携程 App 的网络性能优化实践

作者：陈浩然

阅读数：25518 | 2015 年 4 月 24 日

编者按：在 4 月 23 日~25 日举行的 QCon 全球软件开发大会（北京站）上，携程无线开发总监陈浩然分享了《移动开发网络性能优化实践》，总结了携程在 App 网络性能优化方面的一些实践经验。在 2014 年接手携程无线 App 的框架和基础研发工作之后，陈浩然面对的首要工作就是 App 客户端性能优化，尤其是网络服务性能，这是所有 App 优化工作的重中之重。以下为正文。

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。

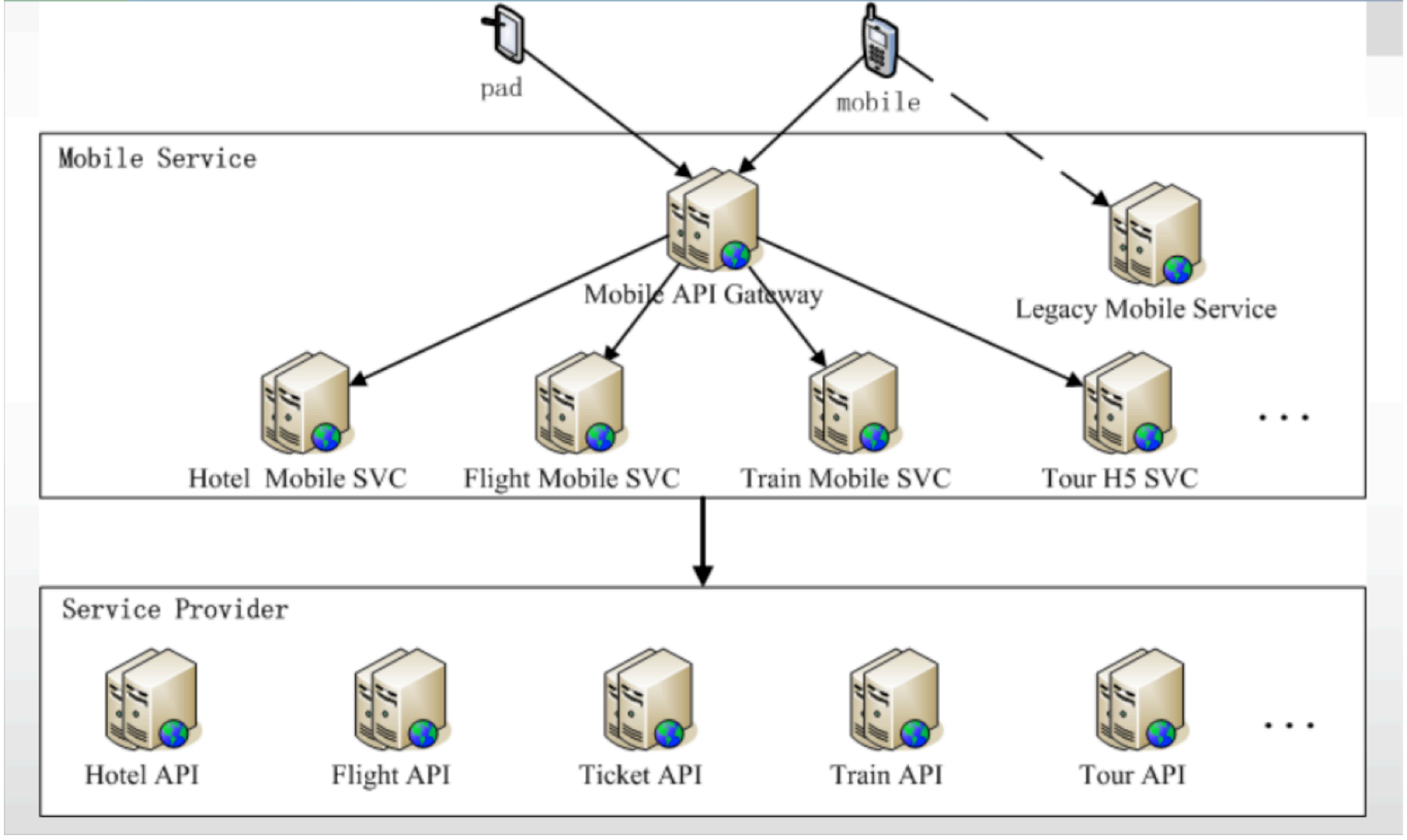
TCP 连接网络服务模块使用了长连接 + 短连接机制，即有一个长连接池保持一定数目长连接，用于减少每次服务额外的连接，服务完成后会将该连接 Socket 放回长连接池，继续保持连接状态（一段时间空闲后会被回收）；短连接作为补充，每次服务完成后便会主动关闭连接。

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。

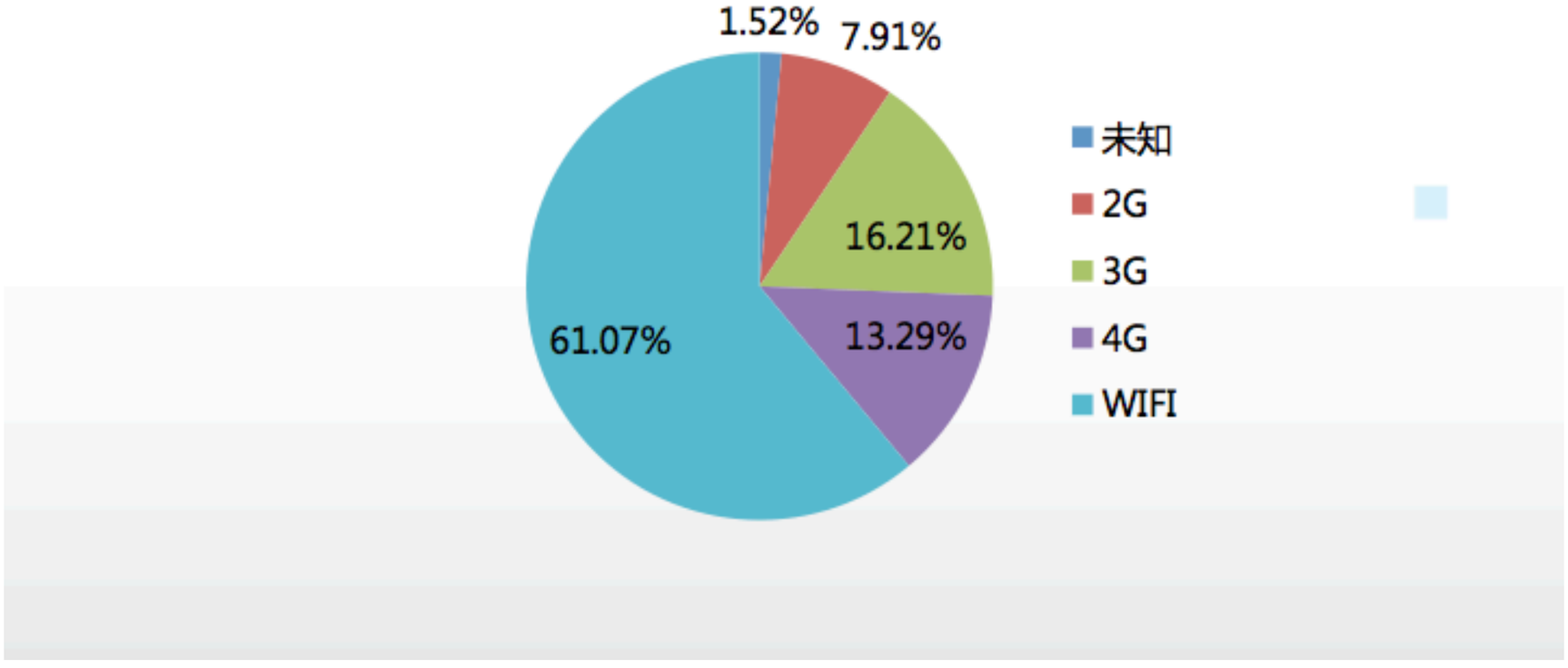
- 问题四：传输 Payload 过大

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：

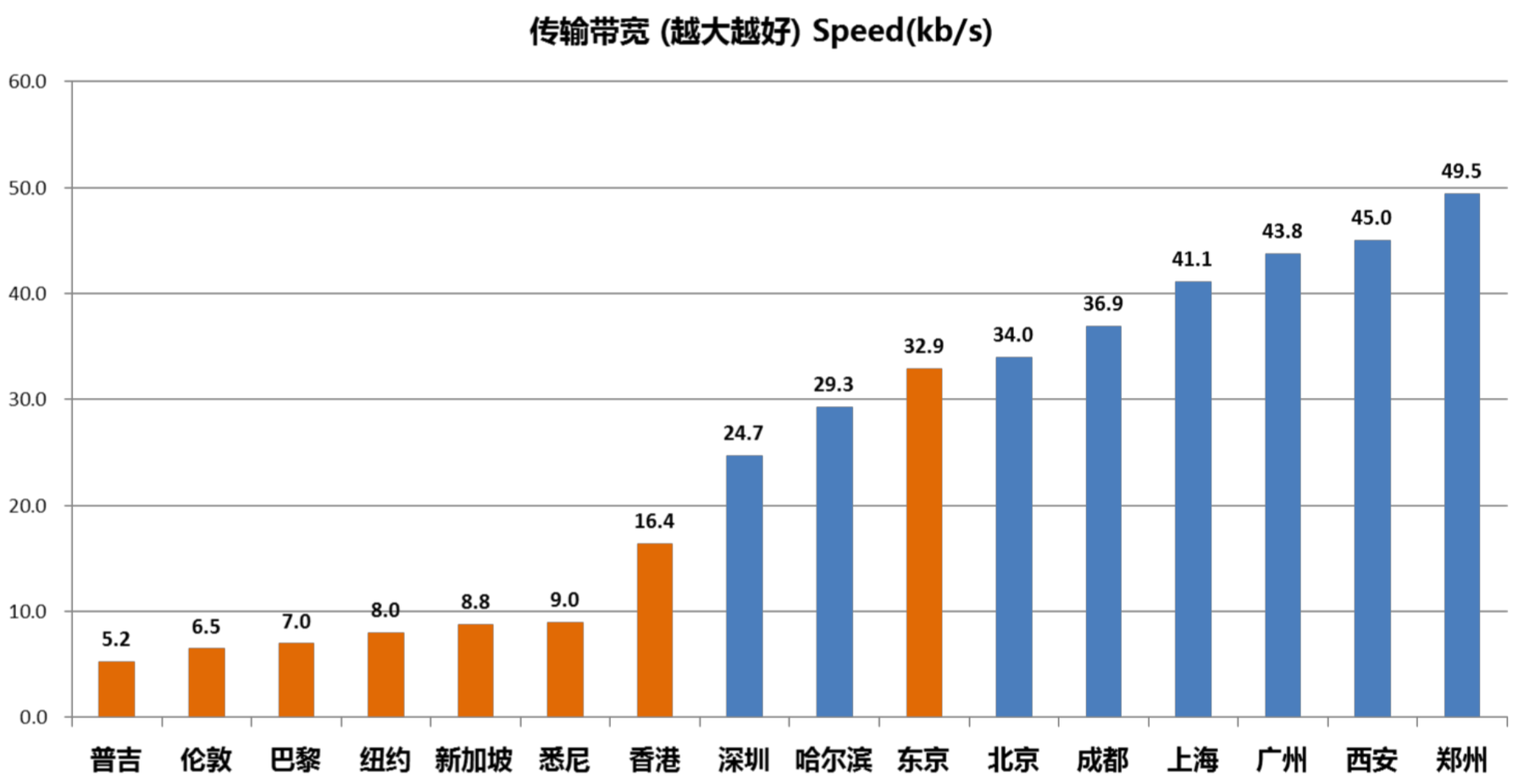


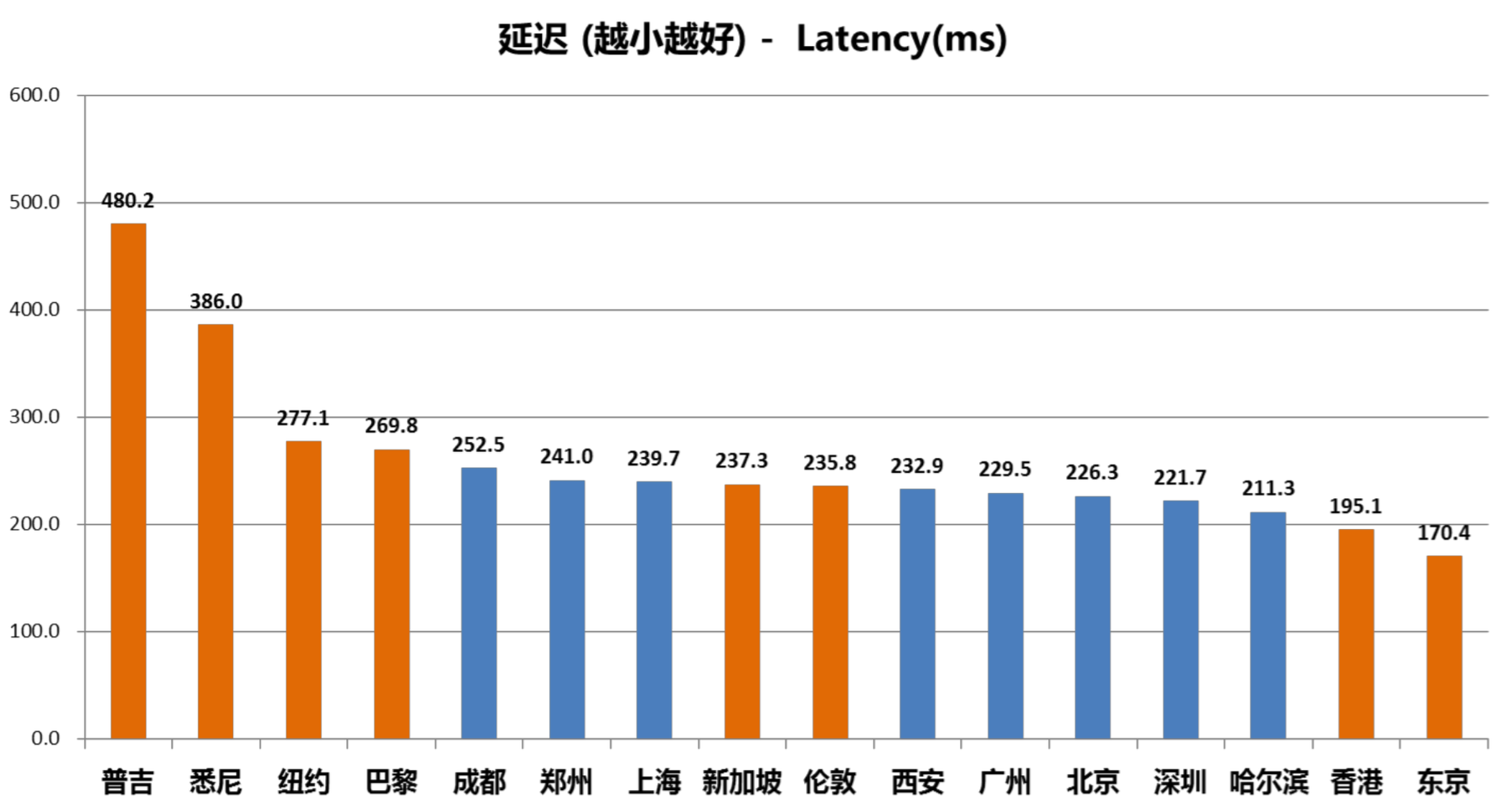
Wi-Fi 用户占比已超过 60%，4G 用户量正接近 3G 用户量，2G 用户在逐步减少，用户的网络越来越好。

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。

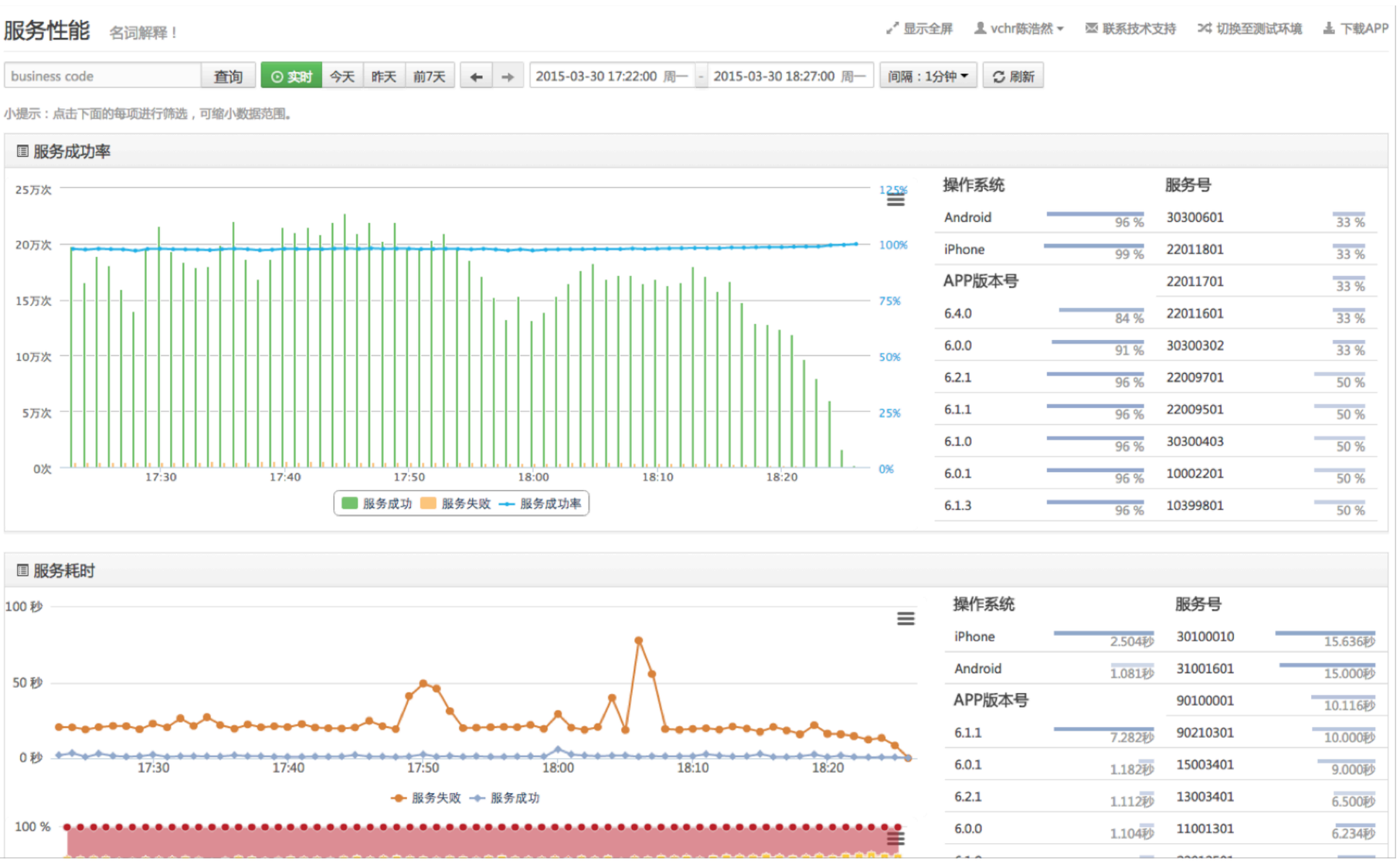
经过上面的优化手段，携程 App 的网络性能从优化之初的 V5.9 版本到现在 V6.4 版本，服务成功率已经有了大幅提升，核心服务成功率都在 99% 以上。注意这是客户端采集的服务成功率，即用户感知到的网络服务成功率，失败量中包含了客户端无网络和服务端的错误。网络服务平均耗时下降了 150-200ms。我们的目标是除 2G 网络外，核心业务的网络服务成功率都能够达到三个九。

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%。

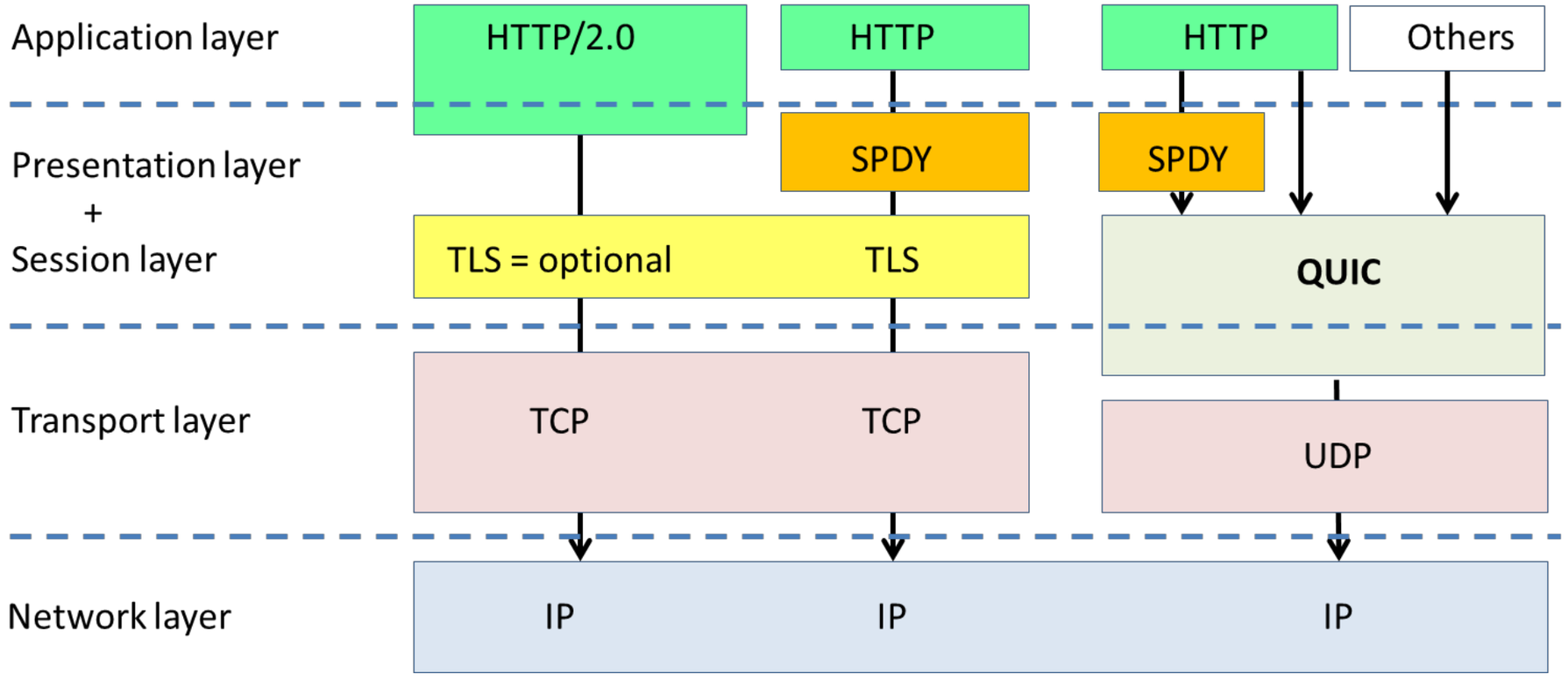
经历了这半年的网络性能优化，体会最深的就是 Logging 基础设施的重要性。如果我们没有完整端到端监控和统计的能力，性能优化只能是盲人摸象。Logging 基础设施需要包括客户端埋点采集、服务端 T+0 处理、后期分析、Portal 展示、自动告警等多种功能，这也不是单纯的客户端框架团队可以完成的，而需要公司多个部门合作完成。

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。



SPDY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 SPDY 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。

移动 QCon DevOps 语言 & 开发 架构 文化 & 方法

发表评论

注册/登录 InfoQ 发表评论

注册/登录

InfoQ

促进软件开发领域知识与创新的传播

特别专题

百度技术沙龙　云+未来　Intel　华为云 MeetUp
百度 AI　AWS　云+社区开发者大会
迅雷链技术专区　工业大数据创新竞赛

关于我们

关于我们
合作伙伴
关注我们
我要投稿
加入我们

联系我们

内容投稿：editors@geekbang.com
业务合作：hezuo@geekbang.org
反馈投诉：feedback@geekbang.org

InfoQ 近期会议

软件开发大会 2019年5月6-8日
QCon广州站 2019年5月27-28日
架构师峰会 2019年7月12-13日
大前端技术大会 2019年6月20-21日

全球 InfoQ

InfoQ En
InfoQ 日本
InfoQ Fr
InfoQ Br