/Wi-Fi的网络环境来设置不同的超时参数，以及网络服务的并发数量。2G/3G/4G网络环境对并发TCP连接的数量是有限制的（2G网络下运营商经常只能允许单个Host一个TCP连接），因此网络服务重要参数能够根据网络质量状况来动态设定对性能和体验都非常重要。

不过目前携程App网络00质量检测的粒度还比较粗，我们正在做的优化就是能够测算到用户当前的网络RTT，根据RTT值来设置参数，那会更加准确。Facebook App的做法是HTTP网络服务在HTTP Response的Header中下发了预估的RTT值，客户端根据这个RTT值便能够设计不同的产品和服务策略。

- **优化实践三：提供网络服务优先级和依赖机制**

由于网络对并发TCP连接的限制，就需要能够控制不必要的网络服务数量，因此我们在通讯模块中加入了网络服务优先级和依赖机制。发送一个网络服务，可以设置它的优先级，高优先级的服务优先使用长连接， 低优先级的就是用短连接。长连接由于是从长连接池中取到的TCP连接，因此节省了TCP连接时间。

网络服务依赖机制是指可以设置数个服务的依赖关系，即主从服务。假设一个App页面要发多个服务，主服务成功的情况下，才去发子服务，如果主服务失败了，自服务就无需再关心成功或者失败，会直接被取消。如果主服务成功了，那么子服务就会自动触发。

- **优化实践四：提供网络服务重发机制**

移动网络不稳定，如果一次网络服务失败，就立刻反馈给用户你失败了，体验并不友好。我们提供了网络服务重发机制，即当网络服务在连接失败、写Request失败、读Response失败时自动重发服务；长连接失败时就用短连接来做重发补偿，短连接服务失败时当然还是用短连接来补偿。这种机制增加了用户体验到的服务成功概率。

当然不是所有网络服务都可以重发，例如当下订单服务在读取Response失败时，就不能重发，因为下单请求可能已经到达服务器，此时重发服务可能会造成重复订单，所以我们添加了重发服务开关，业务段可以自行控制是否需要。

- **优化实践五：减少数据传输量**

我们优化了TCP服务Payload数据的格式和序列化/反序列化算法，从自定义格式转换到了Protocol Buffer数据格式，效果非常明显。序列化/反序列算法也做了调整，如果大家使用JSON数据格式，选用一个高效的反序列化算法，针对真实业务数据进行测试，收益明显。

图片格式优化在业界已有成熟的方案，例如Facebook使用的WebP图片格式，已经被国内众多App使用。

- **优化实践六：优化海外网络性能**

海外网络性能的优化手段主要是通过花钱，例如CDN加速，提高带宽，实现动静资源分离，对于App中的Hybrid模块优化效果非常明显。

经过上面的优化手段，携程App的网络性能从优化之初的V5.9版本到现在V6.4版本，服务成功率已经有了大幅提升，核心服务成功率都在99%以上。注意这是客户端采集的服务成功率，即用户感知到的网络服务成功率，失败量中包含了客户端无网络和服务端的错误。网络服务平均耗时下降了150-200ms。我们的目标是除2G网络外，核心业务的网络服务成功率都能够达到三个九。

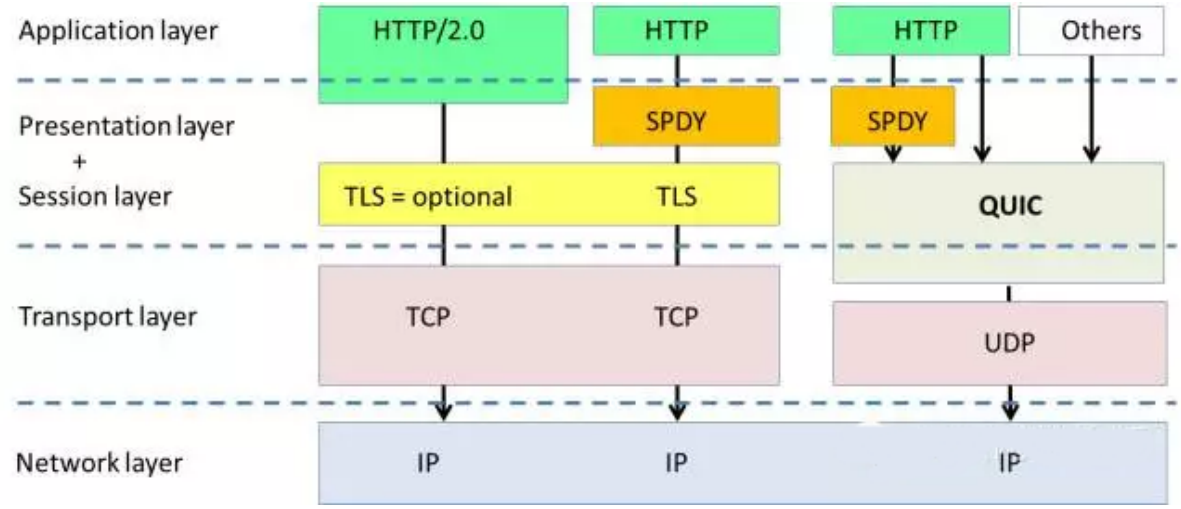
数据格式优化的效果尤其明显，采用新的Protocol Buffer数据格式+Gzip压缩后的Payload大小降低了15%-45%。数据序列化耗时下降了80%-90%。

经历了这半年的网络性能优化，体会最深的就是Logging基础设施的重要性。如果我们没有完整端到端监控和统计的能力，性能优化只能是盲人摸象。Logging基础设施需要包括客户端埋点采集、服务端T+0处理、后期分析、Portal展示、自动告警等多种功能，这也不是单纯的客户端框架团队可以完成的，而需要公司多个部门合作完成。

携程基于Elastic Search开发了网络实时监控Portal，能够实时监控所有的网络服务，包括多种维度，可以跟踪到单个目标用户的所有网络请求信息。

用户的性能数据都被喷到Haddop和Hive大数据平台，我们可以轻松制定并分析网络性能KPI，例如服务成功率、服务耗时、连接成功率和连接耗时等，我们做到了在时间、网络类型、城市、长短连接、服务号等多纬度的分析。下图是我们的网络性能KPI Portal，可以查看任一服务的成功率，服务耗时、操作系统、版本等各种信息，对于某个服务的性能分析非常有用。

最后看看业界网络性能优化的新技术方向，目前最有潜力的是Google推出的SPDY和QUIC协议。



SDPY已成为HTTP/2.0 Draft，有希望成为未来HTTP协议的新标准。HTTP/2.0提供了很多诱人的特性（多路复用、请求优先级、支持服务端推送、压缩HTTP Header、强制SSL传输、对服务端程序透明等）。国内很多App包括美团、淘宝、支付宝都已经在尝试使用SDPY协议，Twitter的性能测试表明可以降低30%的网络延迟，携程也做了性能测试，由于和TCP性能差距不明显，暂未在生产上使用。

QUIC是基于UDP实现的新网络协议，由于TCP协议实现已经内置在操作系统和路由器内核中，Google无法直接改进TCP，因此基于无连接的UDP协议来设计全新协议可以获得很多好处。首先能够大幅减少连接时间，QUIC可以在发送数据前只需要0 RTT时间，而传统TCP/TLS连接至少需要1-3 RTT时间才能完成连接（即使采用Fast-Open TCP或TLS Snapshot）；其次可以解决TCP Head-of-Line Blocking问题，通常前一个TCP Packet发送成功前会拥塞后面的Packet发送，而QUIC可以避免这样的问题；QUIC也有更好的移动网络环境下拥塞控制算法；新的连接方式也大幅减少了Connectiont Migration问题的影响。

随着这些新协议的逐渐成熟，相信未来能够进一步提高移动端的网络服务性能，值得大家保持关注。