QCon 全球软件开发大会
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

BEIJING 2018

《QUIC在手机微博中的应用实践》

聂永 @ 微博产品部

主办方 Geekbang. 极客邦科技 InfoQ

目录

一、起因

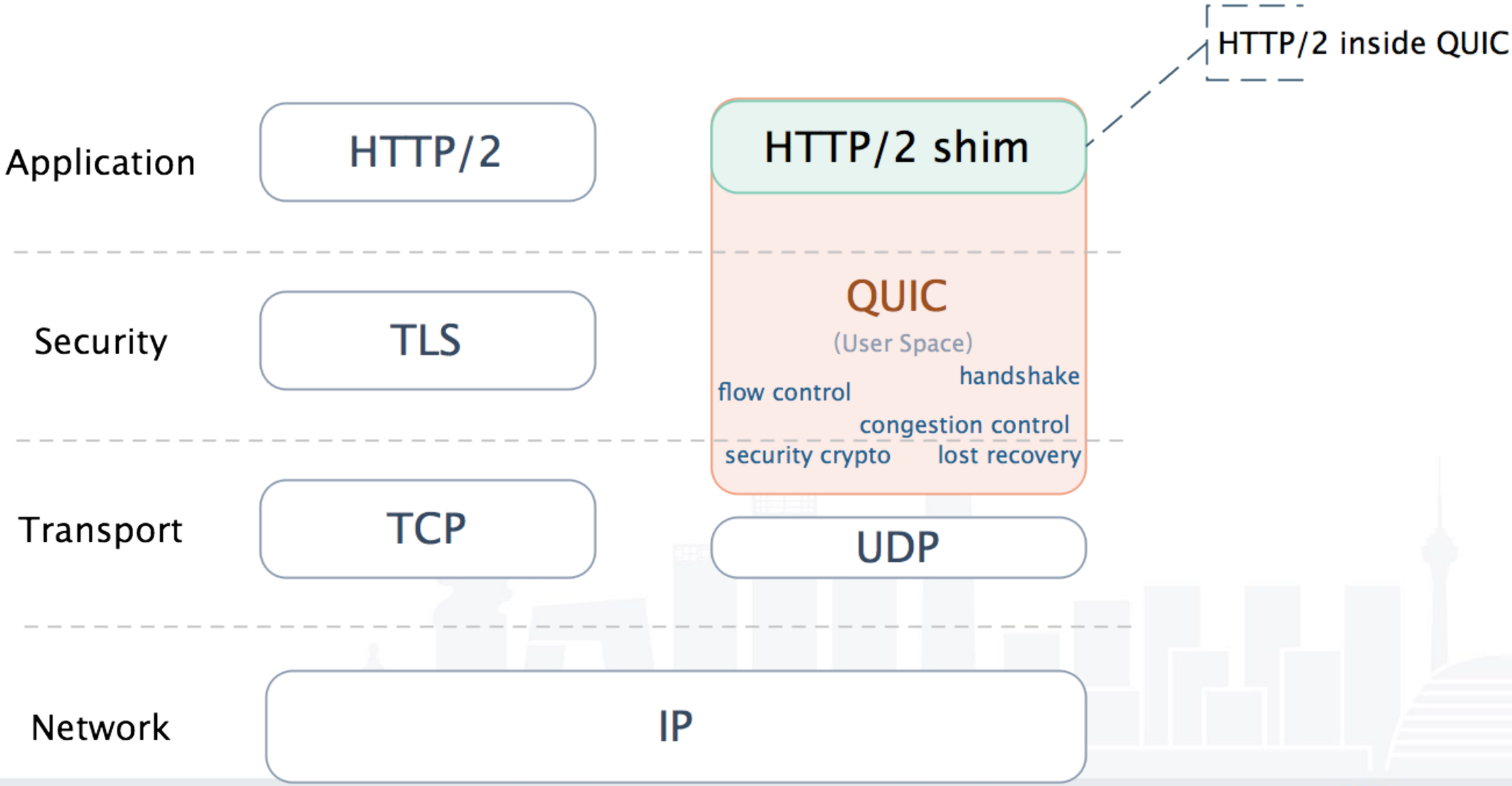
1.1 出发点

- 让微博APP在初始连接时在网络链路层面请求后端业务时能够再快一点
- 在弱网环境下尽可能的满足用户基本请求
- 在复杂多变的网络环境下有更多的链路通道选择，增加后端业务的可用性

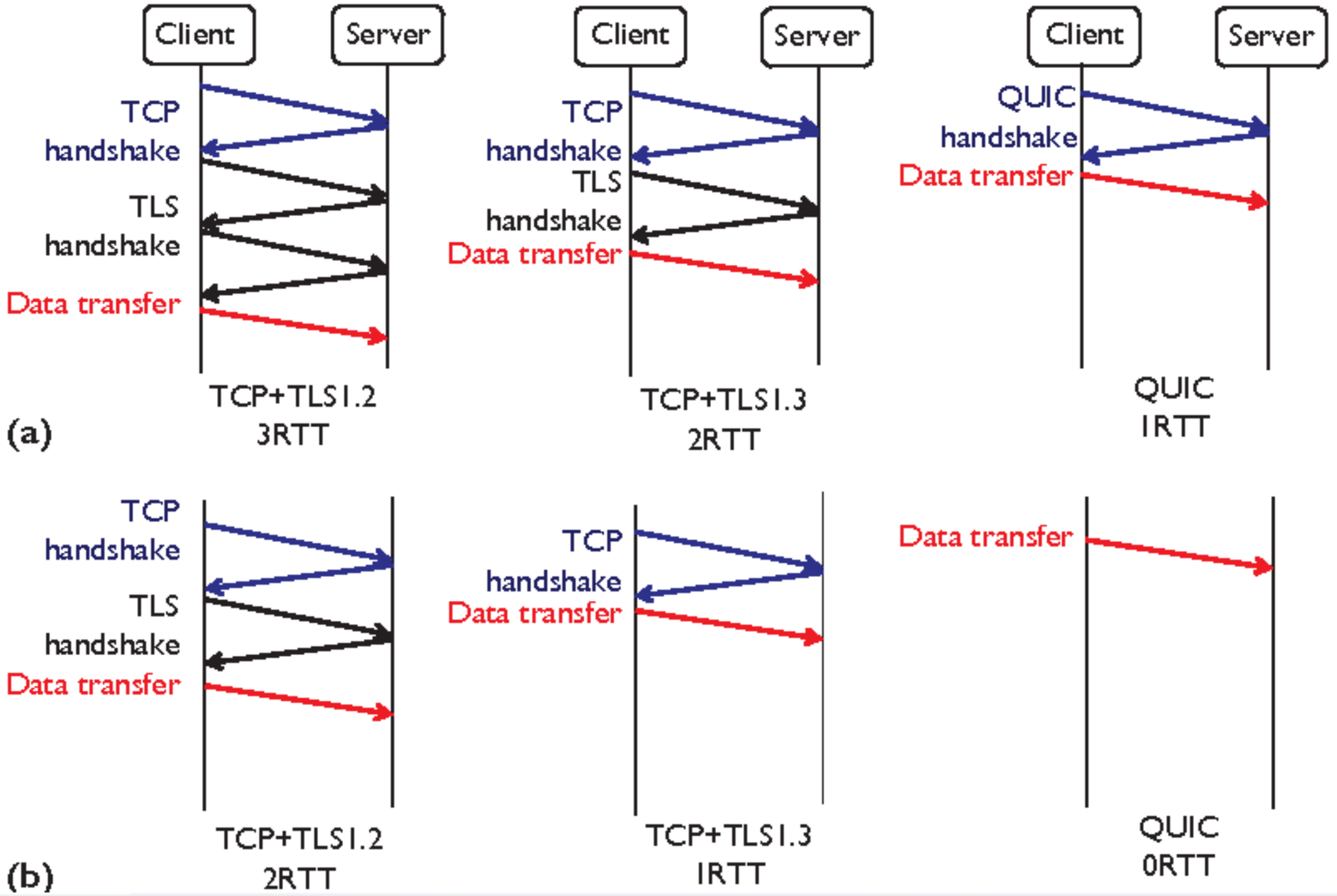
1.2 HTTPS 存在问题

- HTTPS （1.1 & 2.0） Over TCP with TLS
- 完整握手流程过长（基于TLS 1.2）
  - 首次连接 >= 3-RTT
  - 协议握手缓存后： 2-RTT
- 弱网环境，TCP Head-of-Line blocking + TLS Record HOF = 双重阻塞
  - 无法实现真正多路复用
- TCP被嵌入系统内核层，遗留网络设备僵化
- ……

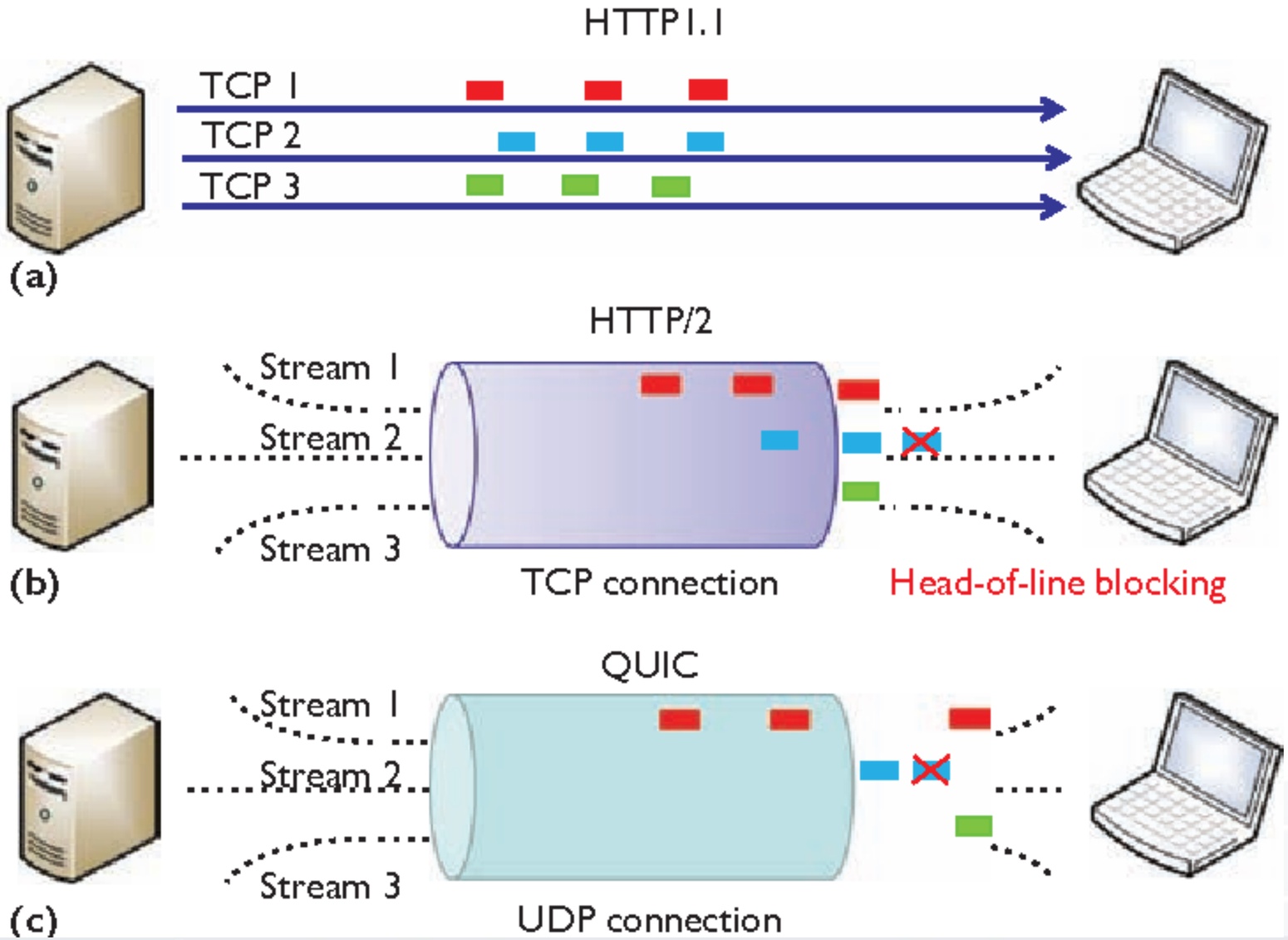
1.3 QUIC，致力于成为下一代的互联网传输协议



1.4 QUIC，握手流程缩短



1.5 QUIC，真正的多路复用



1.6 QUIC其它特性

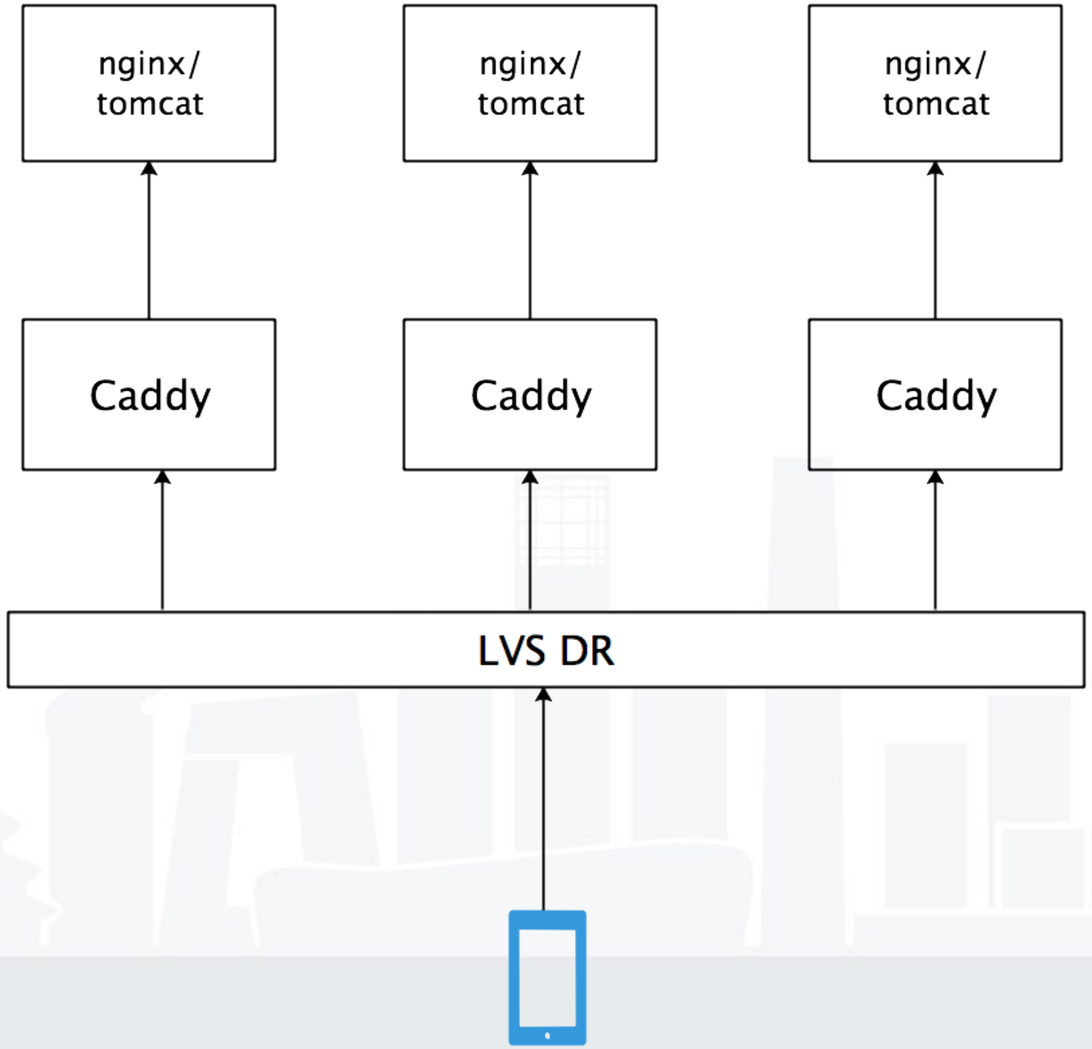
- 改进的、可选的拥塞控制机制
  - Client：BBR，Server：Cubic
- 前向冗余纠错
- 连接迁移
- ……

二、服务端实践过程

2.1 服务端技术选项

- go-quic
  - 不活跃
- google quic server demo
  - 示范Demo存在
  - 缺少线上生产级特性支持
- ……
- Caddy + Quic
  - 提供一站式网络堆栈服务
  - 比较成熟
  - 使用者众多，更新频繁

2.2 部署结构


nginx/
tomcat
nginx/
tomcat
nginx/
tomcat
Caddy
Caddy
Caddy
LVS DR

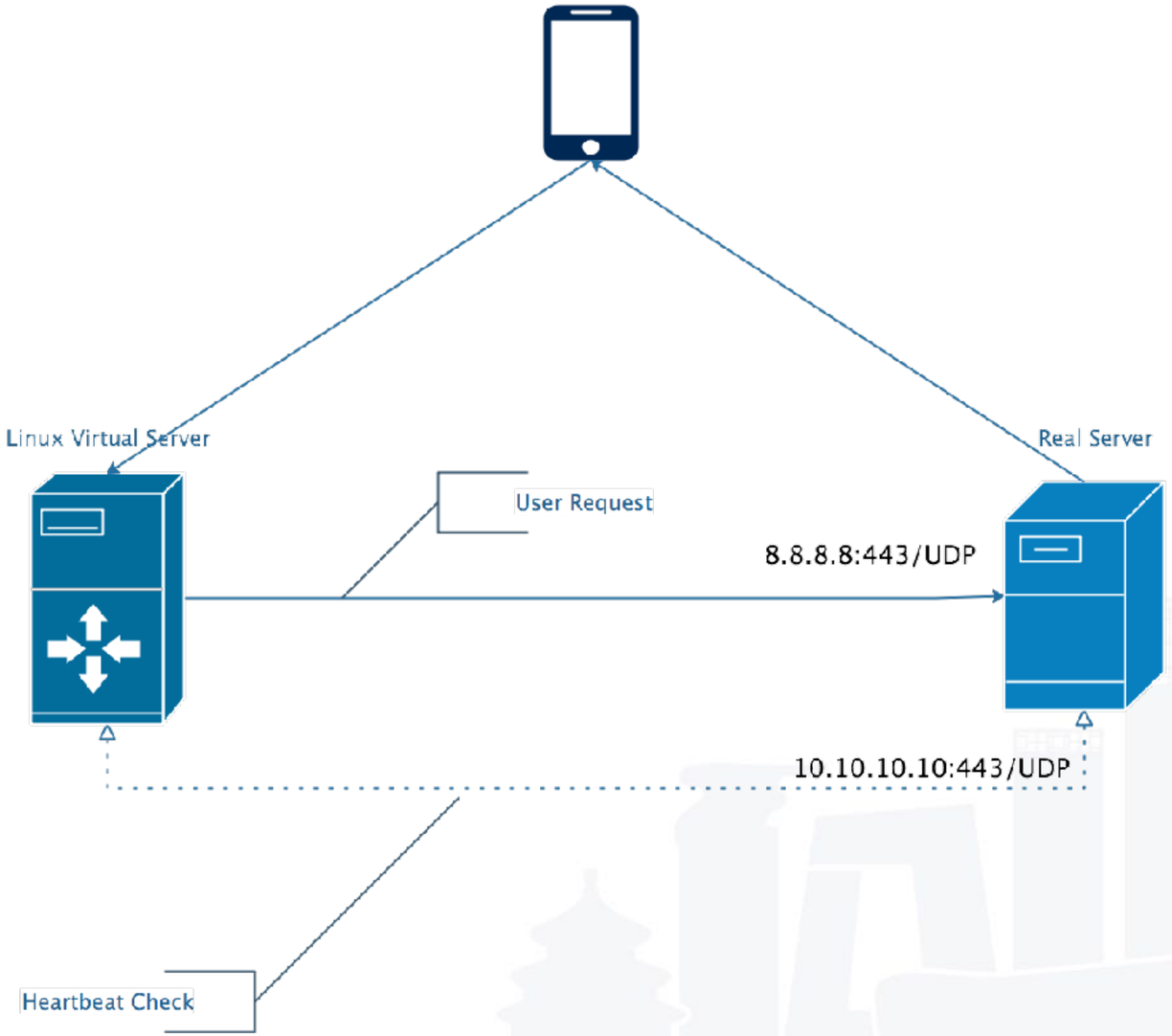
2.3 LVS DR针对UDP支持不够

- 场景：LVS DR模式
- TCP 泛监听，比如：*:8080，LVS DR 运行正常
  - 本机所有 IP上都有监听TCP 8080，外网客户端正常收包&发包
- UDP泛监听，比如：*:8443，LVS DR 表明上看运行正常
  - UDP Server回包时，IP数据包中的Src源地址为内网IP
  - 外网客户端只能发包，但收不到数据包

2.3.1 UDP在LVS-DR下需要显式绑定

- 假设VIP：8.8.8.8，内网RS IP：10.10.10.10
- Caddy需要显式绑定：VIP:Port
  - eg：8.8.8.8:443
- 这样做就够了吗？

2.3.2 LVS需要心跳检测



- LVS健康检测判断RS实例是否存活
- 需要绑定内网IP：Port
  - Caddy已绑定VIP：Port，eg：8.8.8.8:443
  - 同一个Caddy实例没办法重复绑定10.10.10.10:443
- 实际怎么操作？

2.3.3 应对LVS心跳检测

- 方式不一而足
- 比如可以借助于netcat
  - eg：ncat -u -l 10.10.10.10 443 -k -c 'echo “hello”'
- 进阶：supervisor + ncat

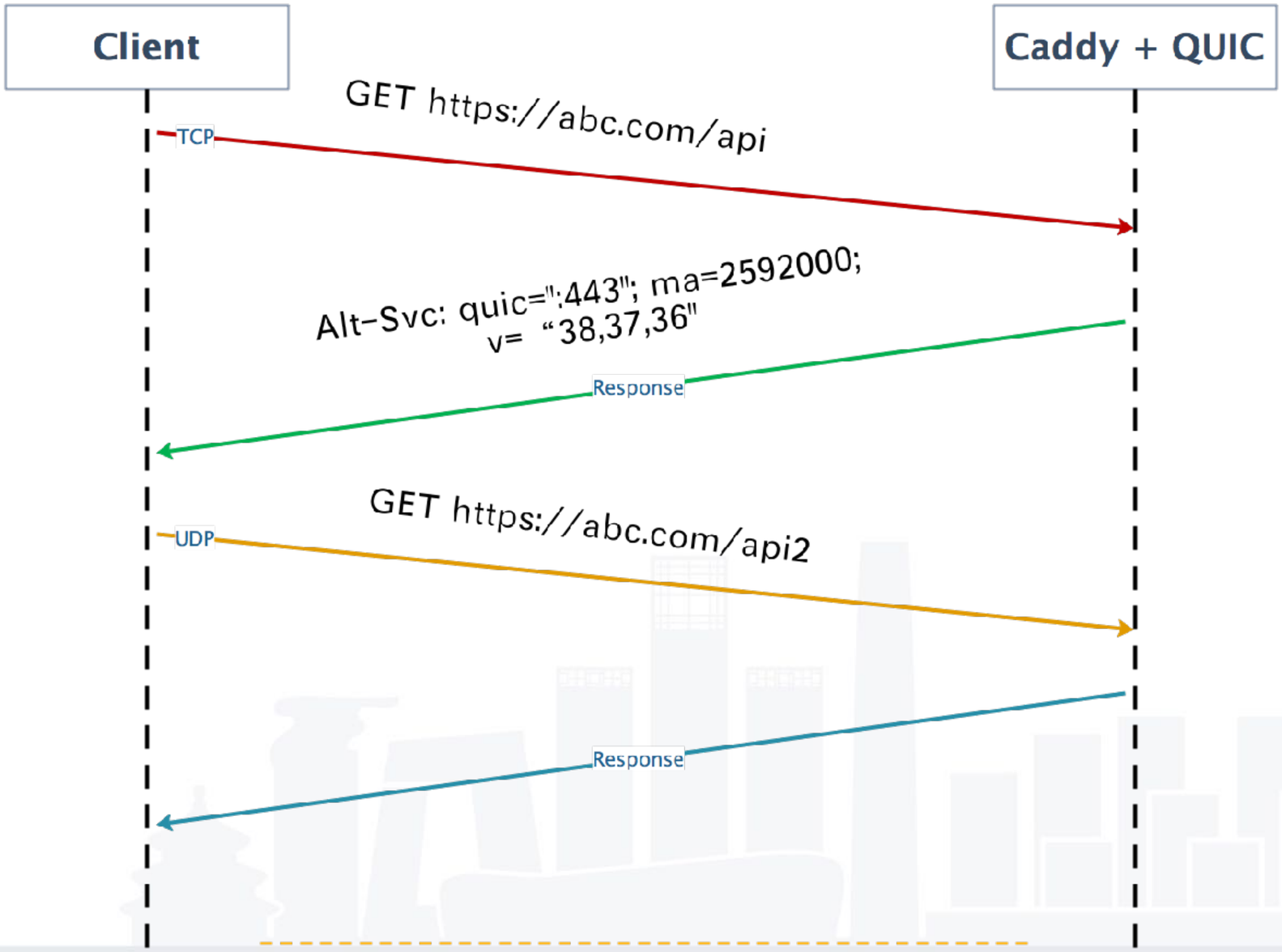
```
[root@75-29-105-yf-core supervisord.conf.d]# cat ncat.conf
[program:ncat]
command = ncat -ukl 10.75.29.105 443 -c 'echo "success"'
autostart = true
startsecs = 10
autorestart = true
startretries = 3
stopsignal = TERM
stopwaitsecs = 10
user = root
```

2.3.4 LVS-DR UDP端口绑定小结

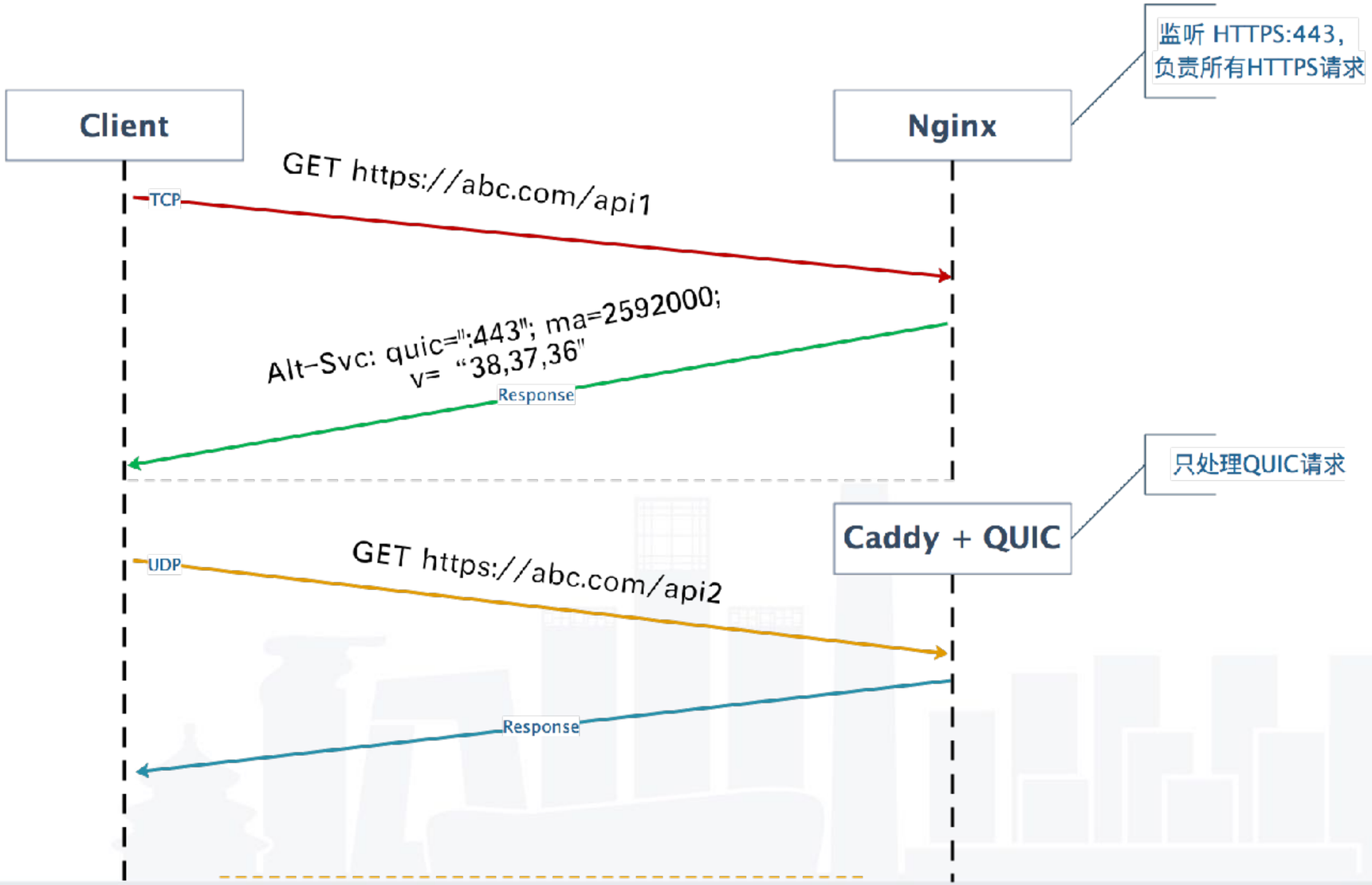
- Caddy bind：VIP:Port （eg：8.8.8.8:443)
- ncat bind：RS_IP:Port（eg:10.10.10.10:443)

2.4 nginx & caddy的协作

2.4.1 QUIC Discovery



2.4.2 Nginx & Caddy协作



2.4.3 Nginx & Caddy端口不冲突

- Caddy监听 444 端口，UDP：444，TCP：444
- Nginx监听TLS 443端口：
  - add_header Alt-Svc 'quic=":444"; ma=2592000; v="38,37,36"';
- OpenResty监听TLS 443端口：
  - ngx.header["Alt-Svc"] = 'quic=":444"; ma=2592000; v="38,37,36"'

2.4.4 Nginx + Caddy混跑都监听443端口

- 实际情况：Nginx TCP 443 + Caddy UDP 443
- Caddy’s TCP替换成一个非常规端口
  - udpAddr, err := net.ResolveUDPAddr("udp", strings.Replace(s.Server.Addr, ":45678", ":443", 1))
  - Caddy TCP 45678，UDP 443
- OpenResty
  - ngx.header["Alt-Svc"] = 'quic=":443"; ma=2592000; v=“38,37,36”’
- nginx
  - add_header Alt-Svc 'quic=":444"; ma=2592000; v=“38,37,36"';

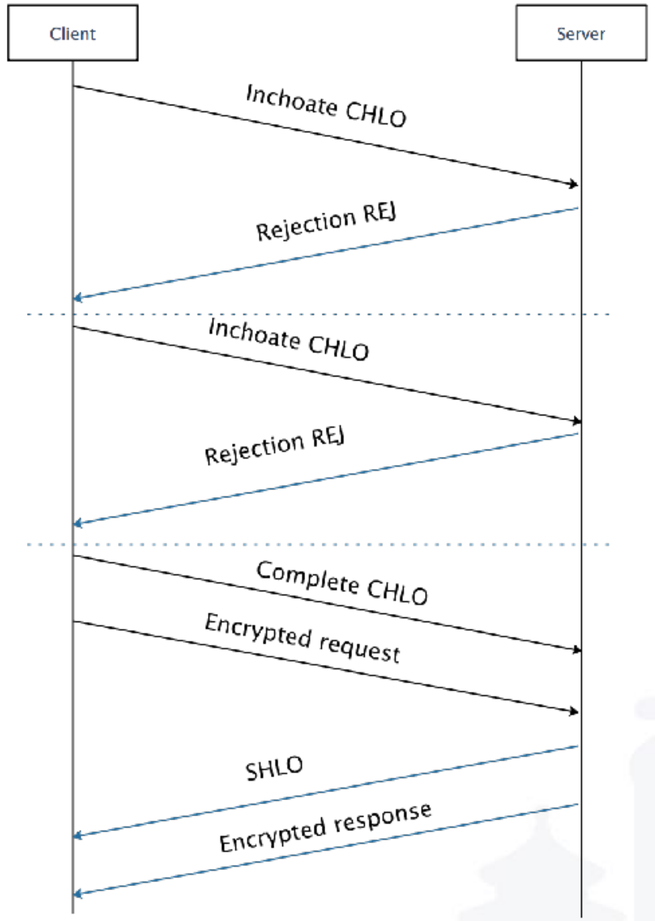
2.4.5 Nginx & Caddy小结

- nginx负责HTTPS，caddy负责QUIC，共同协作
- 在都占用443端口情况下
  - nginx & caddy不建议混跑
  - nginx & caddy 错开部署，LVS指向RS不同
- 端口不冲突，可以选择混合或错开部署

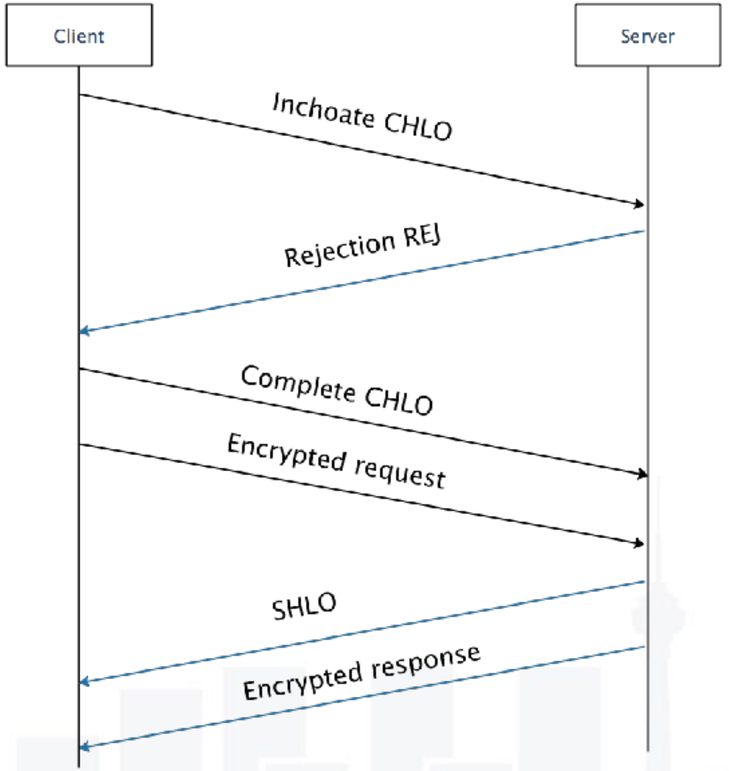
2.5 QUIC握手流程，现实和理想差距

No.	Time	Source	Protoc	Destination	Length	Info
1	20:14:40.826608	172.17.0.2	QUIC	10.222.68.73	1113	Client Hello, PKN: 1, CID: 14610013736658355872
2	20:14:40.827571	10.222.68.73	QUIC	172.17.0.2	73	Payload (Encrypted), PKN: 1, CID: 14610013736658355872
3	20:14:40.827668	10.222.68.73	QUIC	172.17.0.2	298	Rejection, PKN: 2, CID: 14610013736658355872
4	20:14:40.827896	172.17.0.2	QUIC	10.222.68.73	80	Payload (Encrypted), PKN: 2, CID: 14610013736658355872
5	20:14:40.828562	172.17.0.2	QUIC	10.222.68.73	1115	Client Hello, PKN: 3, CID: 14610013736658355872
6	20:14:40.828917	10.222.68.73	QUIC	172.17.0.2	73	Payload (Encrypted), PKN: 3, CID: 14610013736658355872
7	20:14:40.832438	10.222.68.73	QUIC	172.17.0.2	1344	Rejection, PKN: 4, CID: 14610013736658355872
8	20:14:40.832659	172.17.0.2	QUIC	10.222.68.73	80	Payload (Encrypted), PKN: 4, CID: 14610013736658355872
9	20:14:40.832727	10.222.68.73	QUIC	172.17.0.2	668	Payload (Encrypted), PKN: 5, CID: 14610013736658355872
10	20:14:40.859380	172.17.0.2	QUIC	10.222.68.73	77	Payload (Encrypted), PKN: 5, CID: 14610013736658355872
11	20:14:40.860443	10.222.68.73	QUIC	172.17.0.2	76	Payload (Encrypted), PKN: 6, CID: 14610013736658355872
12	20:14:40.868048	172.17.0.2	QUIC	10.222.68.73	1115	Client Hello, PKN: 6, CID: 14610013736658355872
13	20:14:40.869204	10.222.68.73	QUIC	172.17.0.2	267	Payload (Encrypted), PKN: 7
14	20:14:40.869563	172.17.0.2	QUIC	10.222.68.73	80	Payload (Encrypted), PKN: 7, CID: 14610013736658355872
15	20:14:40.870079	172.17.0.2	QUIC	10.222.68.73	122	Payload (Encrypted), PKN: 8, CID: 14610013736658355872
16	20:14:40.870207	172.17.0.2	QUIC	10.222.68.73	69	Payload (Encrypted), PKN: 9, CID: 14610013736658355872
17	20:14:40.870644	10.222.68.73	QUIC	172.17.0.2	68	Payload (Encrypted), PKN: 8
18	20:14:40.877862	10.222.68.73	QUIC	172.17.0.2	166	Payload (Encrypted), PKN: 9
19	20:14:40.877889	10.222.68.73	QUIC	172.17.0.2	77	Payload (Encrypted), PKN: 10
20	20:14:40.877908	10.222.68.73	QUIC	172.17.0.2	80	Payload (Encrypted), PKN: 11
21	20:14:40.878531	172.17.0.2	QUIC	10.222.68.73	76	Payload (Encrypted), PKN: 10, CID: 14610013736658355872
22	20:14:40.896887	10.222.68.73	QUIC	172.17.0.2	68	Payload (Encrypted), PKN: 12

2.5.1 握手优化 2-RTT -> 1-RTT



Initial 2-RTT Handshake



Initial 1-RTT Handshake

第一次握手时放宽条件限制

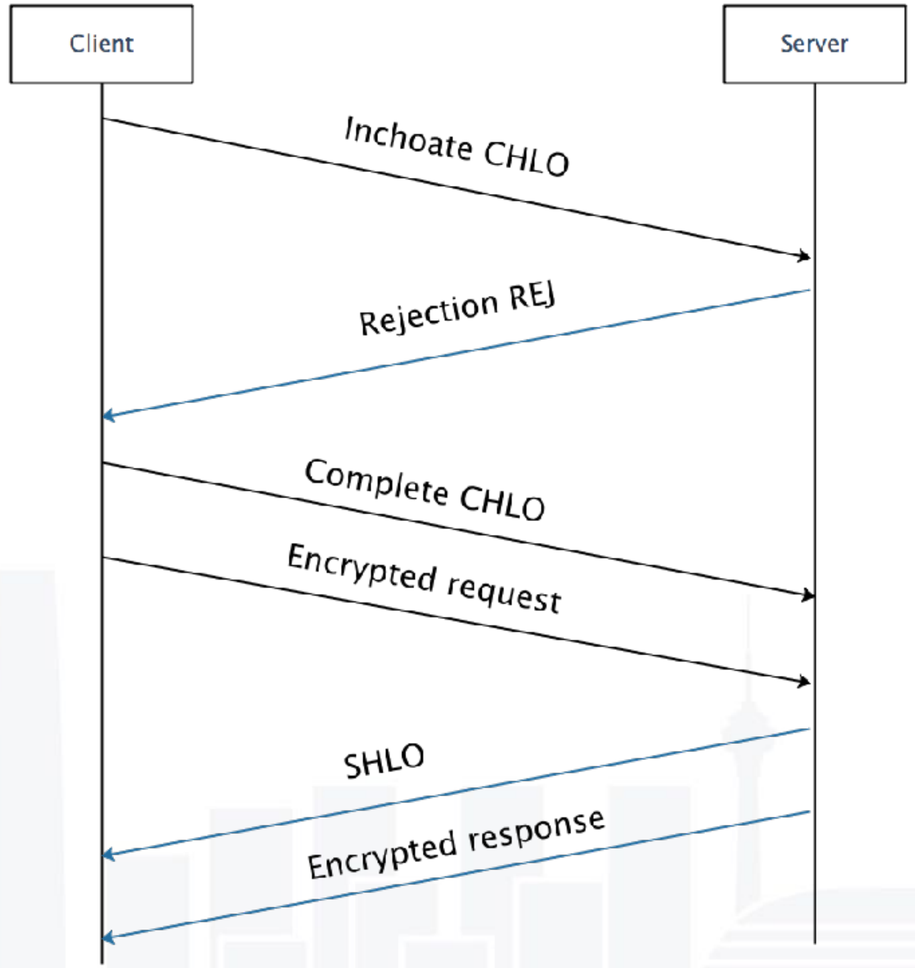
```
if h.acceptSTK(cryptoData[TagSTK]) {
    proof, err := h.scfg.Sign(sni, chlo)
    if err != nil {
        return nil, err
    }

    commonSetHashes := cryptoData[TagCCS]
    cachedCertsHashes := cryptoData[TagCCRT]

    certCompressed, err := h.scfg.GetCertsCompressed(sni, commonSetHashes, cachedCertsHashes)
    if err != nil {
        return nil, err
    }
    // Token was valid, send more details
    replyMap[TagPROF] = proof
    replyMap[TagCERT] = certCompressed
}
```

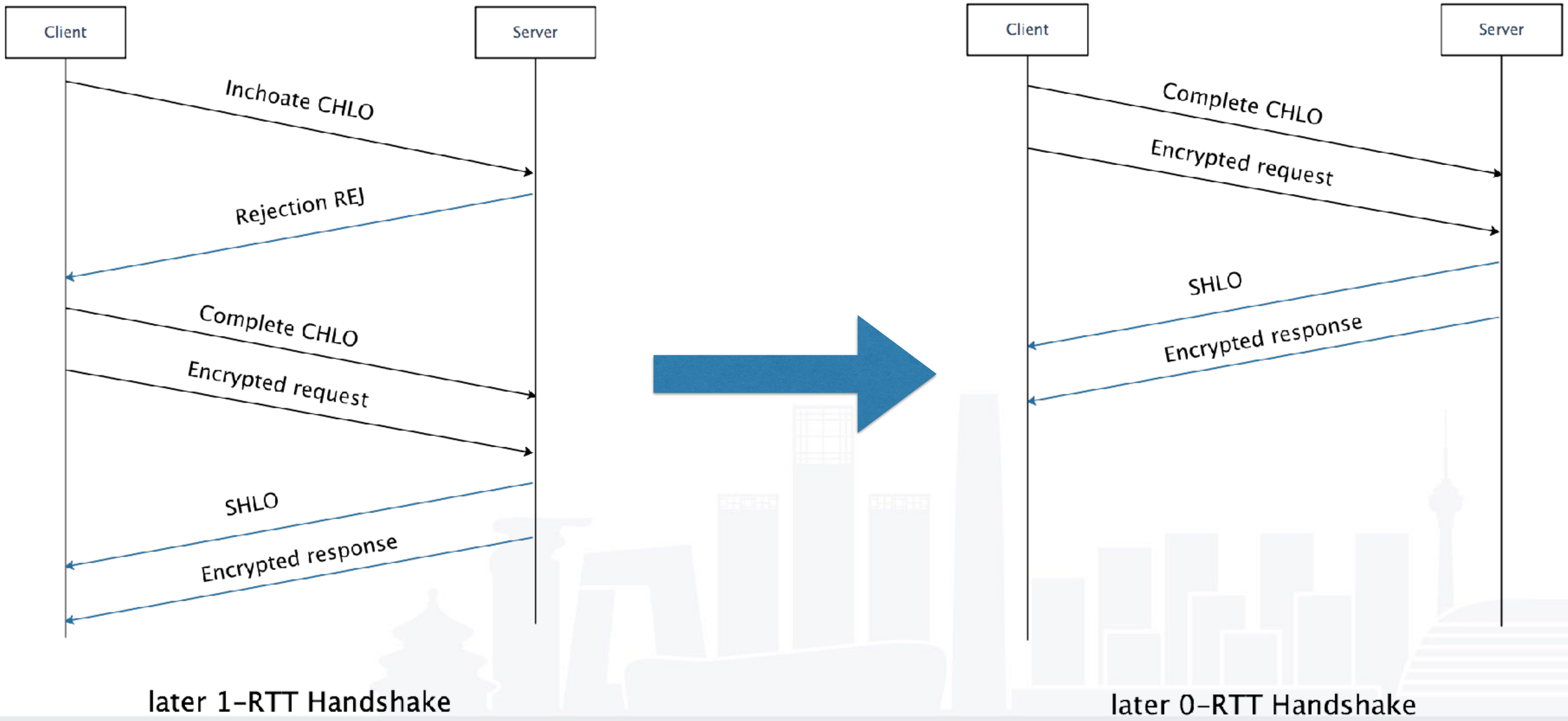
2.5.2 握手优化到此结束了没?

- 场景：在前面优化基础上，针对再次连接用户
- 网络切换导致客户端缓存失效
- 同一机房接入服务实例改变导致缓存失效



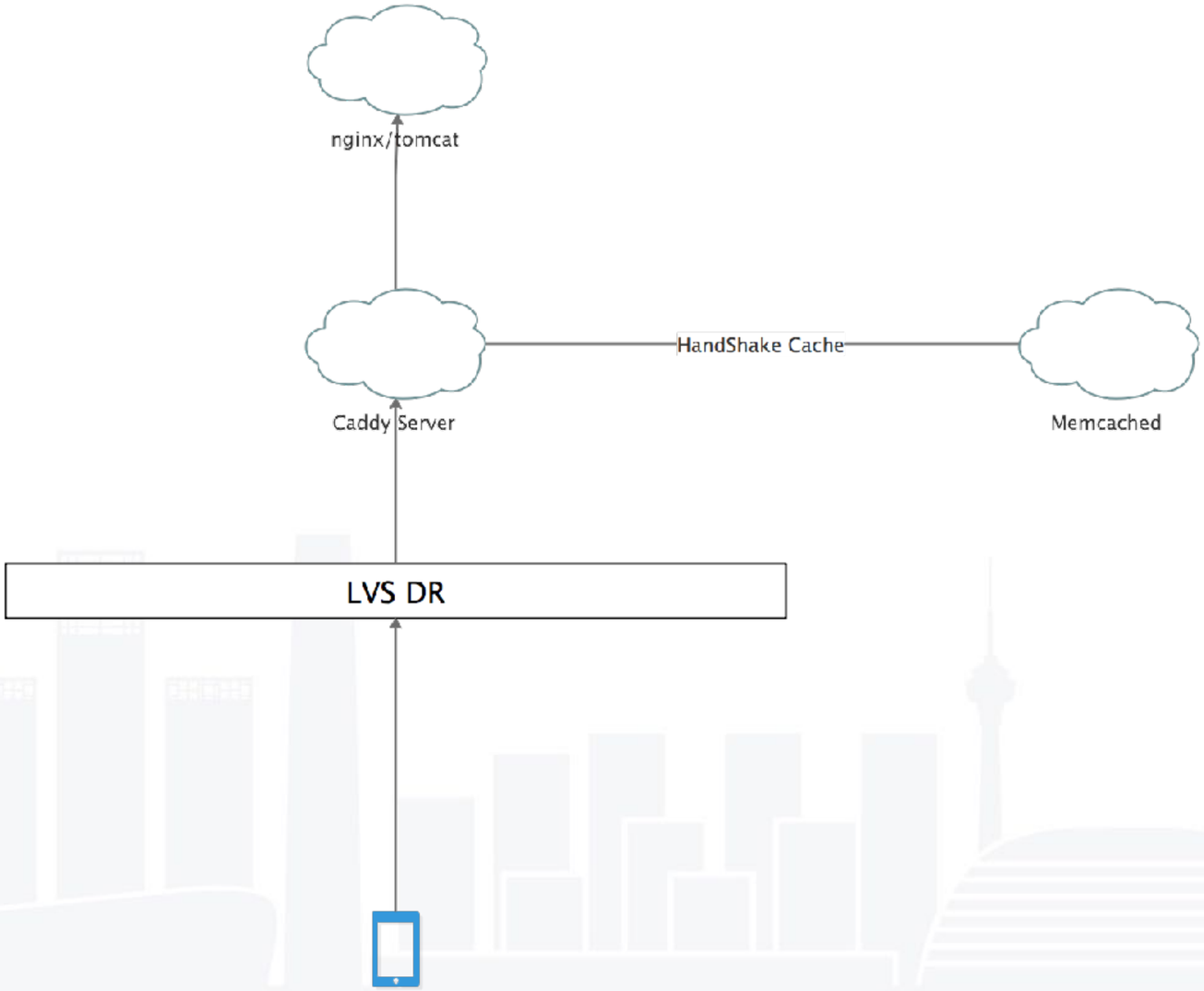
later 1-RTT Handshake

2.5.3 1-RTT -> 0-RTT

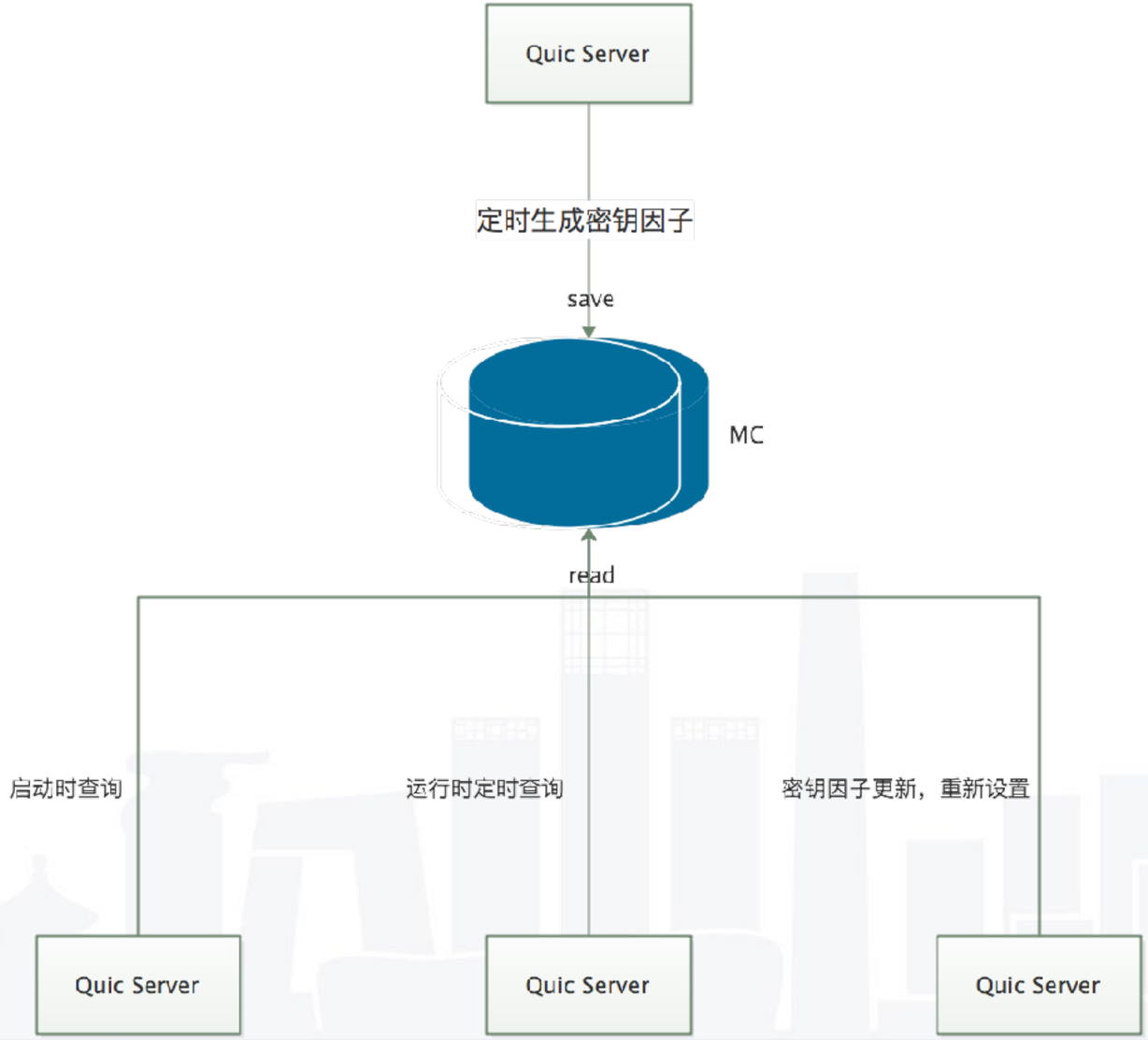


2.5.4 0-RTT，全局级握手缓存

- 针对再次连接用户
- 用户的第一次连接后握手信息保存到MC中
- 用户下次连接落到哪一个机房都会0-RTT握手流程

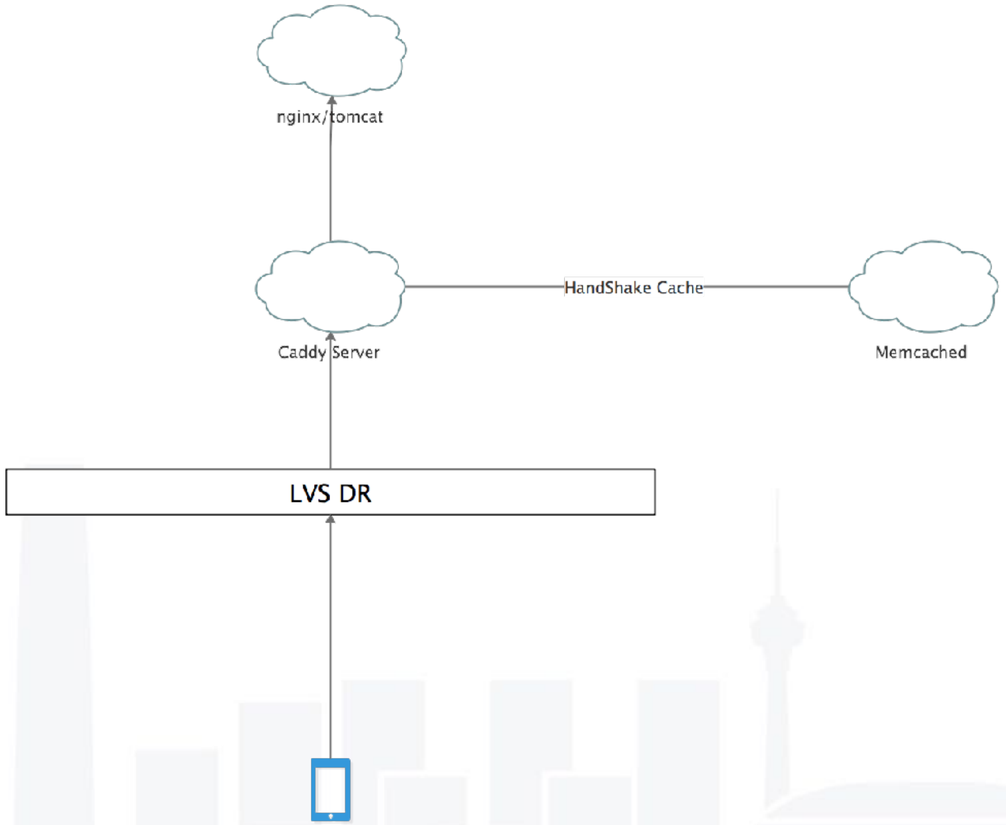


2.5.5 共享密钥机制


Quic Server
定时生成密钥因子
save
MC
read
启动时查询
运行时定时查询
密钥因子更新，重新设置
Quic Server
Quic Server
Quic Server

2.5.6 0-RTT线上数据呈现

- 场景：灰度场景用户活跃度稍低
- 线上 33.09% 用户握手建连：0-RTT
  - 再次为用户节省 6ms-11ms 握手时间
  - 相比优化前2-RTT握手，建连速度提升200%



2.5.7 QUIC握手优化小结

- 针对我们的客户端进行定制
- 客户端初始连接，2-RTT -> 1-RTT
- 缓存握手信息，1-RTT -> 0-RTT

2.6 QUIC Bench

- QUIC Server容量评估
- Support：Q035-Q038
- https://github.com/yongboy/quicbench
- quicbench -u https://abc.com/api -c 100 -r 10

2.7 握手优化前后性能对比

- Go 19.2 + Caddy v0.10.10

场景	优化前	优化后
100用户，每用户10请求	5076.79 hits/sec	5255.15 hits/sec
500用户，每用户10请求	3021.46 hits/sec	4563.99 hits/sec
1000用户，每用户10请求	2645.09 hits/sec （经常出现有没响应问题）	5709.52 hits/sec

三、客户端实践

3.1 客户端

- google chrome cronet
- IOS & Android共享一套代码

3.2 客户端降级策略

- 线上分发系统控制QUIC通道灰度策略
- 优先QUIC直连
- 其次是 TCP/HTTPS/HTTP

3.3 客户端连接复用的尝试

- idle连接空闲时间 30秒 -> 60秒
- 针对较高频率API请求比较有效
  - 每一个请求相隔30秒，避免当前链路断开
- ……

3.4 单个请求响应耗时

- 基于caddy提供的https和quic服务进行测试
- quic_client -t https://www.example.com/api
- HTTPs：　　199.8ms
- 优化前QUIC：　225.6ms
- 优化后QUIC：　188.2ms
- 小结：
  - 优化后QUIC VS HTTPS，　5.81%　效果提升
  - QUIC 优化后 VS 优化前：16.58% 效果提升

3.5 QUIC手机端弱网表现

- 当丢包率提高到20%以上时
- HTTP请求开始出现大量失败的情况
- QUIC的成功率则一直保持在90%以上

1Kb接口耗时对比(单位：毫秒)

丢包率 / 协议	0%	1%	5%	10%	20%	30%	50%
QUIC	167.3	160.5	169.5	213.5	223.8	382.4	1373.1
HTTPS	39.6	62.7	59.5	114.8	285.9	871.1	5862.4

10Kb接口耗时对比(单位：毫秒)

丢包率 / 协议	0%	1%	5%	10%	20%	30%	50%
QUIC	198.4	194.1	270.4	390.3	275.6	485.8	977.7
HTTPS	50.9	84.2	128.7	392.3	1529.5	980.9	15381.3

100Kb接口耗时对比(单位：毫秒)

丢包率 / 协议	0%	1%	5%	10%	20%	30%	50%
QUIC	332	320.6	397.9	506.2	650.1	717.1	5350.1
HTTPS	153.4	165.3	510.3	1573.8	4586.5	27674	

1Mb接口耗时对比(单位：毫秒)

丢包率 / 协议	0%	1%	5%	10%	20%	30%	50%
QUIC	1107.8	1546.5	2275.2	2178.9	2684	6085.5	40592.3
HTTPS	306.8	601.4	3444.8	18043.5	65059.3		

3.6 线上接口QUIC成功率

Variation	PV	UV	成功率(%)	相比Original(%)
Original	2968178	1024432	97.19	*
quic	554685	180981	98.19	1.00

- 线上一个灰度接口
- 成功率提升 1%
- 但不稳定!

3.7 小结

- 客户端一般选用谷歌chrome cronet库
- QUIC在弱网环境下表现好于TCP

四、反思

4.1 链路调试成本高

- 调试复杂
  - 你得需要一个可信的证书
  - 测试环境需要预埋证书
  - 只能看到前期明文握手
  - 链路调试抓包可能性为零
- 比较靠谱方式：Wireshark + 日志输出
- 高阶：中间人代理 + 证书？

4.2 人们还没有为QUIC的到来做好准备

- 运营商针对UDP通道支持不足，表现为不稳定
  - UDP带宽有时相比TCP狭窄
  - UDP 流量可能会因QoS限速判定被丢包
  - 有些小ISP会直接屏蔽UDP
  - UDP有时需要被伪装成TCP才能正常传输，eg：kcptun、udp2raw-tunnel …
- NAT局域网路由、交换机、防火墙等会禁止UDP 443同行
    - 公司出口的交换机默认不开放UDP 443
    - 防火墙有时只认TCP
- 实验室数据比真实环境漂亮很多，需要警惕！

4.3 使用UDP不一定会比TCP快

- 概念：使用UDP不一定会比TCP快，可能会更慢！
- 客户端可同时使用TCP和QUIC竞速，从而选择更优链路

4.4 确认场景是否适合QUIC

- 在乎实时性吗?
- 允许丢弃吗?
- 请求之间互相依赖吗？
- 能够同时使用 TCP & QUIC 吗?
- ……

4.5 目前观望即可

- 协议还在持续变化中
  - Draft -> RFC，前途漫漫
  - Google QUIC VS IETF QUIC ?
- 各种实现百花齐放
  - 成熟度不是很高
  - 缺少生产级特性
- 基础工具各种不完善
- ……

总结

- 我们的实践还只是刚刚开始 ……
- QUIC在移动网络环境下不一定会比TCP高效
- 建议大家目前可观望并在适合场景做小幅度尝试

关注QCon微信公众号，
获得更多干货！

Thanks!

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

主办方 Geekbang. 极客邦科技 InfoQ