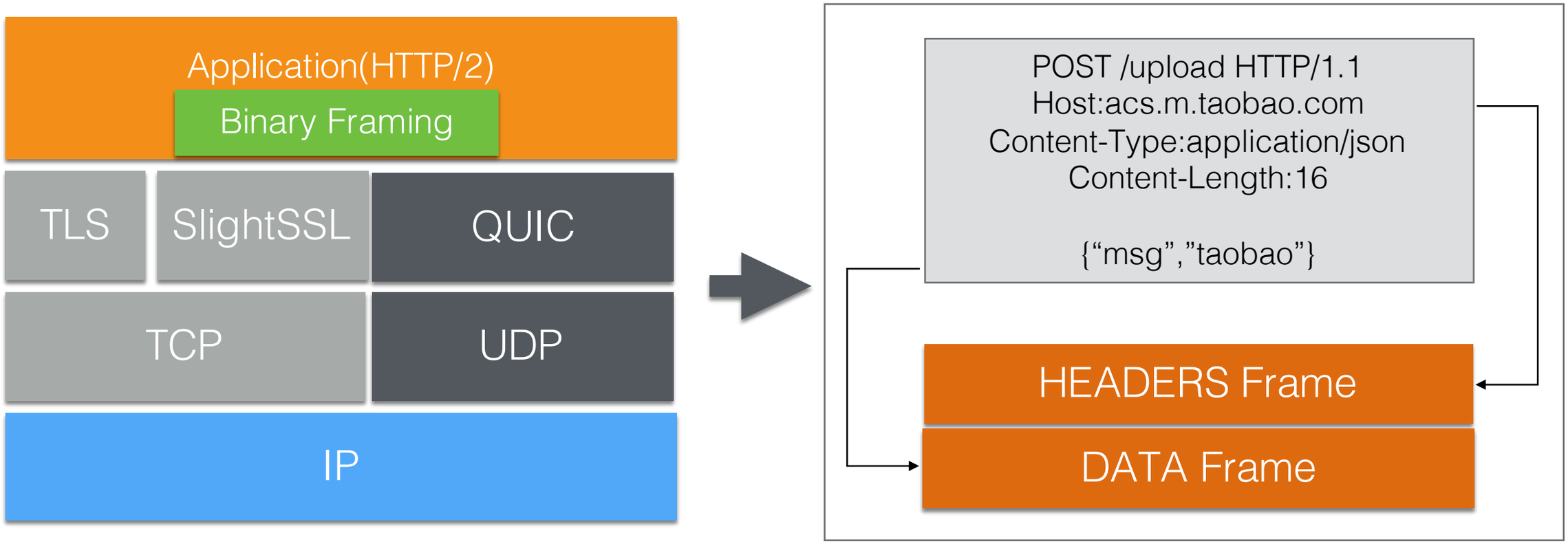
阿里巴巴HTTP2实践及无线通信协议的演进之路

阿里巴巴-移动平台 仲升(陈虓将)

更快、更省流量的标准通信

HTTP2

HTTP/2概况



- 二进制协议
- 流控
  - 会话级别 &连接级别
- 双工通信&多路复用
  - 主动下行
  - 多个请求并发

- 协议协商
  - ALPN（TLS）或protocol upgrade(明文)
  - 连接序言
- 会话协商
  - Settings Frame
- 头部压缩
  - HPACK

HTTP/2 Frames

Bit	0-7	8-15	16-23	24-31
0	Length			Type
32	Flags			
40	R	Stream Identifier		
…	Frame Payload			

http/2的公共头部

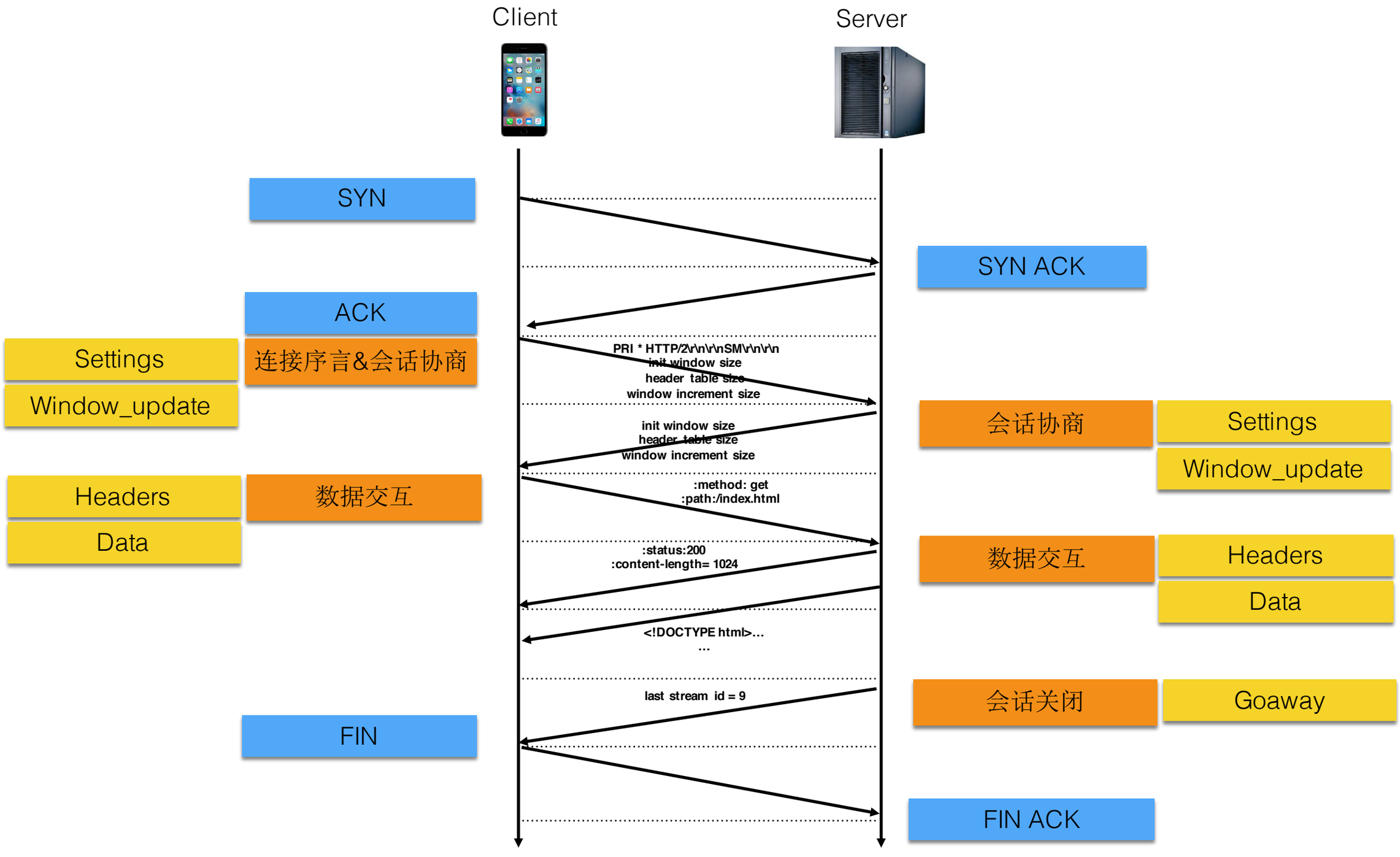
帧格式	用途
headers	存放头部数据，用以打开一个stream
continuation	延续之前未发送完毕的包头信息
data	存放应用数据
rst_stream	异常关闭一个stream
settings	参数协商
ping	心跳包，用以刺探连接是否存活
goaway	发送端优雅关闭
window_update	流控，分为stream和connection两个级别

http/2的帧格式

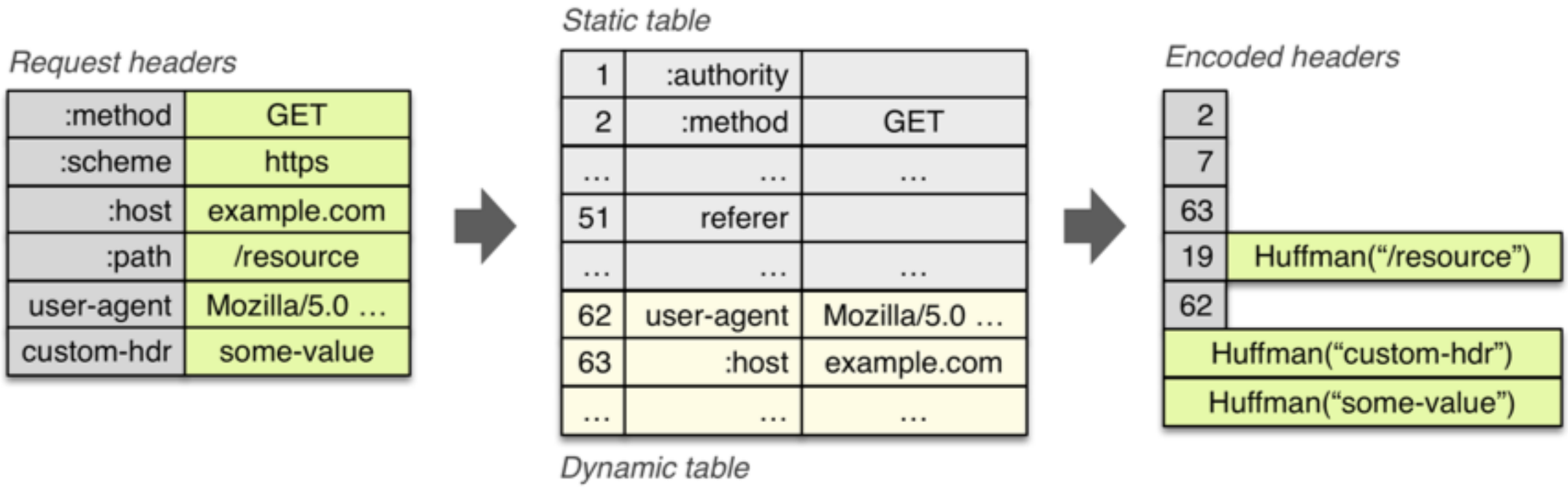
协商参数	含义
SETTINGS_HEADER_TABLE_SIZE	用于解压的头部动态压缩表最大大小，默认4096
SETTINGS_ENABLE_PUSH	用于禁止或启用服务端推送
SETTINGS_MAX_CONCURRENT_STREAMS	最大并发流数，默认无限制
SETTINGS_INITIAL_WINDOW_SIZE	会话级别的流控的初始窗口大小，默认为65535
SETTINGS_MAX_FRAME_SIZE	帧的payload大小限制，默认为16384
SETTINGS_MAX_HEADER_LIST_SIZE	压缩前头部列表的最大大小，默认无限制

http/2的SETTINGS帧的各参数的含义

HTTP/2 Workflow



HTTP/2 & HPACK



http2 HPACK

安全

DEFLATE压缩算法存在攻击风险

映射表

经常出现或重复出现的Header用映射表的Index表示

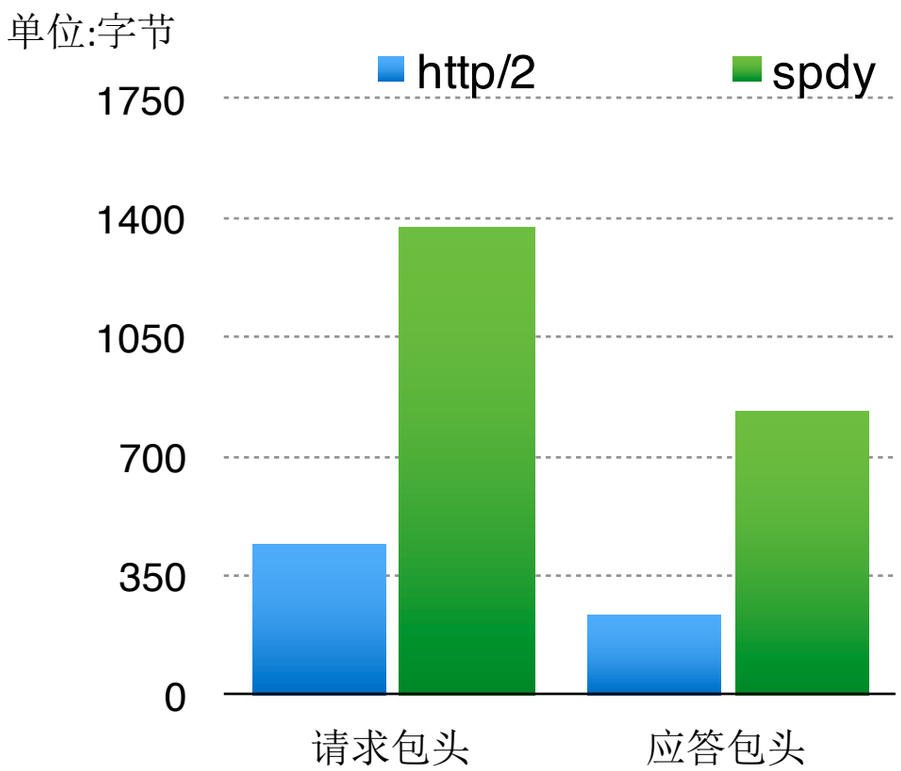
压缩率

通过新的算法得到进一步提升

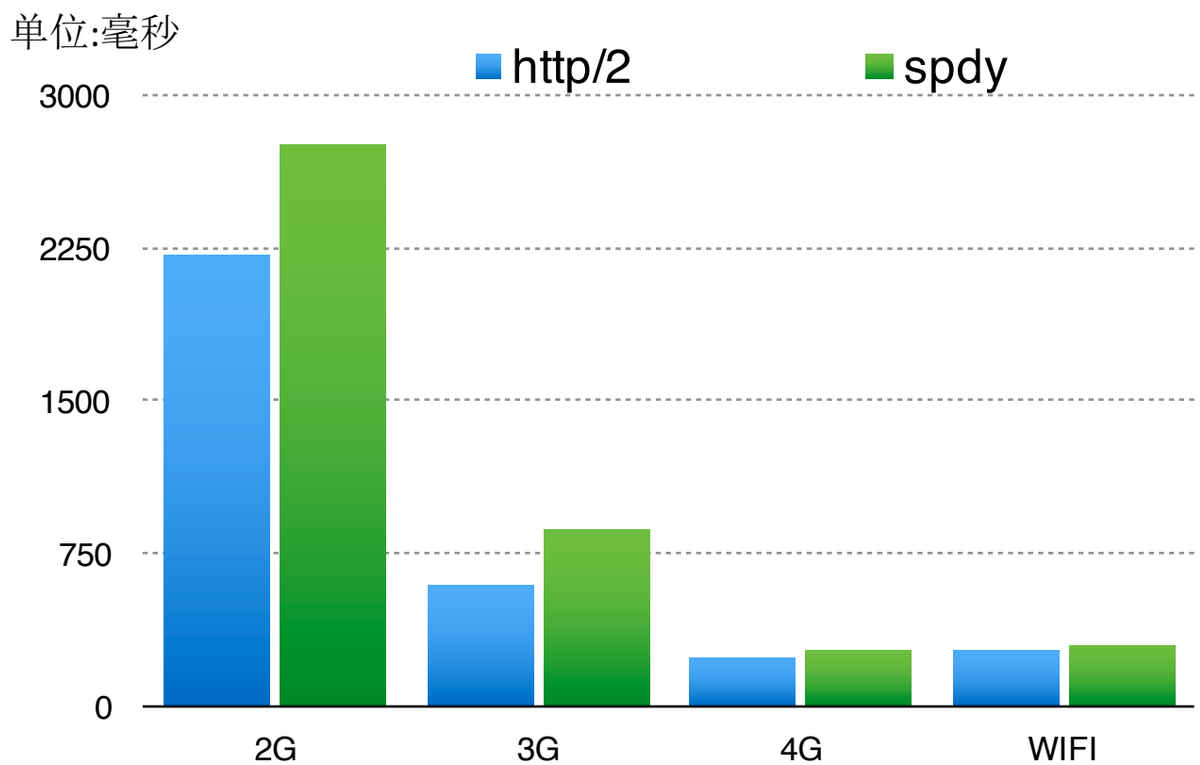
静态Huffman编码

未命中映射表的Header用Huffman编码

HTTP/2的效果

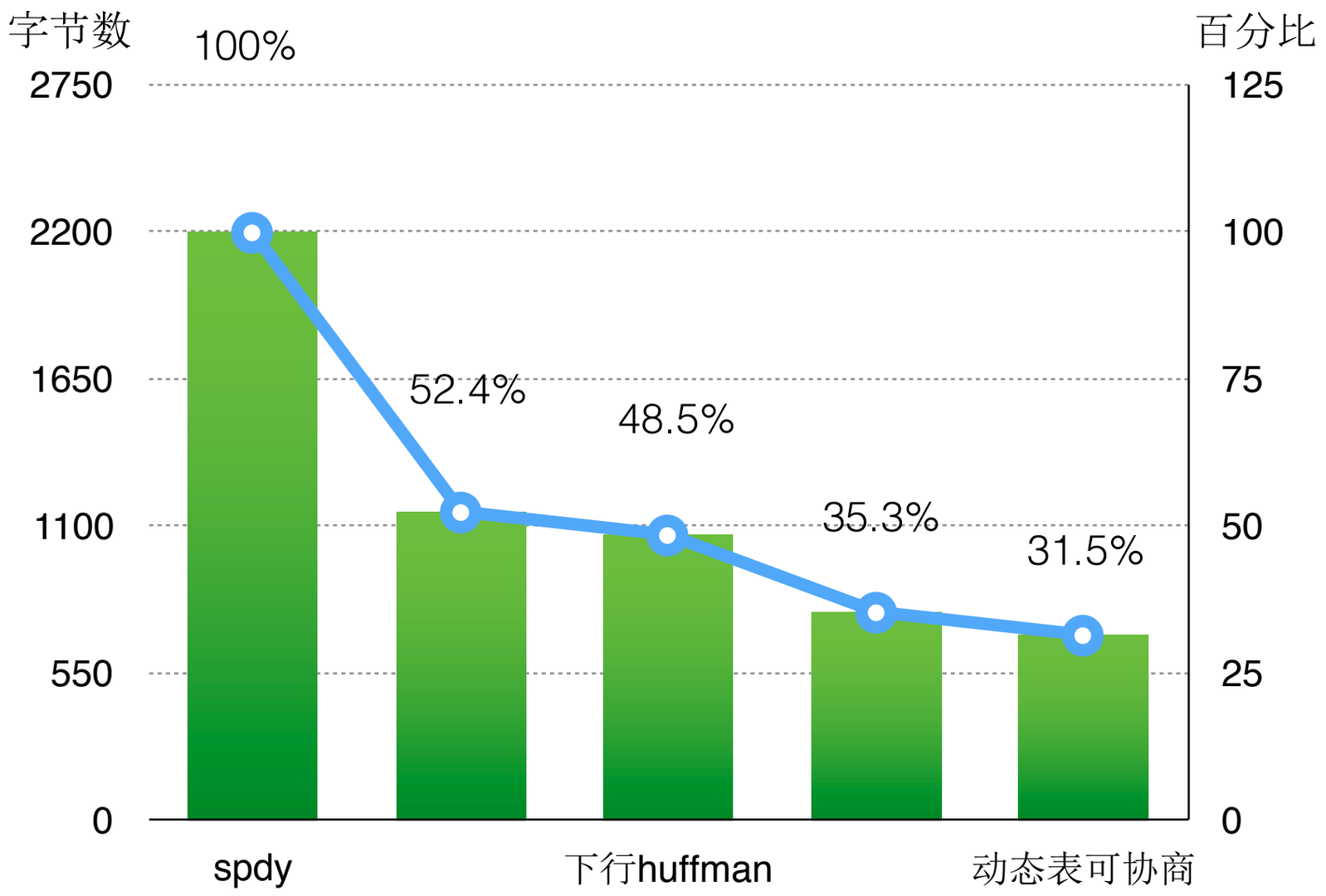


http/2请求和应答包头的流量下降



http/2请求整体提速

HTTP/2的优化过程



http2头部压缩分阶段优化

HTTP/2的实现

Nginx Patch

- 原生
  - 上下行均支持静态表
  - 上行支持动态表和Huffman编码
  - 采用默认的动态表大小，无协商
- 扩展
  - 下行动态表和Huffman编码
  - 上下行动态表大小协商

无线下的调优

- 小包合并
  - 连接序言/settings/headers合并成一个TCP包
- 流控
  - 会话级别下行流控

SDK支持

- 网络库SDK实现HTTP/2
  - 复用网络库框架，统一上层接口
  - 内部解析、封装HTTP/2

HTTP2的细节

- HPACK的动态表大小
  - 上行和下行分别独立
  - 均由服务端控制
    - 通过控制SETTING ACK实现
    - 适配两种场景
      - 压缩率优先　　调整至32K
      - 内存优先　　采用默认的4K
- HPACK动态表的更新
  - 更新必须同步，否则会出错
    - 请求封装完毕后必须发出
- HTTP2 VS SPDY
  - 预置HPACK静态表
  - 包大小
    - HTTP2 40K
    - SPDY 20K
  - 场景选择
    - PUSH场景　　优选SPDY
    - Req/Resp场景　　优选HTTP2
- HPACK的延伸
  - 统计常见字段出现的频度
  - 自定义映射表，优化自定义协议

最新的压缩协议

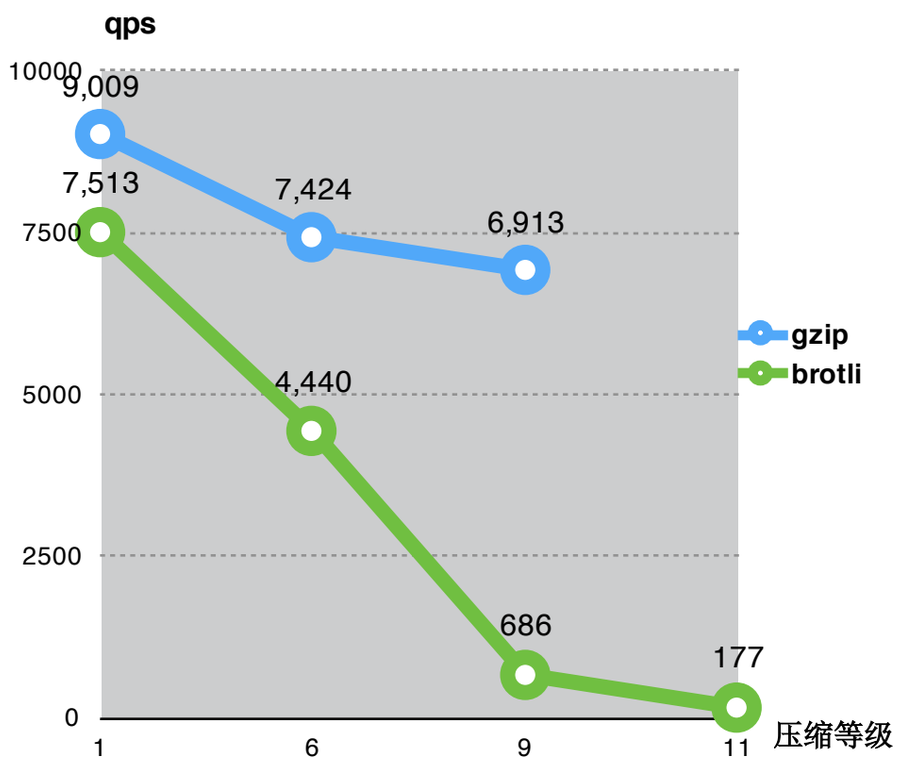
Brotli

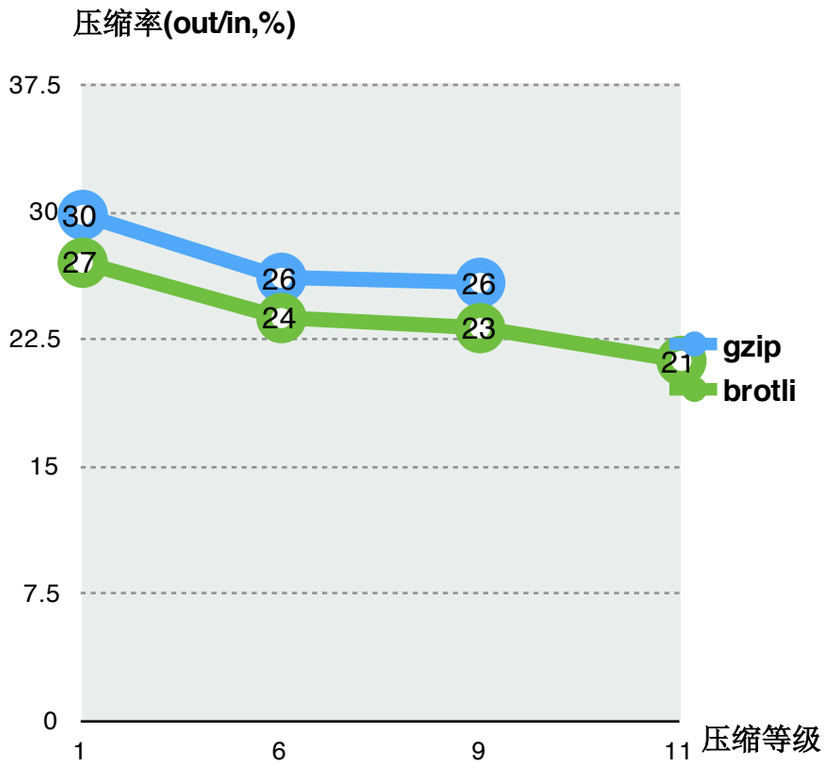
brotli vs gzip

性能

压缩率

实施





热插拔式

accept-encoding:gzip,brotli

服务端控制

压缩格式自由选择，兼顾性能和流量

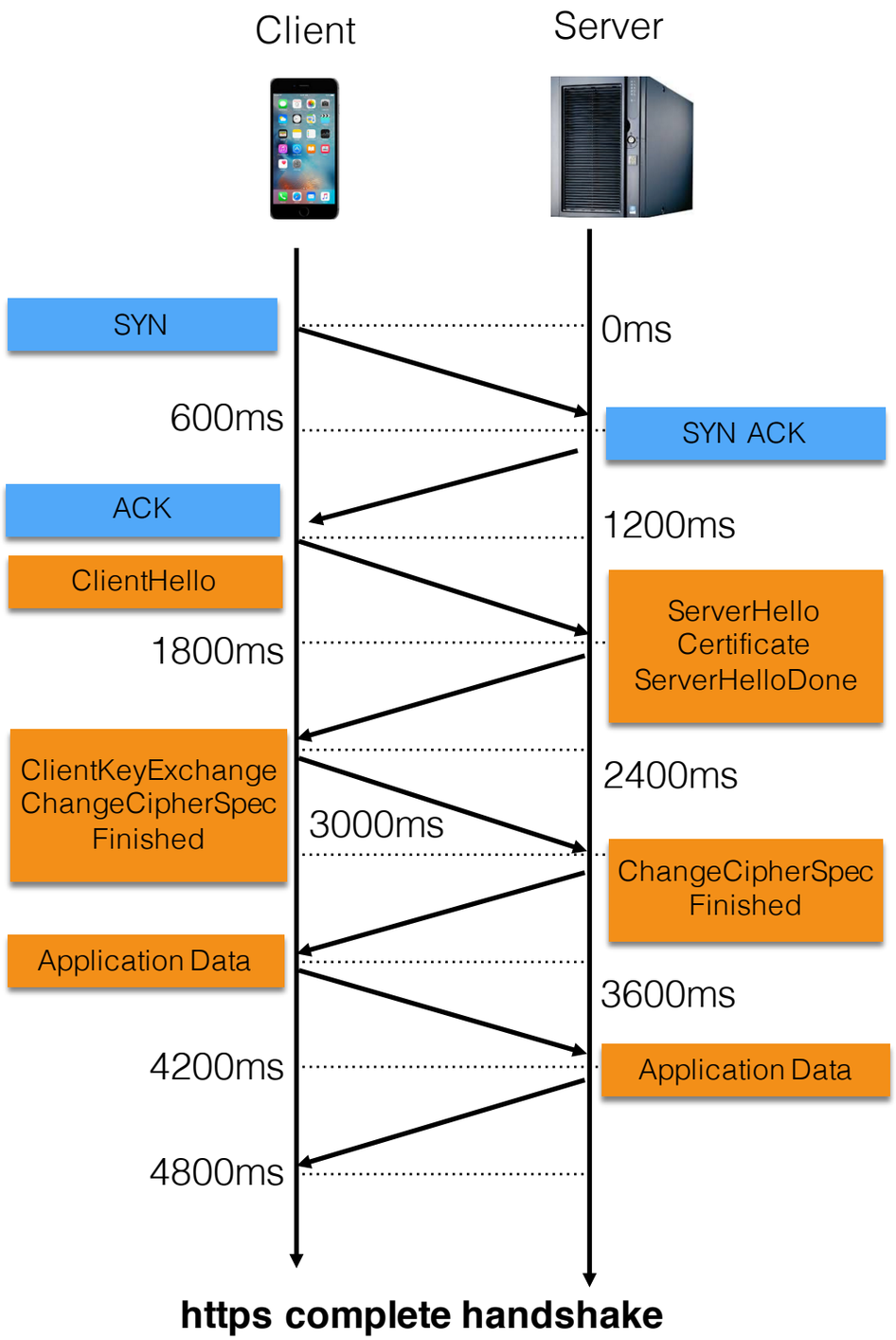
效果(统计淘宝的某个大流量域名)

相对gzip节省17.5%的流量(315字节)

无线互联网下的全站加密

SLIGHT SSL

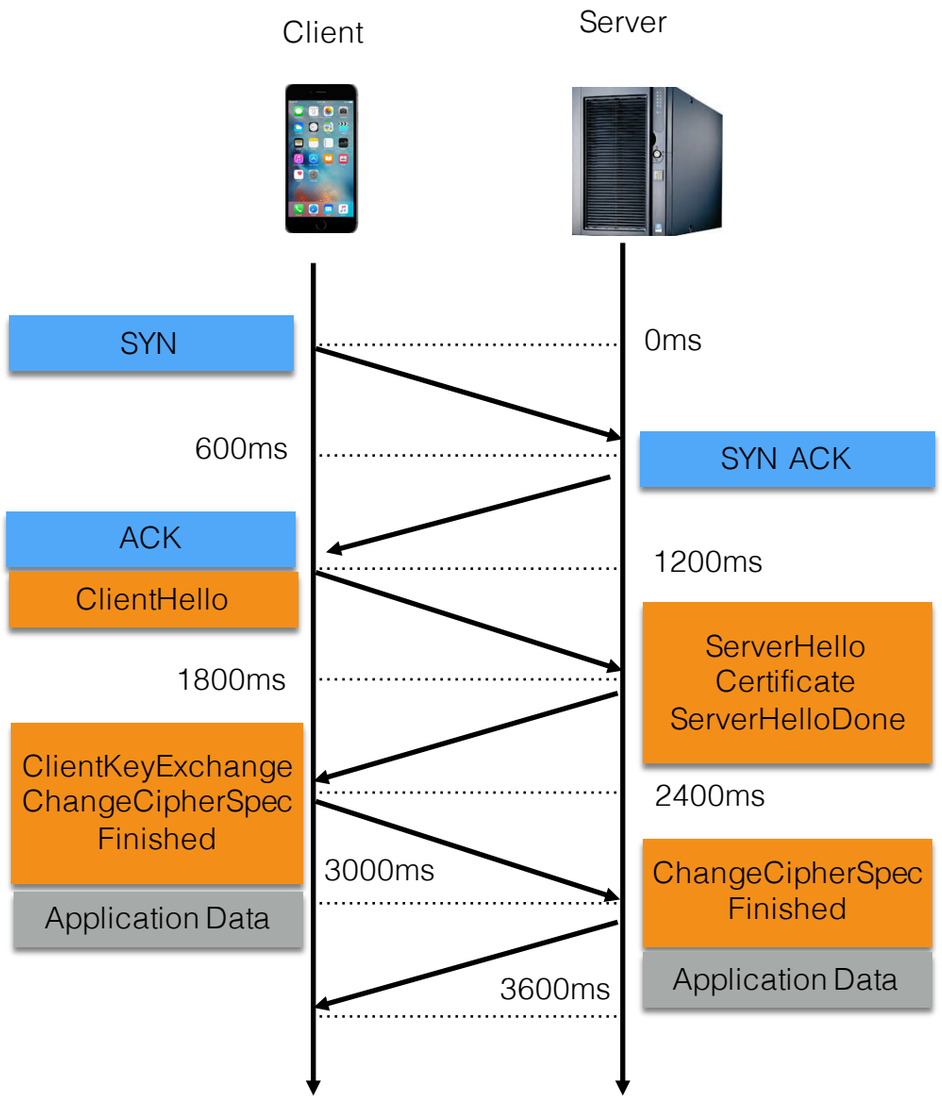
HTTPS的挑战



https complete handshake

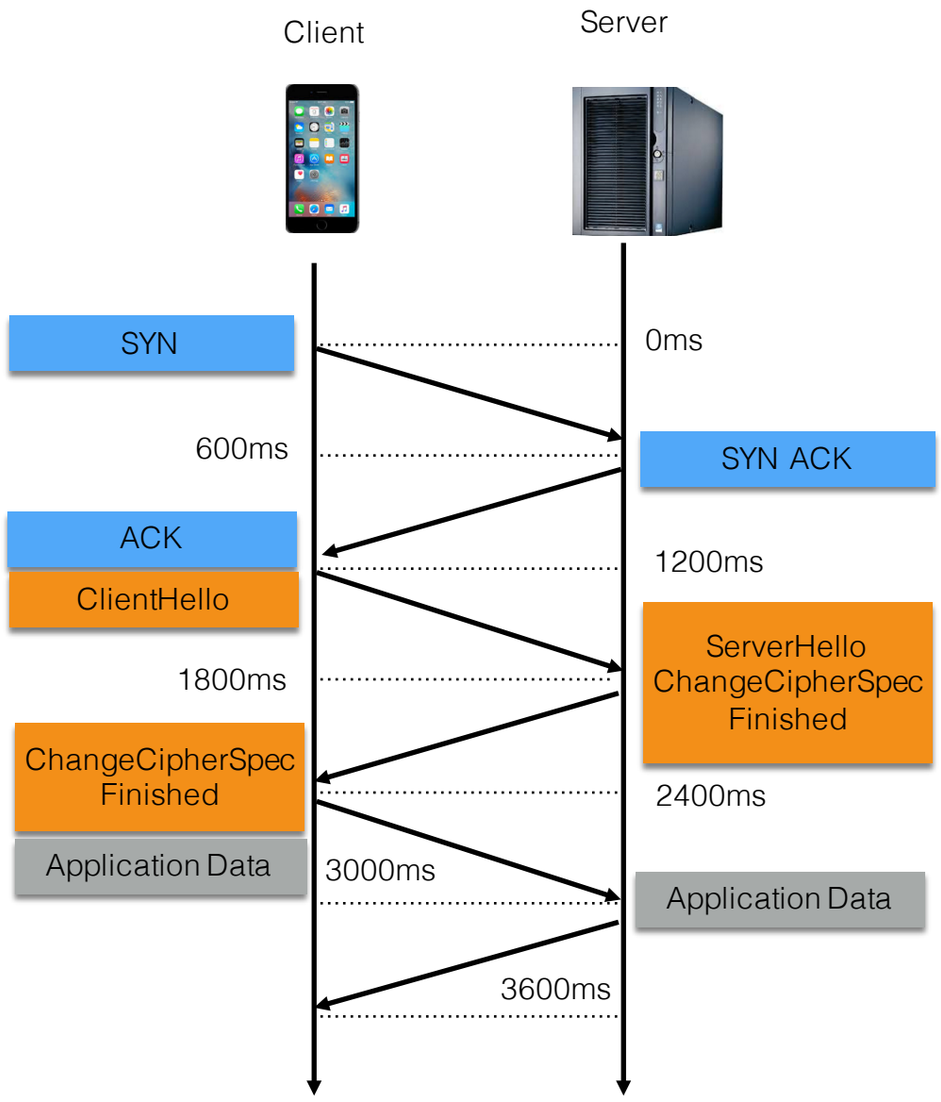
- 特点
  - 1张证书
    - 4KB+
  - 2-RTT的协商成本
    - 加密数据前先协商会话密钥
  - 3个随机数
    - 两个明文，一个密文(pre master key)
- 挑战
  - 流量放大攻击
    - 证书下放导致上下行流量相差10倍+
  - 网络时延大
    - 在弱网下尤其是2g，时延不可接受(1RTT > 1S)
  - 硬件成本高 & 首次建连攻击
    - https首次建连的性能是http的1/10

HTTPS的优化和不足



https false start

优势	节省一个RTT
劣势	需要端上和后台都做特殊支持



https session ticket

优势	节省一个RTT/提升服务器性能/降低流量
劣势	有实效性，且无法解决首次建连问题

SlightSSL设计思想

证书缓存

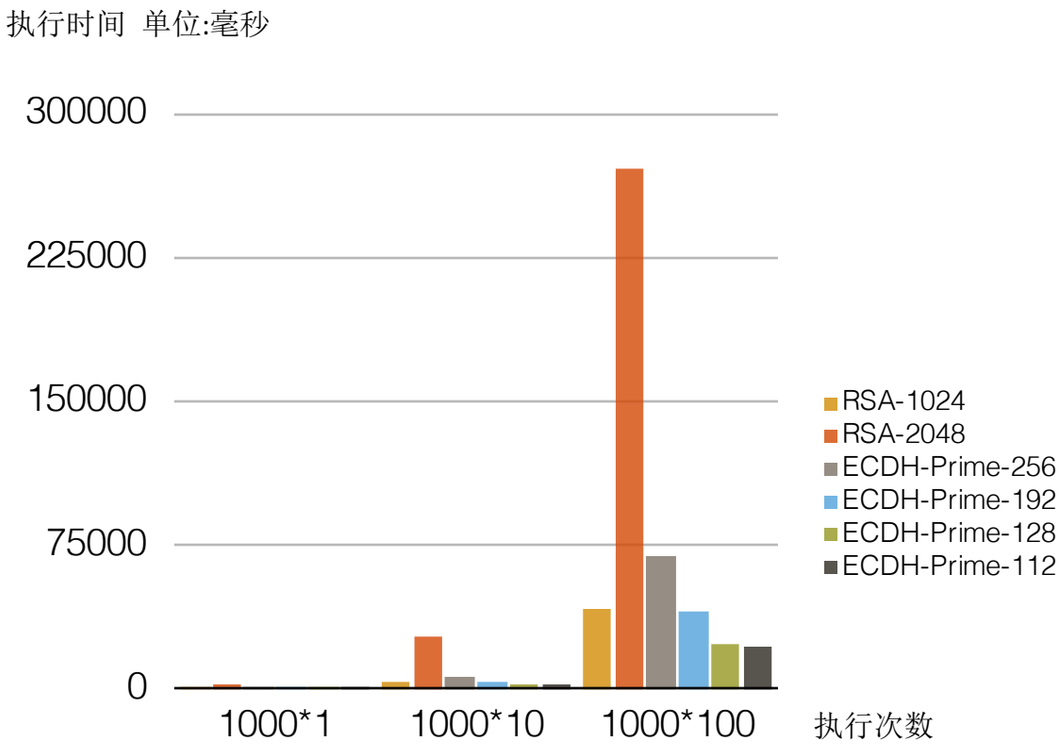
- 证书缓存到本地
- 节省流量/时延/成本

0-RTT

- 协商报文和数据报文允许一起下发
- 兼顾前向安全

性能提升

- **ECDH代替RSA**
- **Session Ticket会话复用**



ECDH性能是RSA的10倍

对称密钥长度	ECC非对称密钥长度	RSA密钥强度
80	160	1024
112	224	2048
128	256	3072

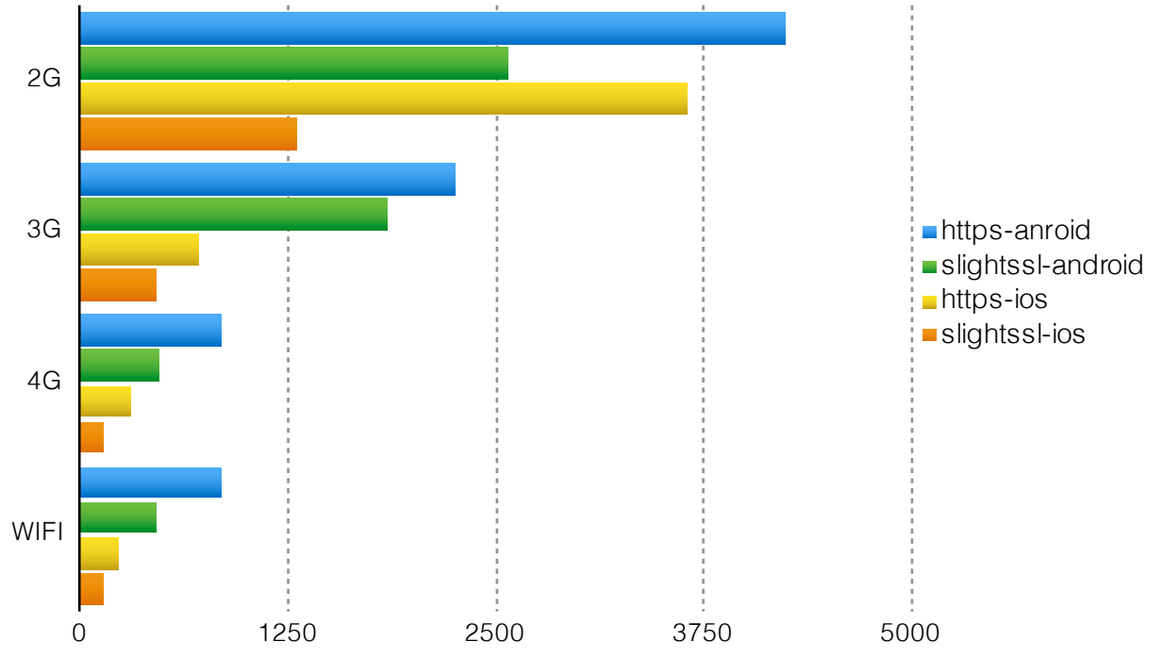
ECDH-PRIME-128

可比较密钥大小(密码分析所需计算量的角度)

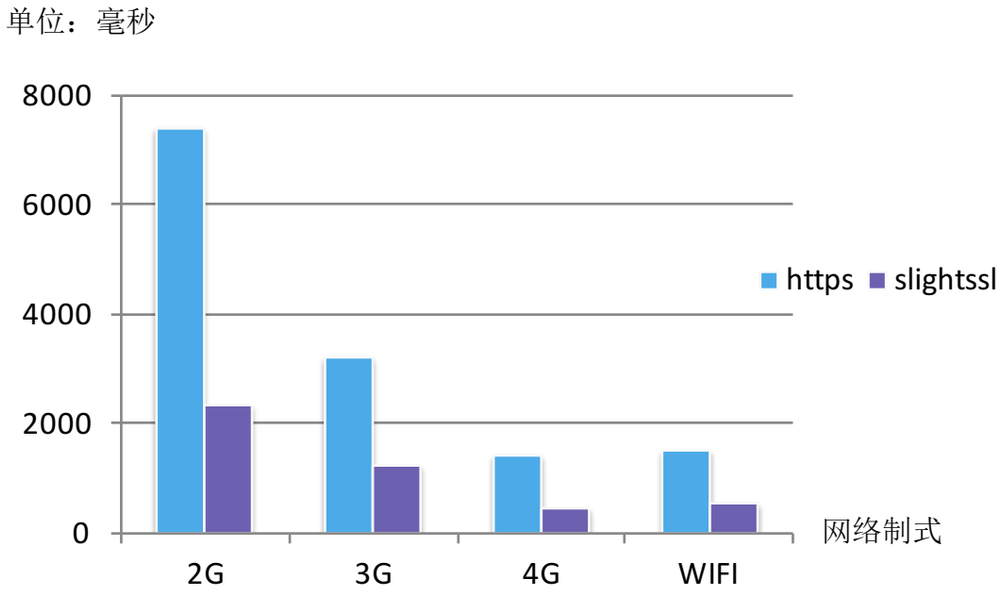
SlightSSL的效果

	网络制式	建连时间	鉴权时间	握手时间	总时间
Android	2G	1164	1126	21	**2311**
	3G	505	691	21	1217
	4G	200	236	15	451
	WIFI	228	270	19	517
IOS	2G	1082	1218	26	**2326**
	3G	567	723	10	1300
	4G	233	239	6	478
	WIFI	285	302	13	600

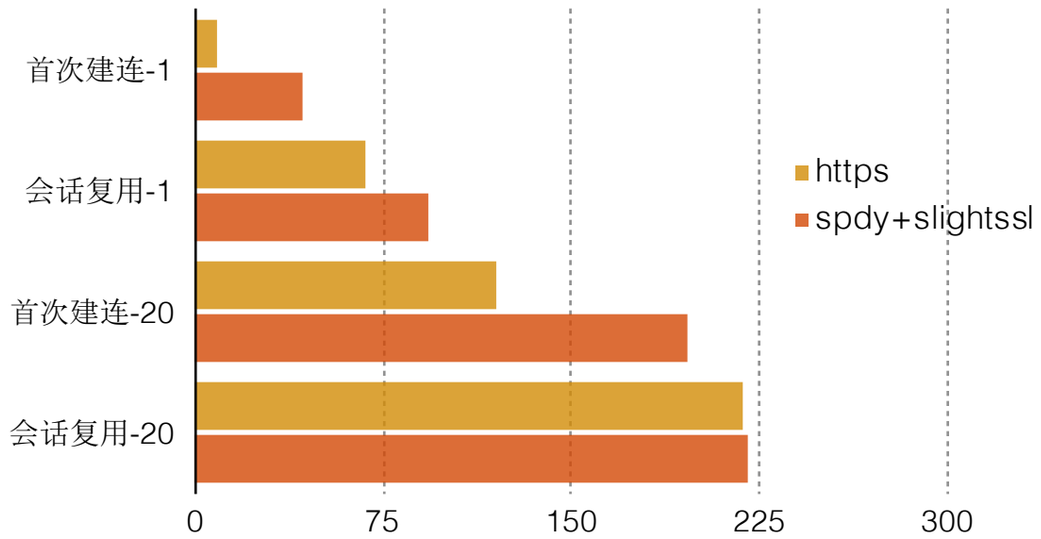
首次建连2G下2.5秒内完成



请求平均耗时对比HTTPS缩短40%



首次建连对比HTTPS缩短60%



服务端性能提升50%

SlightSSL的优化

网络优化

TCP参数调整

- 开启**TCP_NODELAY**
- 开启**TCP_QUICKACK**
- 适当增大**TCP**发送/接收缓冲区**(>=64K)**

小包精简

- 去掉不必要的报文
- 多个小包合并到一个**TCP**报文中
  - **0-RTT**

场景优化

- 会话复用避免**SDK**创建椭圆曲线密钥对
- **CDN**场景
  - 提高服务端**send buf**
  - 开启**0-RTT**

性能优化

SessionTicket有效

- **SessionTicket**的性能远高于**ECDH**
- 延长有效期提高会话复用度，进而提升性能

小包精简

- **0-RTT**
- 断连时不发送**goaway**

降级策略

- 通过调度中心下发明文策略

谢谢!