

HTTP的升级改进

@M了个J
李明杰

<https://github.com/CoderMJLee>

<https://space.bilibili.com/325538782>



实力IT教育 www.520it.com



HTTP协议的不足 (HTTP/1.1)

- 同一时间，一个连接只能对应一个请求
 - 针对同一个域名，大多数浏览器允许同时最多6个并发连接
- 只允许客户端主动发起请求
 - 一个请求只能对应一个响应
- 同一个会话的多次请求中，头信息会被重复传输
 - 通常会给每个传输增加500~800字节的开销
 - 如果使用 Cookie，增加的开销有时能达到上千字节

- SPDY (speedy的缩写)，是基于TCP的应用层协议，它强制要求使用SSL/TLS

- 2009年11月，Google宣布将SPDY作为提高网络速度的内部项目

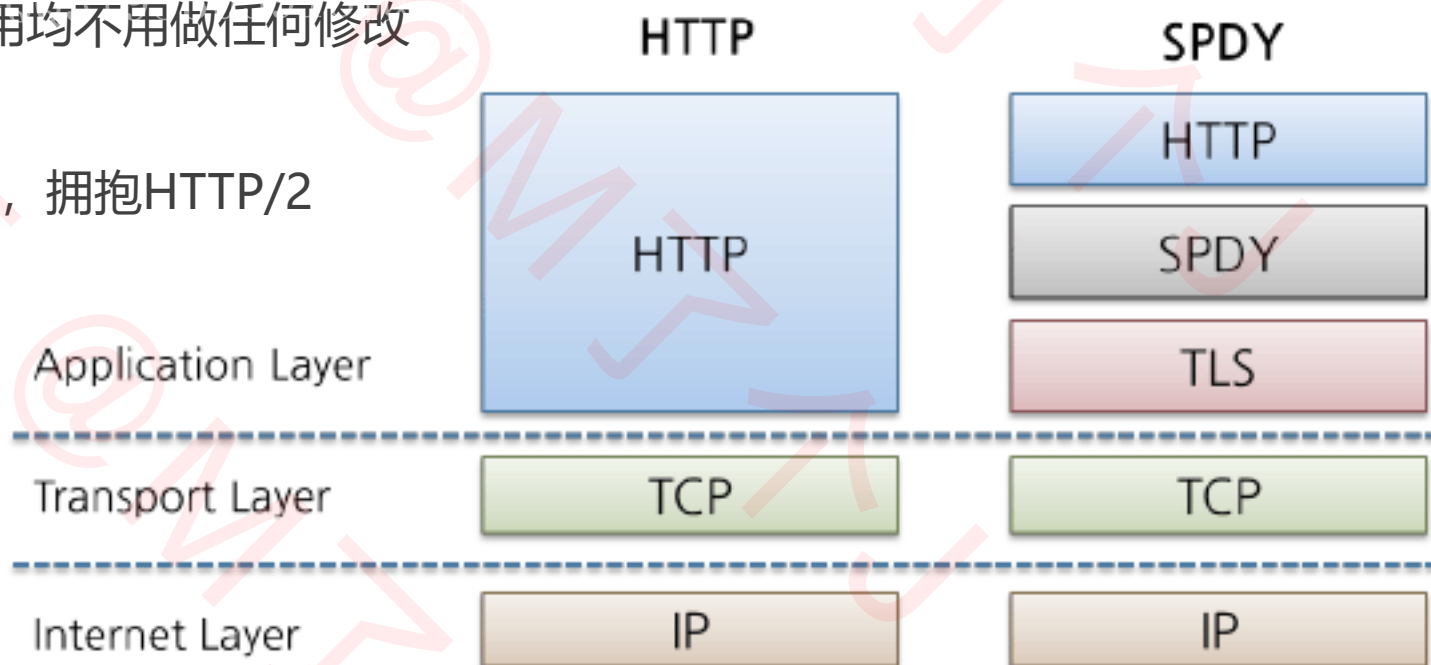
- SPDY与HTTP的关系

- SPDY并不用于取代HTTP，它只是修改了HTTP请求与响应的传输方式

- 只需增加一个SPDY层，现有的所有服务端应用均不用做任何修改

- SPDY是HTTP/2的前身

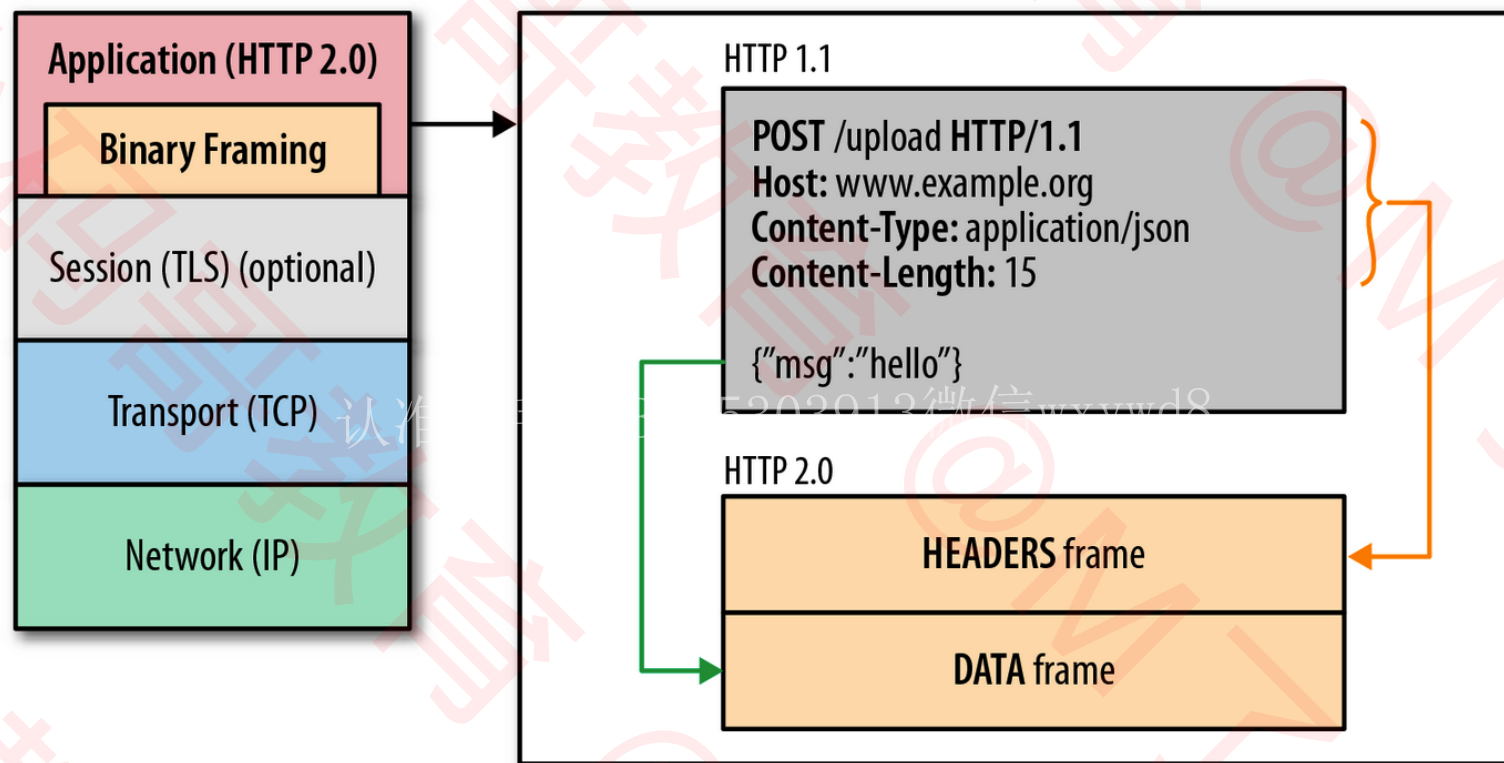
- ✓ 2015年9月，Google宣布移除对SPDY的支持，拥抱HTTP/2



HTTP/2

- HTTP/2, 于2015年5月以[RFC 7540](#)正式发表
 - 根据W3Techs的数据, 截至2019年6月, 全球有36.5%的网站支持了HTTP/2
- HTTP/1.1和HTTP/2速度对比
 - <http://www.http2demo.io/>
 - <https://http2.akamai.com/demo>
- HTTP/2在底层传输做了很多的改进和优化, 但在语意上完全与HTTP/1.1兼容
 - 比如请求方法 (如GET、POST) 、Status Code、各种Headers等都没有改变
 - 因此, 要想升级到HTTP/2
 - ✓ 开发者不需要修改任何代码
 - ✓ 只需要升级服务器配置、升级浏览器

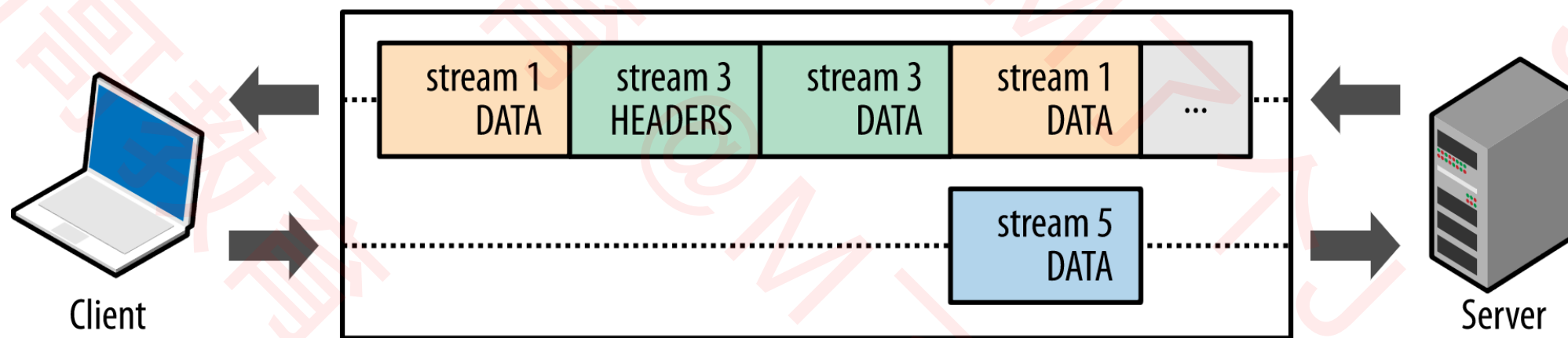
HTTP/2的特性 – 二进制格式



- HTTP/2采用二进制格式传输数据，而非HTTP/1.1的文本格式
- 二进制格式在协议的解析和优化扩展上带来更多的优势和可能

HTTP/2 — 一些基本概念

- 数据流：已建立的连接内的双向字节流，可以承载一条或多条消息
- 所有通信都在一个TCP连接上完成，此连接可以承载任意数量的双向数据流
- 消息：与逻辑HTTP请求或响应消息对应，由一系列帧组成
- 帧：HTTP/2通信的最小单位，每个帧都包含帧头（会标识出当前帧所属的数据流）
- 来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据流标识符重新组装



HTTP/2 — 一些基本概念

Connection

Stream 1

Request message

HEADERS frame (stream 1)

:method: GET
:path: /index.html
:version: HTTP/2.0
:scheme: https
user-agent: Chrome/26.0.1410.65

Response message

HEADERS frame (stream 1)

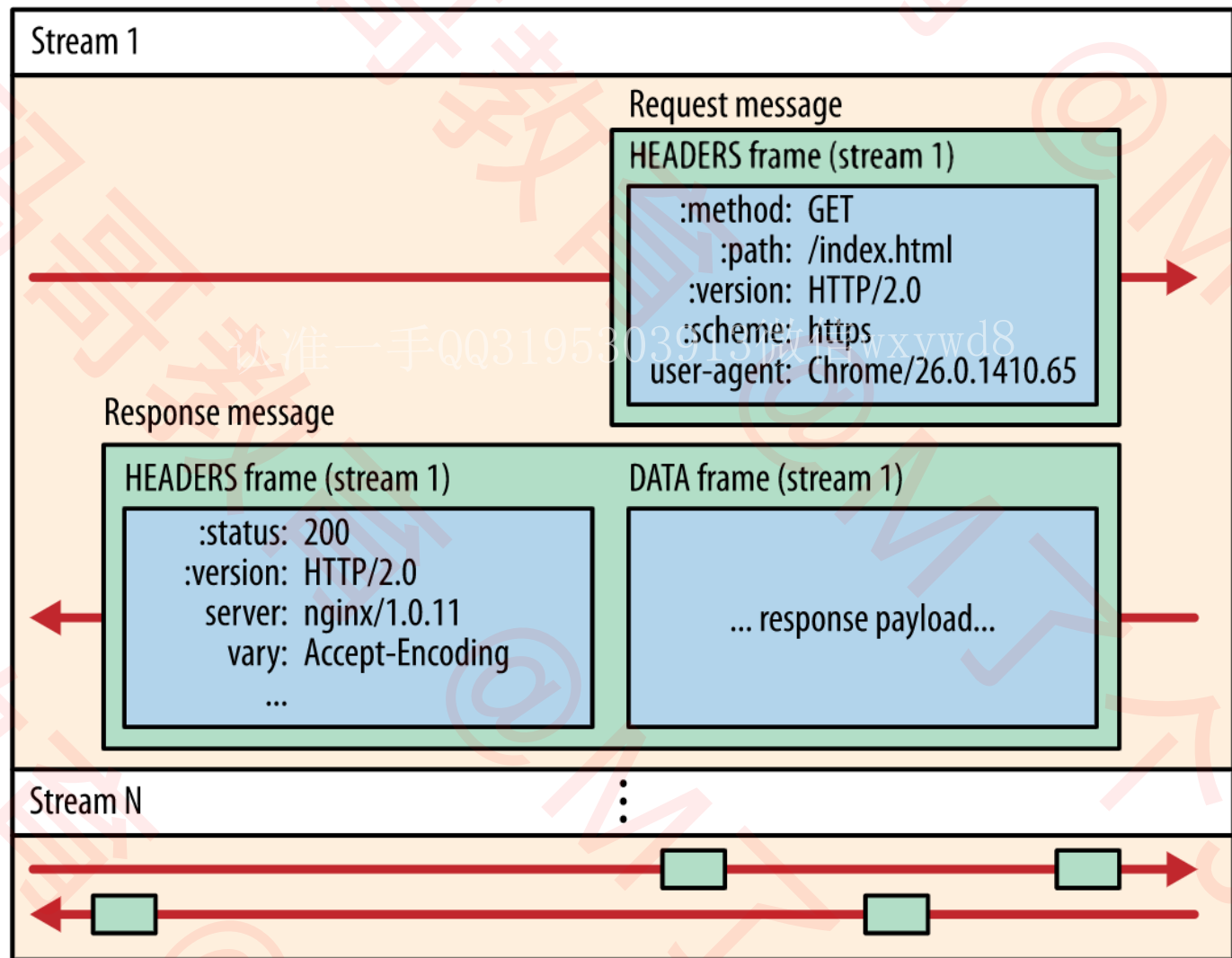
:status: 200
:version: HTTP/2.0
server: nginx/1.0.11
vary: Accept-Encoding
...

DATA frame (stream 1)

... response payload...

Stream N

⋮



HTTP/2的特性 – 多路复用 (Multiplexing)

- 客户端和服务端可以将 HTTP消息分解为互不依赖的帧，然后交错发送，最后在另一端把它们重新组装起来
- 并行交错地发送多个请求，请求之间互不影响
- 并行交错地发送多个响应，响应之间互不干扰
- 使用一个连接并行发送多个请求和响应
- 不必再为绕过HTTP/1.1限制而做很多工作
 - 比如image sprites、合并CSS\JS、内嵌CSS\JS\Base64图片、域名分片等

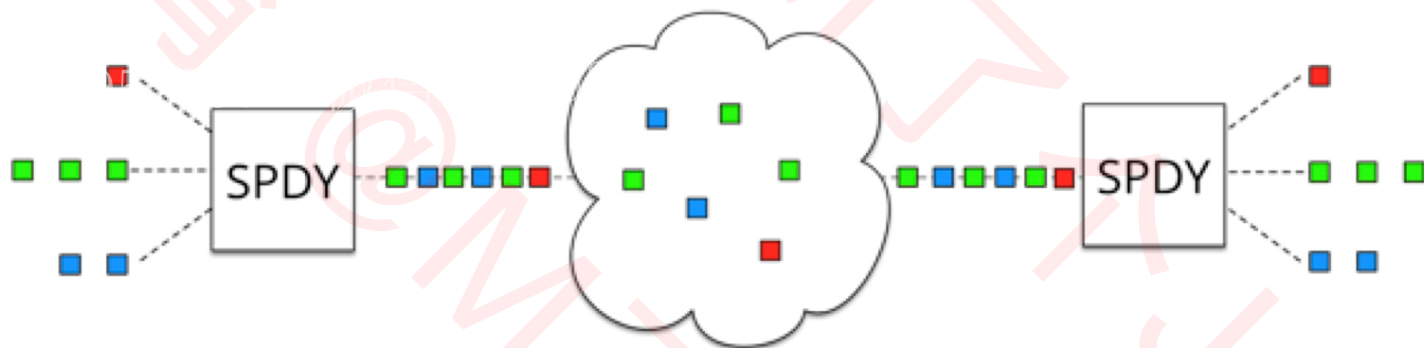


image sprites

- image sprites (也叫做CSS Sprites)，将多张小图合并成一张大图
- 最后通过CSS结合小图的位置、尺寸进行精准定位



HTTP/2的特性 – 多路复用 (Multiplexing)

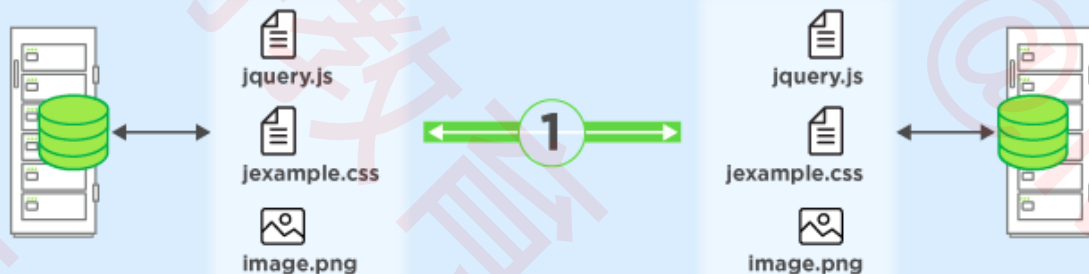
HTTP 1.1

3 TCP CONNECTIONS

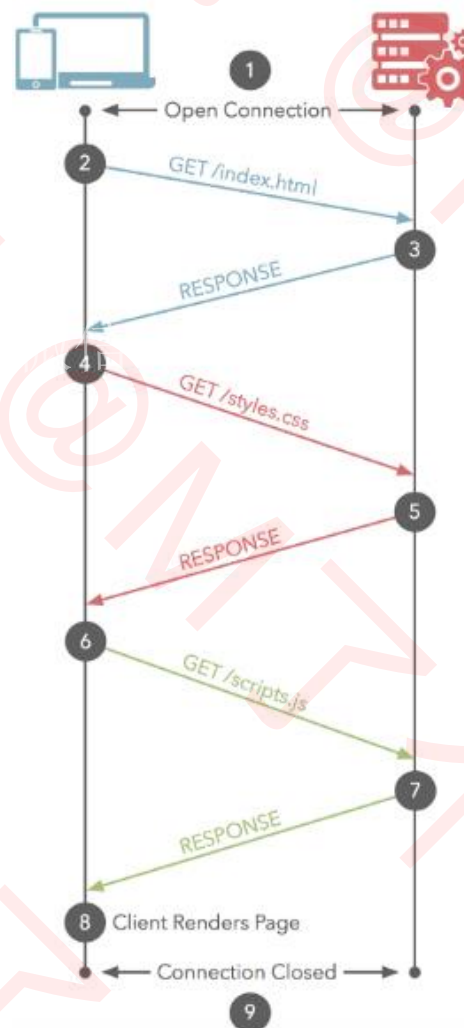


HTTP/2

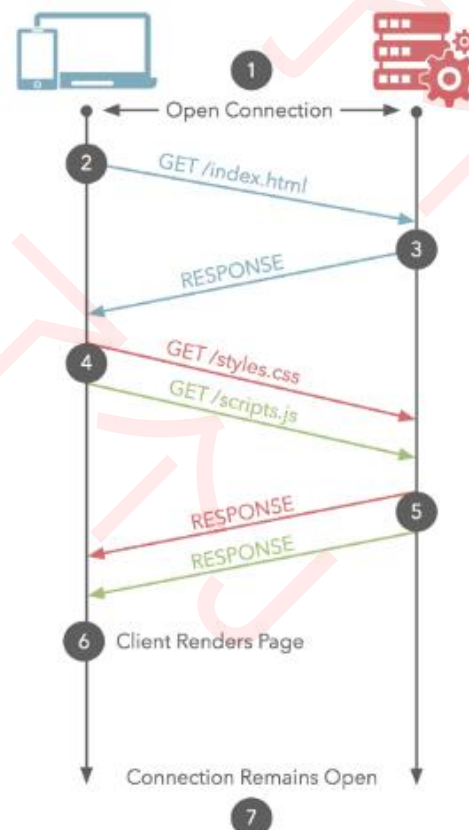
1 TCP CONNECTION



HTTP/1.1 Baseline



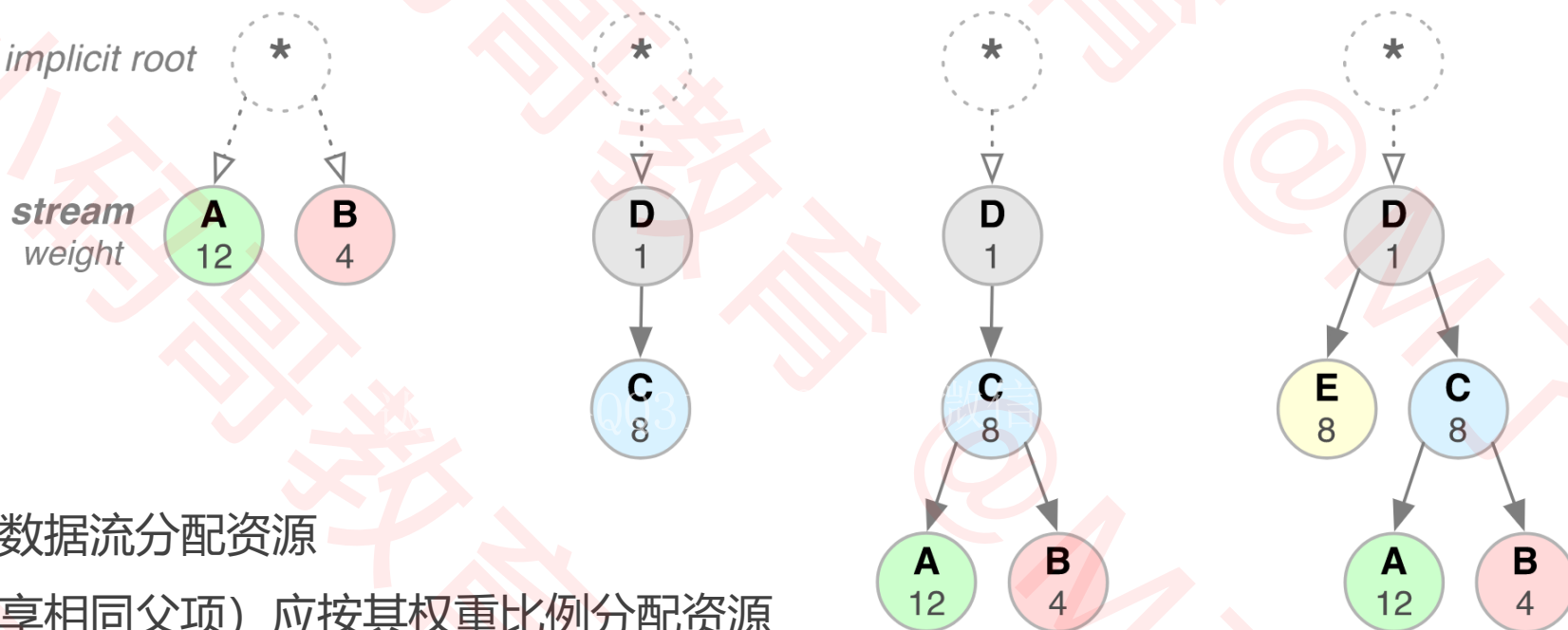
HTTP/2 Multiplexing



HTTP/2的特性 – 优先级

- HTTP/2 标准允许每个数据流都有一个关联的权重和依赖关系
 - 可以向每个数据流分配一个介于1至256之间的整数
 - 每个数据流与其他数据流之间可以存在显式依赖关系
- 客户端可以构建和传递“优先级树”，表明它倾向于如何接收响应
- 服务器可以使用此信息通过控制CPU、内存和其他资源的分配设定数据流处理的优先级
 - 在资源数据可用之后，确保将高优先级响应以最优方式传输至客户端

HTTP/2的特性 – 优先级



- 应尽可能先给父数据流分配资源
- 同级数据流（共享相同父项）应按其权重比例分配资源

- ① A、B依赖于隐式“根数据流”，A获得的资源比例是12/16，B获得的资源比例是4/16
- ② D依赖于根数据流，C依赖于D，D应先于C获得完整资源分配
- ③ D应先于C获得完整资源分配，C应先于A和B获得完整资源分配，B获得的资源是A所获资源的1/3
- ④ D应先于E和C获得完整资源分配，E和C应先于A和B获得相同的资源分配，B获得的资源是A所获资源的1/3

HTTP/2的特性 – 头部压缩

Request #1

:method	GET
:scheme	https
:host	example.com
:path	/resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

HEADERS frame (Stream 1)

:method: GET
:scheme: https
:host: example.com
:path: /resource
accept: image/jpeg
user-agent: Mozilla/5.0 ...

Request #2

:method	GET
:scheme	https
:host	example.com
:path	/new_resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

HEADERS frame (Stream 3)

:path: /new_resource

- HTTP/2使用HPACK压缩请求头和响应头
 - 可以极大减少头部开销，进而提高性能
- 早期版本的HTTP/2和SPDY使用 zlib压缩
 - 可以将所传输头数据的大小减小85%~88%
 - 但在2012年夏天，被攻击导致会话劫持
 - 后被更安全的HPACK取代

HTTP/2的特性 - 头部压缩

Request headers

:method	GET
:scheme	https
:host	example.com
:path	/resource
user-agent	Mozilla/5.0 ...
custom-hdr	some-value

Static table

1	:authority	
2	:method	GET
...
51	referer	
...
62	user-agent	Mozilla/5.0 ...
63	:host	example.com
...

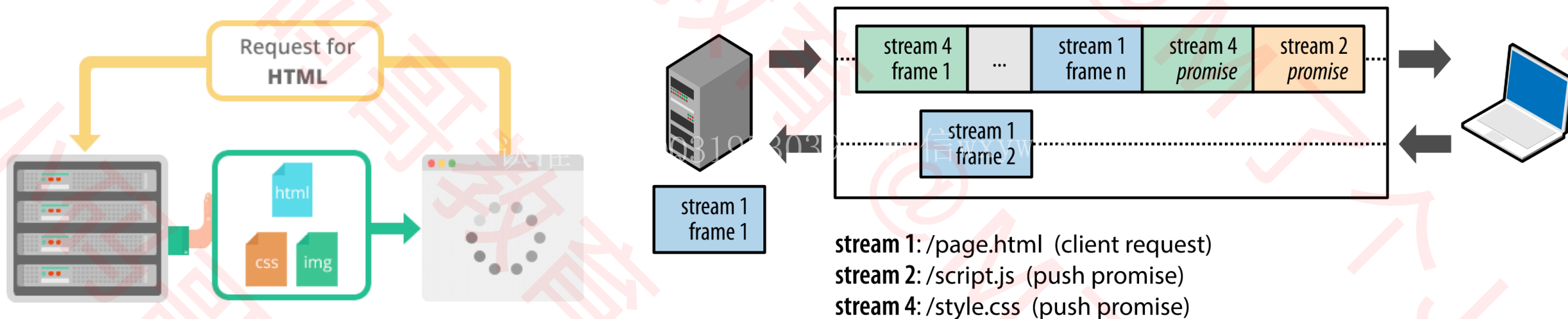
Dynamic table

Encoded headers

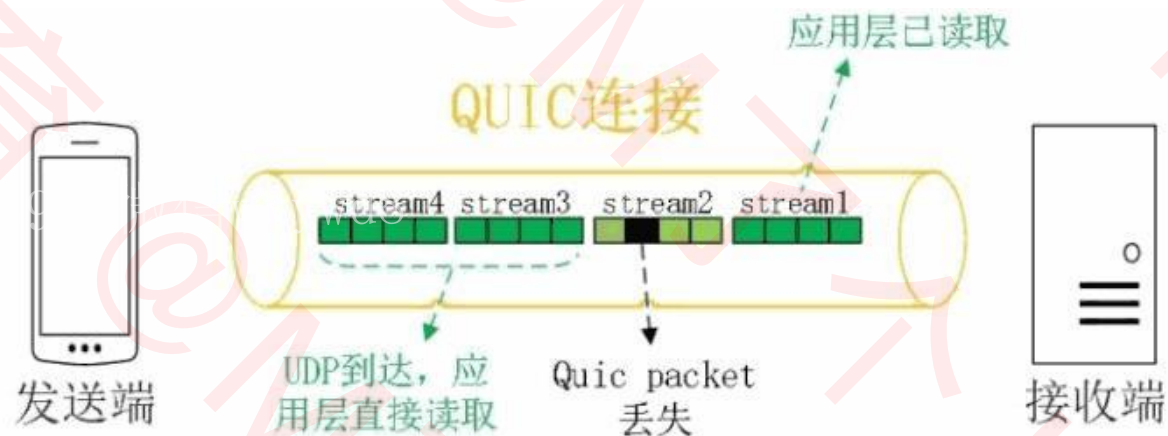
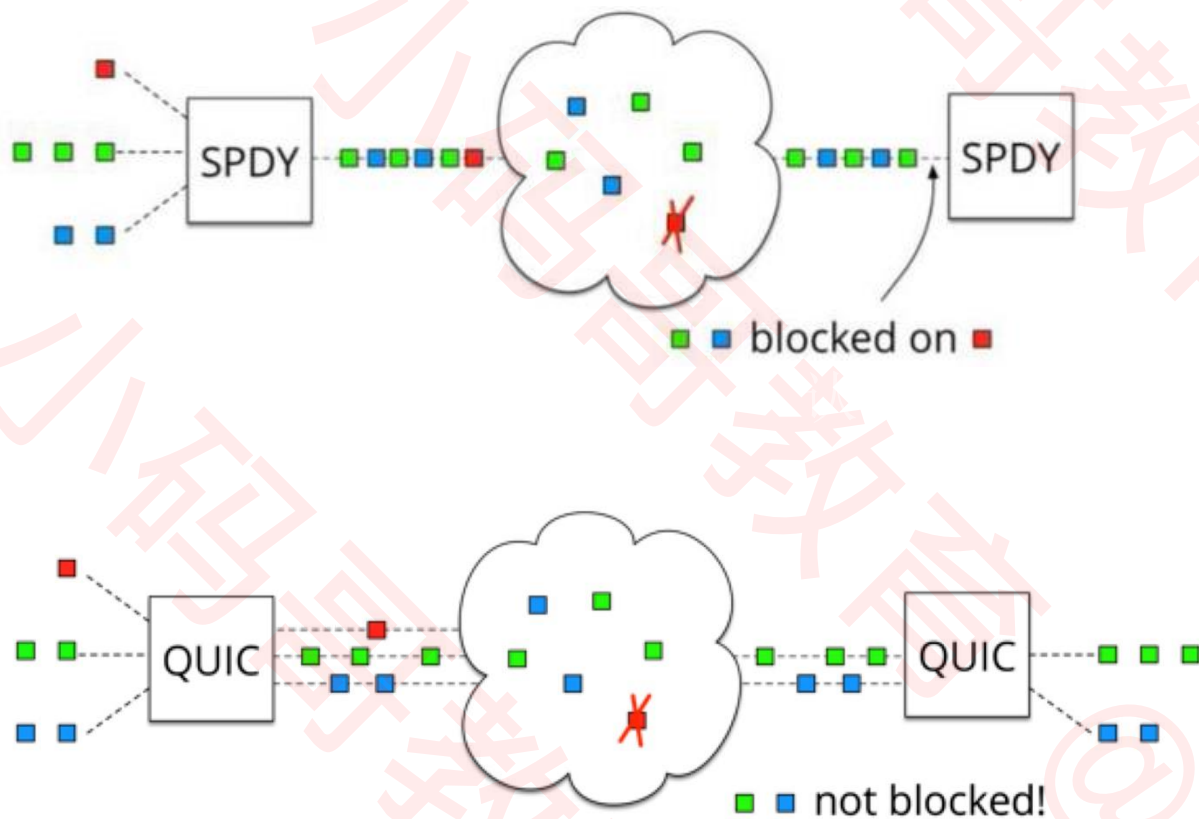
2	
7	
63	
19	Huffmann("/resource")
62	
	Huffmann("custom-hdr")
	Huffmann("some-value")

HTTP/2的特性 – 服务器推送 (Server Push)

- 服务器可以对一个客户端请求发送多个响应
- 除了对最初请求的响应外，服务器还可以向客户端推送额外资源，而无需客户端额外明确地请求

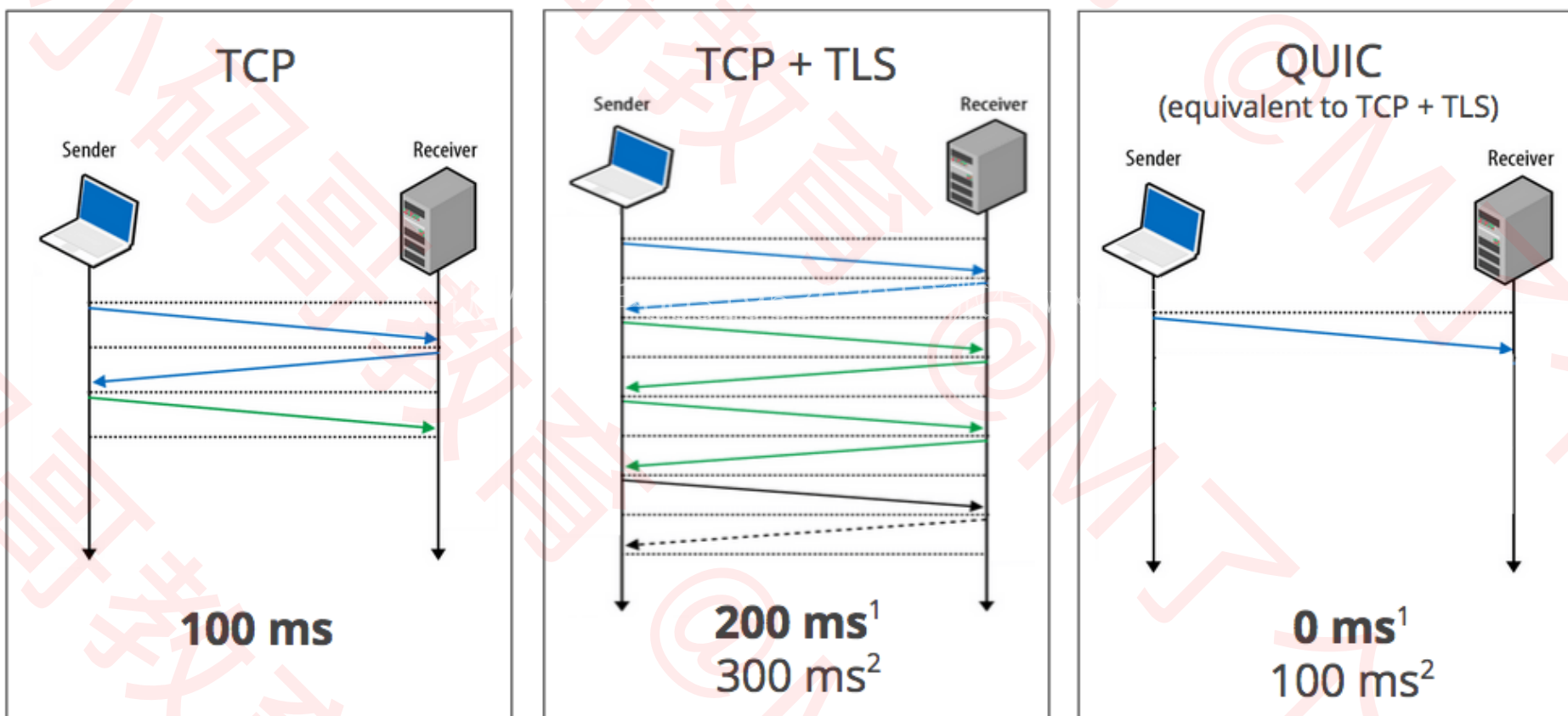


HTTP/2的问题 — 队头阻塞 (head of line blocking)



HTTP/2的问题 – 握手延迟

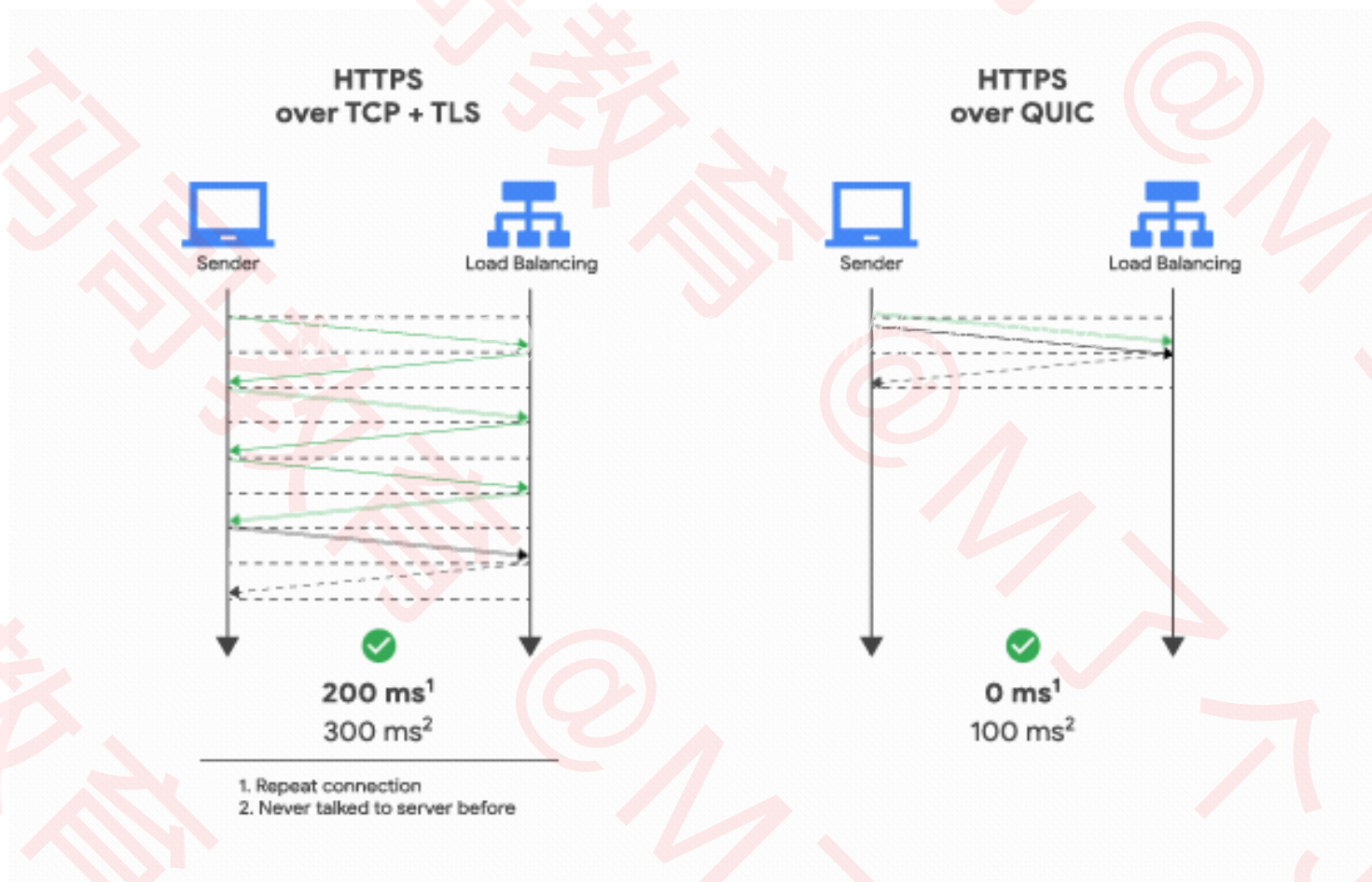
Zero RTT Connection Establishment



1. Repeat connection
2. Never talked to server before

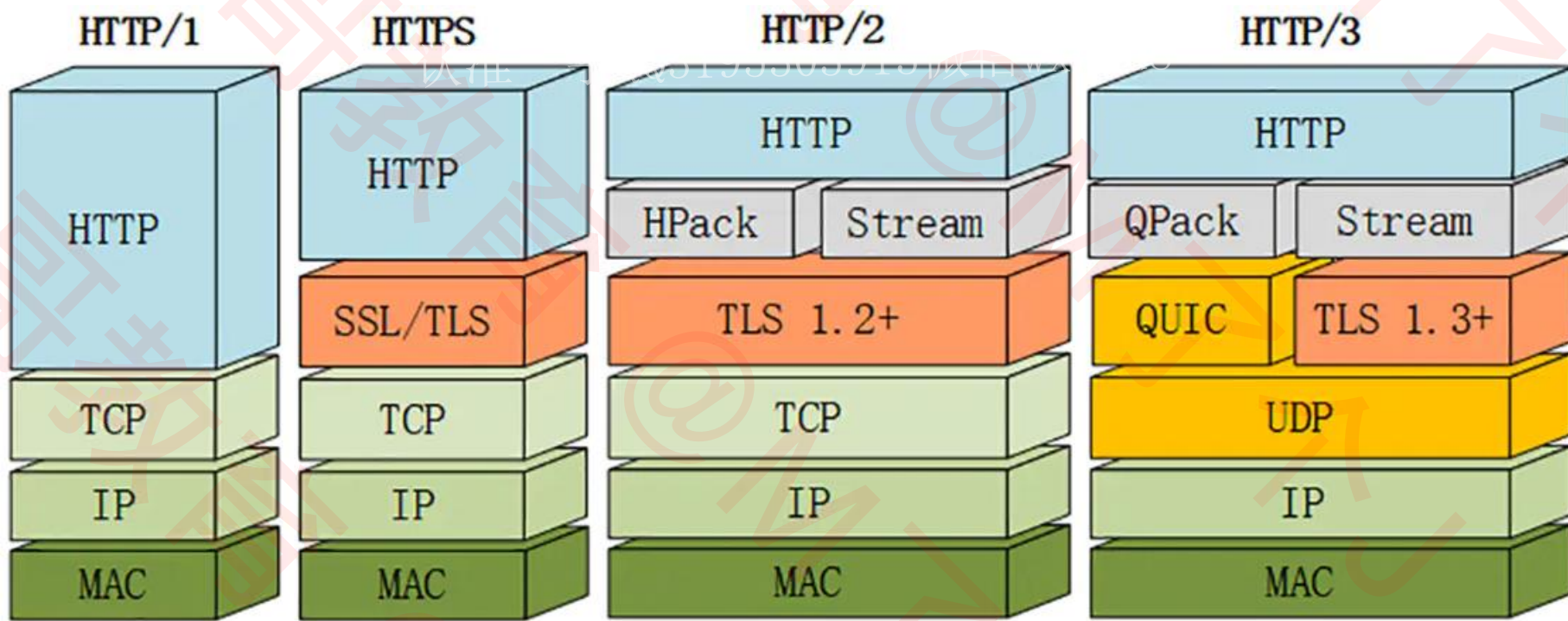
■ RTT (Round Trip Time) : 往返时延, 可以简单理解为通信一来一回的时间

HTTP/2的问题 – 握手延迟



HTTP/3

- Google觉得HTTP/2仍然不够快，于是就有了HTTP/3
- HTTP/3由Google开发，弃用TCP协议，改为使用基于UDP协议的QUIC协议实现
- QUIC (**Q**uick **U**DP **I**nternet **C**onnections)，译为：快速UDP网络连接，由Google开发，在2013年实现
- 于2018年从HTTP-over-QUIC改为HTTP/3



HTTP/3 — 疑问

- HTTP/3基于UDP，如何保证可靠传输？
 - 由QUIC来保证
- 为何Google不开发一个新的不同于TCP、UDP的传输层协议？
 - 目前世界上的网络设备基本只认TCP、UDP
 - 如果要修改传输层，意味着操作系统的内核也要修改
 - 另外，由IETF标准化的许多TCP新特性都因缺乏广泛支持而没有得到广泛的部署或使用
 - 因此，要想开发并应用一个新的传输层协议，是极其困难的一件事情

HTTP/3的特性 – 连接迁移

- TCP基于4要素（源IP、源端口、目标IP、目标端口）
 - 切换网络时至少会有一个要素发生变化，导致连接发生变化
 - 当连接发生变化时，如果还使用原来的TCP连接，则会导致连接失败，就得等原来的连接超时时后重新建立连接
 - 所以我们有时候发现切换到一个新网络时，即使新网络状况良好，但内容还是需要加载很久
 - 如果实现得好，当检测到网络变化时立刻建立新的TCP连接，即使这样，建立新的连接还是需要几百毫秒的时间
- QUIC的连接不受4要素的影响，当4要素发生变化时，原连接依然维持
 - QUIC连接不以4要素作为标识，而是使用一组Connection ID（连接ID）来标识一个连接
 - 即使IP或者端口发生变化，只要Connection ID没有变化，那么连接依然可以维持
 - 比如
 - ✓ 当设备连接到Wi-Fi时，将进行中的下载从蜂窝网络连接转移到更快速的Wi-Fi连接
 - ✓ 当Wi-Fi连接不再可用时，将连接转移到蜂窝网络连接

HTTP/3的问题 — 操作系统内核、CPU负载

- 据Google和Facebook称，与基于TLS的HTTP/2相比，它们大规模部署的QUIC需要近2倍的CPU使用量
 - Linux内核的UDP部分没有得到像TCP那样的优化，因为传统上没有使用UDP进行如此高速的信息传输
 - TCP和TLS有硬件加速，而这对于UDP很罕见，对于QUIC则基本不存在
- 随着时间的推移，相信这个问题会逐步得到改善