okhttp离线缓存实现逻辑

```
OkHttpClient okHttpClient = new OkHttpClient();
OkHttpClient newClient = okHttpClient.newBuilder()
        .addNetworkInterceptor(new CacheInterceptor())
        .cache(new Cache(new File(this.getExternalCacheDir(), "okhttpcache"), 10
* 1024 * 1024))
        .connectTimeout(20, TimeUnit.SECONDS)
        .readTimeout(20, TimeUnit.SECONDS)
        .build();
```

主要就是设置拦截器和缓存时间（addNetworkInterceptor和cache）

```
public class CacheInterceptor implements Interceptor {
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request();
        Response response = chain.proceed(request);
        Response response1 = response.newBuilder()
                .removeHeader("Pragma")
                .removeHeader("Cache-Control")
                //cache for 30 days
                .header("Cache-Control", "max-age=" + 3600 * 24 * 30)
                .build();
        return response1;
    }
```

> 这样配置一下就缓存就实现了。

页面加载超过3秒，57%的用户会离开。

Amazon页面加载延长1秒，一年就会减少16亿美金营收

在做网络优化前，我们首先要为网络通信质量设立一个标尺。

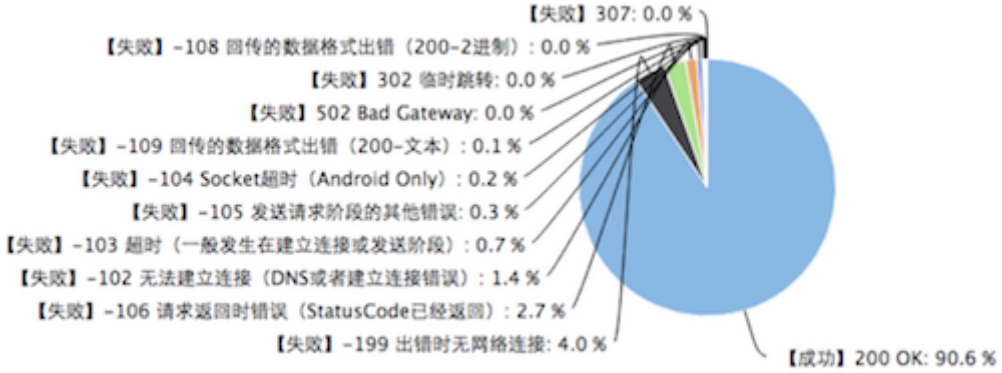
在网络优化，监控团队开发了基于端到端的客户端监控平台。这里要先解释一下“端到端”的含义：是指请求从客户端发出到服务端响应返回的整个过程。它区别于后台服务监控，是一种从用户角度观察到的真实体验监控。

监控页面如图所示：

1

通过基于命令字的、多维度、实时的监控工具，可以及时发现线上问题。

为了方便发现问题，我们在公司内统一了网络响应状态码的范围。通过状态码的分段范围，也可以迅速清晰地看到网络成败的原因和占比。



2

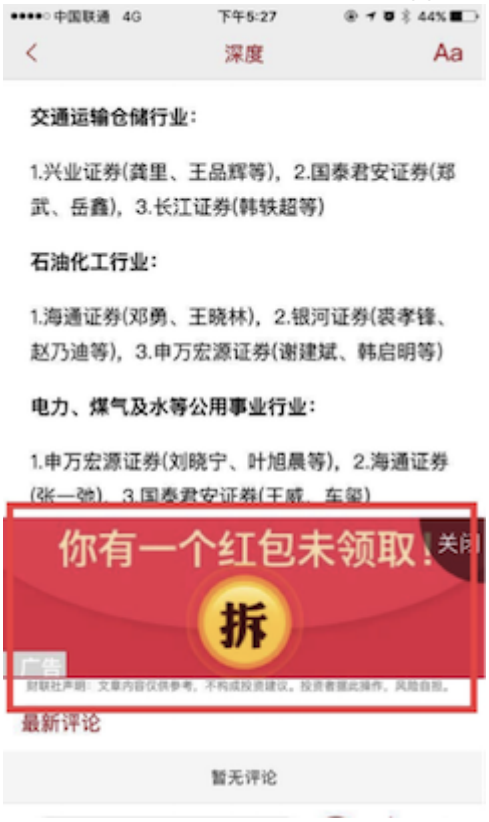
有了监控工具后，我们来讨论：移动网络请求过程中，出现了哪些最常见的问题？

首先是网络不可用的问题。主要由以下几种原因导致：

- GFW的拦截，原因你懂的。
- DNS的劫持，端口的意外封禁等。
- 偏远地区网络基础设施比较差。

其次是网络加载时间长。原因包括： * 移动设备出于省电的目的，发出网络请求前需要先预热通信芯片。 * 网络请求需要跨网络运营商，物理路径长。 * HTTP请求是基于Socket设计的，请求发起之前会经历三次握手，断开时又会进行四次挥手。

最后是HTTP协议的数据安全问题。原因有： * HTTP协议的数据容易被抓包。Post包体数据经过加密能够避免泄露，但协议中的URL和header部分还是会暴露给抓包软件。HTTPS也面临相似的问题。 * 运营商数据恶意篡改严重。如下图中，App的网页中就被运营商插入了广告。



3

面对上述网络问题，我们首先在HTTP短连请求中进行了一些优化尝试。

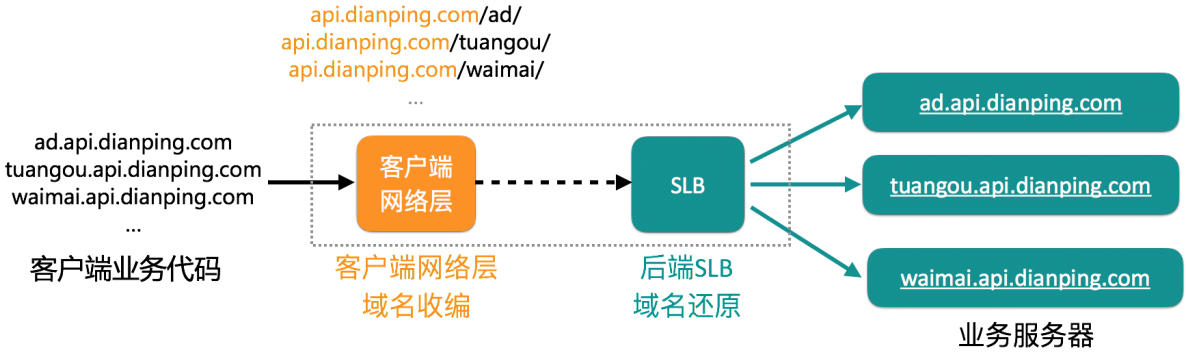
短连方案一、域名合并方案

随着开发规模逐渐扩大，各业务团队出于独立性和稳定性的考虑，纷纷申请了自己的三级域名。App中的API域名越来越多。如下所示： search.api.dianping.com ad.api.dianping.com tuangou.api.dianping.com waimai.api.dianping.com movie.api.dianping.com ...

App中域名多了之后，将面临下面几个问题： * HTTP请求需要跟不同服务器建立连接。增加了网络的并发连接数量。 * 每条域名都需要经过DNS服务来解析服务器IP。

如果想将所有的三级域名都合并为一个域名，又会面临巨大的项目推进难题。因为不同业务团队当初正是出于独立性和稳定性的考虑才把域名进行拆分，现在再想把域名合并起来，势必会遭遇巨大的阻力。

所以我们面临的是：既要将域名合并，提升网络连接效率，又不能改造后端业务服务器。经过讨论，我们想到了一个折中的方案。



该方案的核心思想在于：保持客户端业务层代码编写的网络请求与后端业务服务器收到的请求保持一致，请求发出前，在客户端网络层对域名收编，请求送入后端，在SLB(Server Load Balancing)中对域名进行还原。

网络请求发出前，在客户端的网络底层将URL中的域名做简单的替换，我们称之为“域名收编”。

例如：URL “http://ad.api.dianping.com/command?param1=123" 在网络底层被修改为 “http://api.dianping.com/ad/command?param1=123" 。

这里，将域名”ad.api.dianping.com”替换成了”api.dianping.com”，而紧跟在域名后的其后的”ad”表示了这是一条与广告业务有关的域名请求。

依此类推，所有URL的域名都被合并为”api.dianping.com”。子级域名信息被隐藏在了域名后的path中。

被改造的请求被送到网络后端，在SLB中，拥有与客户端网络层相反的一套域名反收编逻辑，称为“域名还原”。

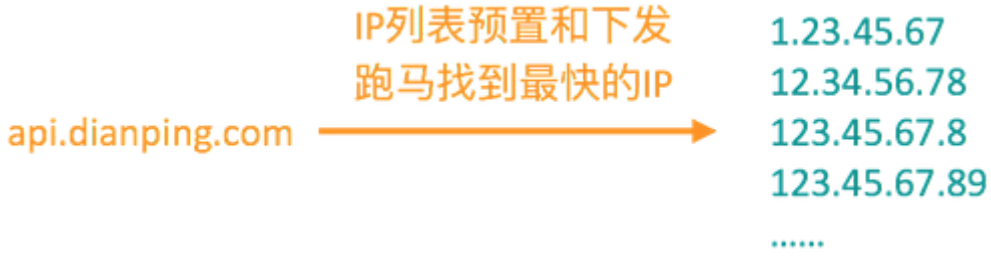
例如：”http://api.dianping.com/ad/command?param1=123" 在SLB中被还原为 “http://ad.api.dianping.com/command?param1=123" 。 SLB的作用是将请求分发到不同的业务服务器，在经过域名还原之后，请求已经与客户端业务代码中原始请求一致了。

该方案具有如下优势： 1. 域名得到了收编，减少了DNS调用次数，降低了DNS劫持风险。 2. 针对同一域名，可以利用Keep-Alive来复用Http的连接。 3. 客户端业务层不需要修改代码，后端业务服务也不需要进行任何修改。

短连方案二、IP直连方案

经过域名合并方案，我们已经将所有的域名都统一成了”api.dianping.com”。针对这唯一的域名，我们可以在客户端架设自己的DNS服务。

方案很简单：程序启动的时候拉取”api.dianping.com”对应的所有的IP列表；对所有IP进行跑马测试，找到速度最快的IP。后续所有的HTTPS请求都将域名更换为跑马最快的IP即可。



举个例子，假如：经过跑马测试发现域名”api.dianping.com”对应最快的IP是”1.23.456.789”。

URL”http://api.dianping.com/ad/command?param1=123"将被替换为"http://1.23.456.789/ad/command?param1=123"

IP直连方案有下面几大优势： 1. 摒弃了系统DNS，减少外界干扰，摆脱DNS劫持困扰。 2. 自建DNS更新时机可以控制。 3. IP列表更换方便。

此外，如果你的App域名没有经过合并，域名比较多，也建议可以尝试使用HttpDNS方案。参考：http://www.tuicool.com/articles/7nAJBb 对HTTPS中的证书处理： > HTTPS由于要求证书绑定域名，如果做IP直连方案可能会遇到一些麻烦，这时我们需要对客户端的HTTPS的域名校验部分进行改造，参见：http://blog.csdn.net/github_34613936/article/details/51490032 。

经过域名合并加上IP直连方案改造后，HTTP短连的端到端成功率从95%提升到97.5%，网络延时从1500毫秒降低到了1000毫秒，可谓小投入大产出。

接下来要想进一步提升端到端成功率，就要开始进行长连通道建设了。

提到长连通道建设，首先让人想到的应该是HTTP/2技术。它具有异步连接多路复用、头部压缩、请求响应管线化等众多优点。

如果查看HTTP/2的拓扑结构，其实非常简单：

6

HTTP/2在客户端与服务器之间建立长连通道，将同一域名的请求都放在长连通道上进行。这种拓扑结构有如下一些缺点： 1. 请求基于DNS，仍将面临DNS劫持风险。 2. 不同域名的请求需要建立多条连接。 3. 网络通道难以优化。客户端与服务器之间是公网链路。如果在多地部署服务器，成本消耗又会很大。 4. 业务改造难度大。部署HTTP/2，需要对业务服务器进行改造，而且使用的业务服务器越多，需要改造的成本也越大。 5. 网络协议可订制程度小。

与HTTP/2相区别，我们这里推荐另一种代理长连的模式。这种模式的拓扑图如下：

7

基本思路为：在客户端与业务服务器之间架设代理长连服务器，客户端与代理服务器建立TCP长连通道，客户端的HTTP请求被转换为了TCP通道上的二进制数据包。代理服务器负责与业务服务器进行HTTP请求，请求的结果通过长连通道送回客户端。

与HTTP/2模式对比，代理长连模式具有下面一些优势： 1. 对DNS无依赖。客户端与代理服务器之间的长连通道是通过IP建立的，与DNS没有关系。客户端的HTTP请求被转换为二进制数据流送到代理服务器，也不需要进行DNS解析。代理服务器转发请求到业务服务器时，都处于同一内网，因此可以自己搭建DNS服务，减少对公网DNS服务的依赖。从这个层面上说，代理长连模式天生具有防DNS劫持的能力。 2. 不同域名的请求可以复用同一条长连通道。 3. 通道易优化。与部署业务服务器相比，部署代理长连服务器的代价就小了很多，可以在全国甚至全世界多地部署代理长连服务器。客户端在选择代理长连服务器时，可以通过跑马找到最快的服务器IP进行连接。另一方面，代理服务器与业务服务器之间的网络通道也可以进行优化，通过架设专线或者租用腾讯云等方式可以大大提升通道服务质量。 4. 对业务完全透明。客户端的业务代码只要接入网络层的SDK即可，完全不用关心网络请求使用的是长连通道还是短连通道。代理服务器将客户端的请求还原为HTTP短连方式送到业务服务器，业务服务器不需要进行任何改造。 5. 网络协议完全自定义。

在长连通道项目的早期，出于快速推进的目的，同时受限于建设代理长连服务器需要投入大量资金，我们首先接入使用了腾讯的维纳斯（WNS）服务（官网地址：https://www.qcloud.com/product/wns ）。

8

WNS服务采用的也是代理长连模式，依托腾讯云的强大硬件建设，我们使用下来发现端到端成功率可以达到99.6%以上。（PS：这里的提到的端到端成功率与官网宣传的99.9%不同是由于统计口径的不同。）

由于腾讯WNS服务是面向公众的云服务，服务的客户远不止一家，无法完全满足我们公司技术需求的快速变更，因此还是需要进行自己的长连通道项目建设。

自建长连建设大概可以分为以下几个周期：

① 中转服务的开发和部署

9

作为开发的初级阶段，这一时期的任务主要是搭建代理中转服务器，并架设完整链路结构。

② 加密通道的建设

10

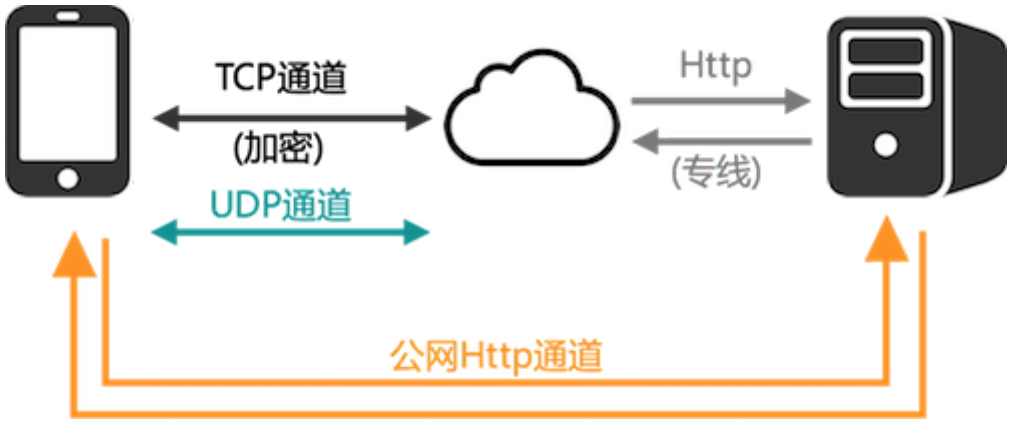
为了保护TCP通道上数据的安全性，客户端与代理长连服务器之间的二进制通信数据可以利用加密来保障数据安全。

③ 专线建设

11

在代理长连服务器与后台业务服务器之间建设专线。使用专线，可以大大降低公网环境的干扰，保障服务的稳定性。

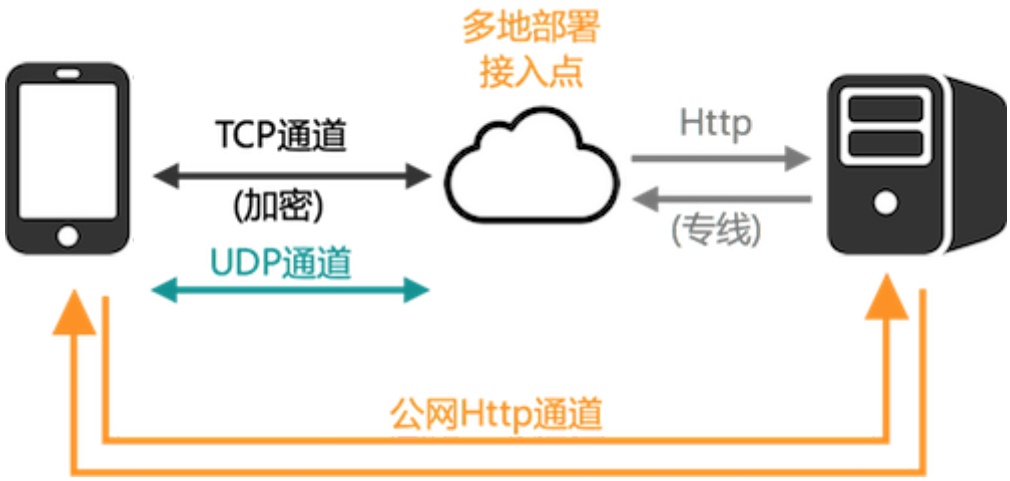
④ 自动降级Failover建设



12

由于客户端的请求都放在TCP通道上进行，当代理长连服务器需要升级或者由于极端情况发生了故障时，将会造成客户端的整体网络服务不可用。为了解决这个问题，我们准备了Failover降级方案。当TCP通道无法建立或者发生故障时，可以使用UDP面向无连接的特性提供另一条请求通道，或者绕过代理长连服务器之间向业务服务器发起HTTP公网请求。本文的后面章节有展示Failover机制的实际效果。

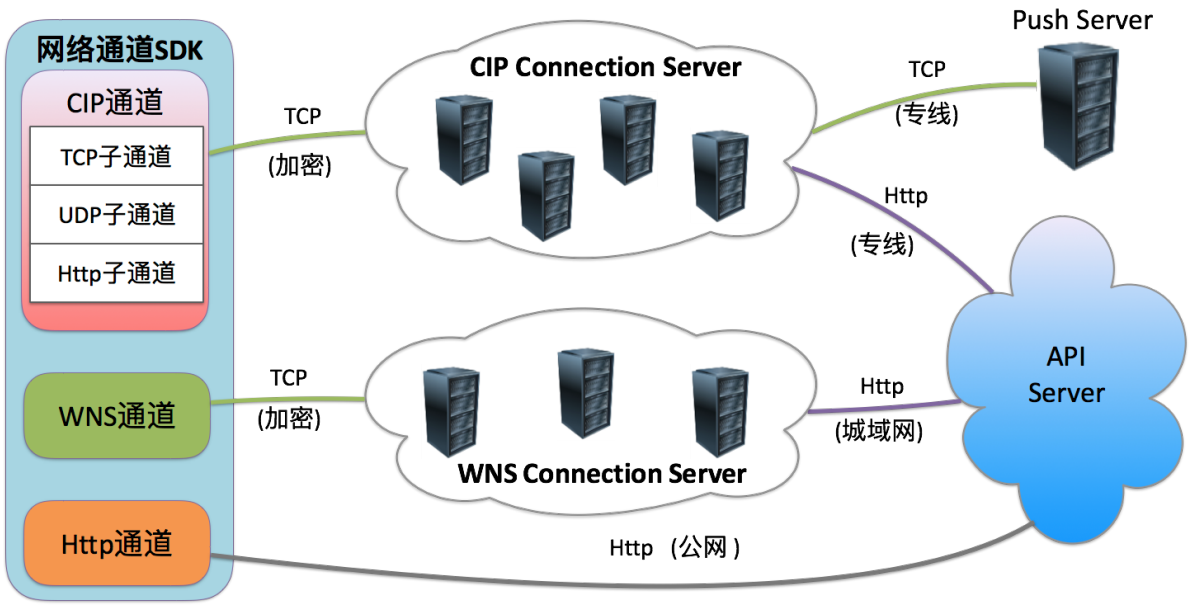
⑤ 多地部署接入点



13

在全国多地部署代理长连接入点。客户端与接入点建立长连通道时，可以选择最快的服务器就近接入，从而大大降低通道连接速度并提升通信质量。 我们在近两年的网络优化实践中，将客户端的网络通道服务整理成了一个独立的SDK，SDK内除了包含了自建的长连通信服务，也包含了WNS等网络通道。

完整的网络通道拓扑图如下所示：



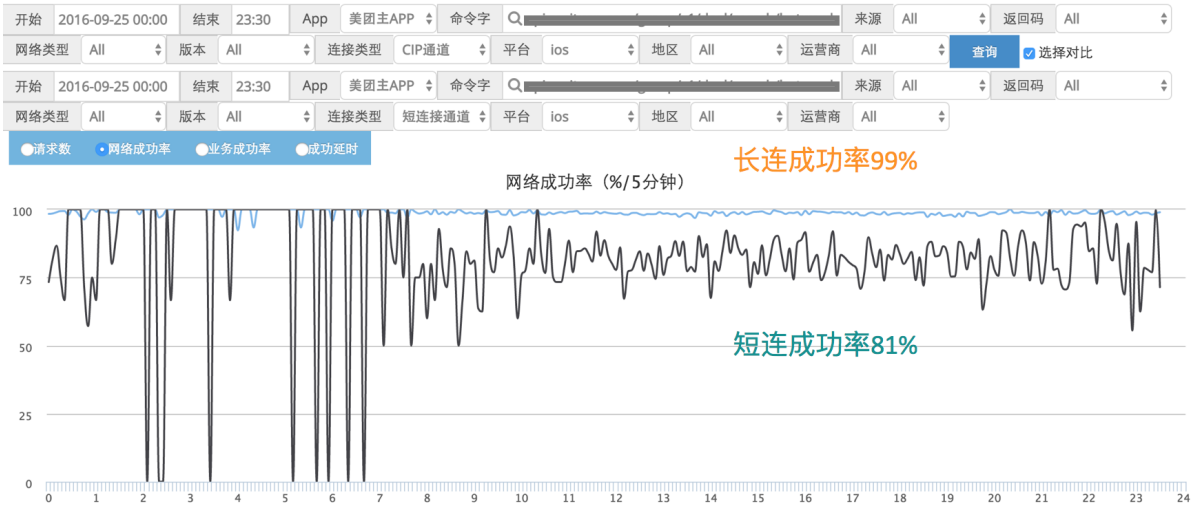
14

图中网络通道SDK包含了三大通信通道： 1. CIP通道：CIP通道就是上文中提到的自建代理长连通道。CIP是China Internet Plus的缩写，为网络优化集团的注册英文名称。App中绝大部分的请求通过CIP通道中的TCP子通道与长连服务器（CIP Connection Server）通信，长连服务器将收到的请求代理转发到业务服务器（API Server）。由于TCP子通道在一些极端情况下可能会无法工作，我们在CIP通道中额外部署了UDP子通道和HTTP子通道，其中HTTP子通道通过公网绕过长连服务器与业务服务器进行直接请求。CIP通道的平均端到端成功率目前已达99.7%，耗时平均在350毫秒左右。 2. WNS通道：出于灾备的需要，腾讯的WNS目前仍被包含在网络通道SDK中。当极端情况发生，CIP通道不可用时，WNS通道还可以作为备用的长连替代方案。 3. HTTP通道：此处的HTTP通道是在公网直接请求API Server的网络通道。出于长连通道重要性的考虑，上传和下载大数据包的请求如果放在长连上进行都有可能导致长连通道的拥堵，因此我们将CDN访问、文件上传和频繁的日志上报等放在公网利用HTTP短连进行请求，同时也减轻代理长连服务器的负担。

> 推送方案：在网络通道拓扑图的右上角，有个Push Server。它是考虑到TCP通道的双工特性，为网络通道SDK提供推送的能力。利用通知推送，可以在服务器数据发生变化时及时通知客户端。推送方案可以替换掉代码中常见的耗时低效的轮询方案。

案例展示

下图展示了某开机接口在接入长连后的端到端成功率对比：



15

上图中黑色曲线是某开机接口在短连通道下的成功率曲线。成功率平均只有81%，抖动的特别剧烈，说明网络服务稳定性不够。

蓝色曲线是同一接口在长连通道下的成功率曲线。成功率平均已达到99%，抖动大幅减小。

成功延时对比图：

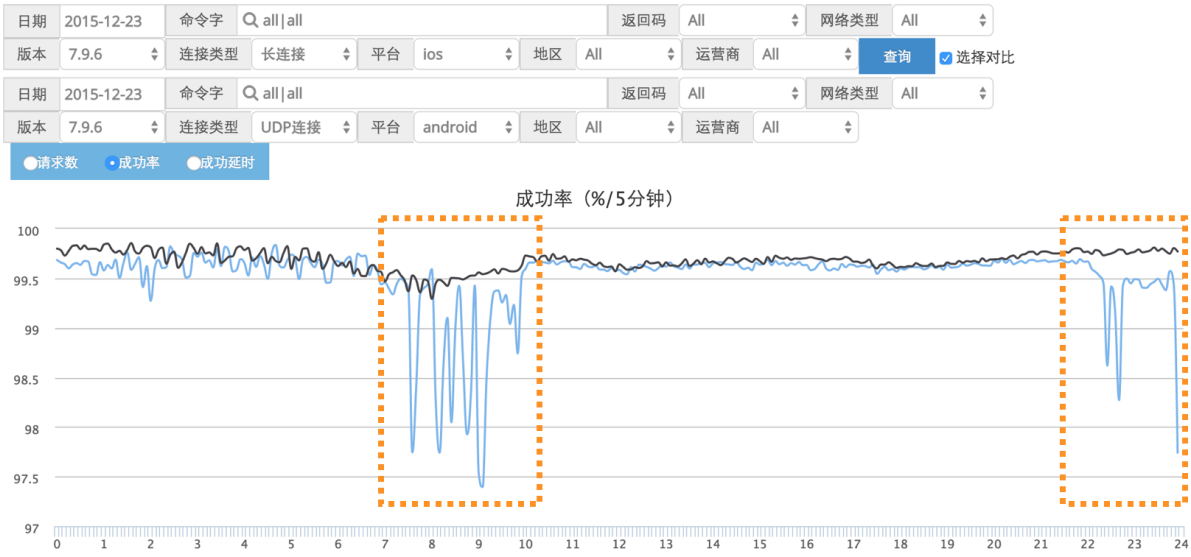
16

上图中展示了同样情况下的成功延时曲线。蓝色线为长连延时曲线，黑色线为短连延时曲线。

接下来我们看Failover的效果展示图。

下图展示了2015年的一次长连服务器故障。

当时Android客户端采用了Failover方案，在长连不可用时Failover到短链或者UDP通道上。与未采用Failover方案的iOS客户端相比，Failover机制在维持网络整体可用性方面体现出了非常大的优势。



17

网络配置系统

网络通道SDK包含了CIP|WNS|HTTP三大通道，不同的通道具有各自的优缺点，控制各请求选择合适的网络通道成了迫在眉睫的重要课题。

为此我们开发了网络配置系统，通过下发指令，调整App中网络通道SDK中的通道选择策略，可以控制不同的API请求动态切换网络通道。

下图是某接口的线上通道切换示意图：

类别	成功率(%)	总请求数	成功平均延迟(ms)	平均发包(B)	平均回包(B)
当前值	99.552%	173,785	1,824.492	0	988
对比值	96.901%	130,919	816.690	0	996

18

图中展示了某接口切换WNS通道的过程。图中的黑色线代表短连通道下的请求数量曲线，蓝色线代表WNS通道下的请求数量曲线。通过线上控制系统下发了通道切换指令后，绝大部分的短连请求在5分钟之内被切换成了WNS通道请求。

在客户端开发过程中，我们发现： * 长连通道建立越早，成功率越高。长连通道越早建立，越多的请求能够在长连通道上进行。特别是当App刚打开时，数量众多的请求同时需要发出。面对这种情况，我们采取的策略是首先建立长连通道，将众多请求放入等待发送队列中，待长连通道建立完毕后再将等待队列中的请求放在长连通道上依次送出。采用这种策略后，我们发现启动时的接口成功率平均提升了1.4%，延时平均降低了160毫秒。 * TCP数据包越大，传输时间越长。如果长连通道未采用类似HTTP/2中的数据切片技术，大的数据包非常容易导致长连通道的堵塞。 * 底层SDK上线新功能一定要有线上降级手段。当新功能上线了发生故障时，可以通过开关或参数控制，或是采用ABTest方式等进行降级，防止故障扩大化。 * iOS和Android系统网络库存在很多默认行为。例如系统网络库会在内部处理网络重定向，再比如请求头中如果没有填写Accept-Encoding或Content-Type等字段，系统网络库会自动填写默认值。 * 一个容易忽视的地方：HTTP的请求头键值对中的的键是允许相同和重复的。例如下图所示的”Set-Cookie”字段就是包含了多组的相同的键名称。与之类似的还有”Cookie”字段。在长连通信中，如果对header中的键值对用不加处理的字典方式保存和传输，就会造成数据的丢失。

```
HTTP/1.1 200 OK
Date: Tue, 27 Dec 2016 08:00:37 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: Keep-Alive
Vary: Accept-Encoding
Cache-Control: private
Cxy_all: baidu+974f270685250a6065fa7b5b8ad8dda7
Expires: Tue, 27 Dec 2016 07:59:48 GMT
X-Powered-By: HPHP
Server: BWS/1.1
X-UA-Compatible: IE=Edge,chrome=1
BDPAGETYPE: 1
BDQID: 0x960326970002436d
BDUSERID: 0
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=1422_21084_18133_17001_21554_21672_20929; path=/; domain=.baidu.com
Content-Encoding: gzip

<!DOCTYPE html>
<!--STATUS OK-->
```


对于正在成长中的创业公司，我们有如下改善网络状况的建议： * 收拢网络底层。随着公司的成长，开发团队越来越多，不可避免的将会引入越来越多的网络库。网络库多了之后，再对网络请求进行集中管理就非常困难了。我们的建议是在网络库与业务代码之间架设自己的网络层，业务的网络请求全部经过网络层代码进行请求。这样未来进行底层网络库的更换，或者网络通道的优化将变得容易很多。 * 使用网络监控。引入网络监控机制，发现网络问题。这里推荐我公司开发的开源的Cat监控系统。Cat开源地址为http://github.com/dianping/cat 。 * 尝试进行短连优化。前文中提到的域名合并和IP直连方案都是简单有效的手段。 * 可以尝试HTTP/2或腾讯WNS长连服务。