

第三部分：HTTP/2 协议



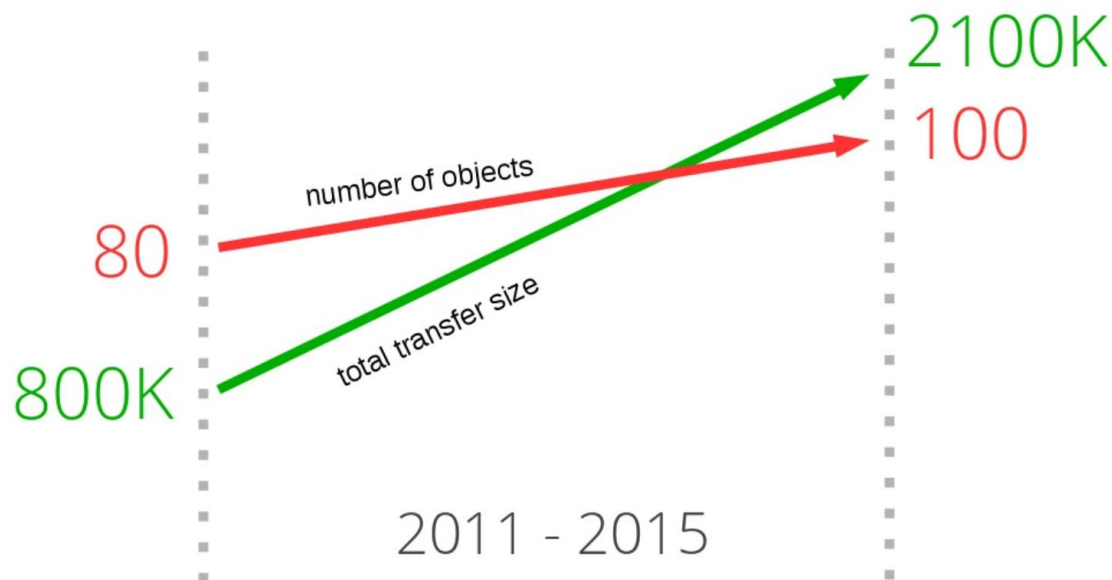
扫码试看/订阅极客时间

《Web协议详解与抓包实战》视频课程

第 1 课 HTTP/1.1 发展中遇到的问题

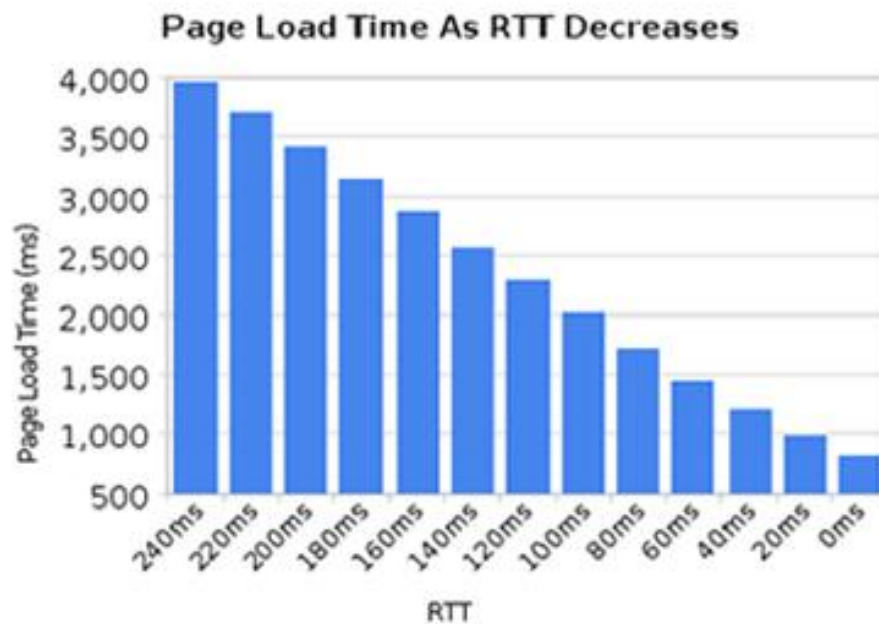
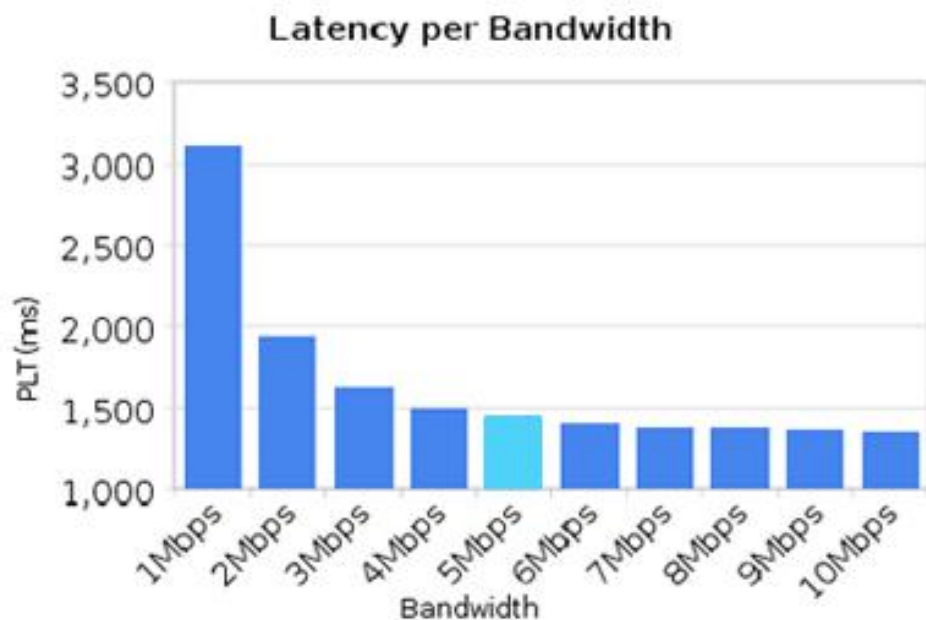
HTTP/1.1 发明以来发生了哪些变化？

- 从几 KB 大小的消息，到几 MB 大小的消息
- 每个页面小于 10 个资源，到每页面 100 多个资源
- 从文本为主的内容，到富媒体（如图片、声音、视频）为主的内容
- 对页面内容实时性高要求的应用越来越多



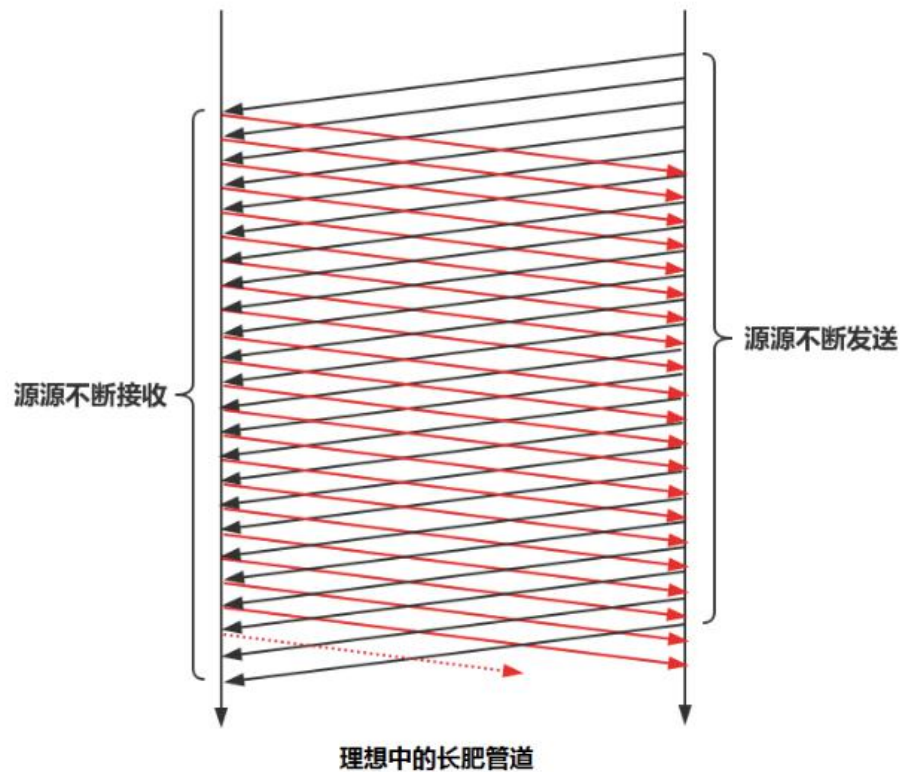
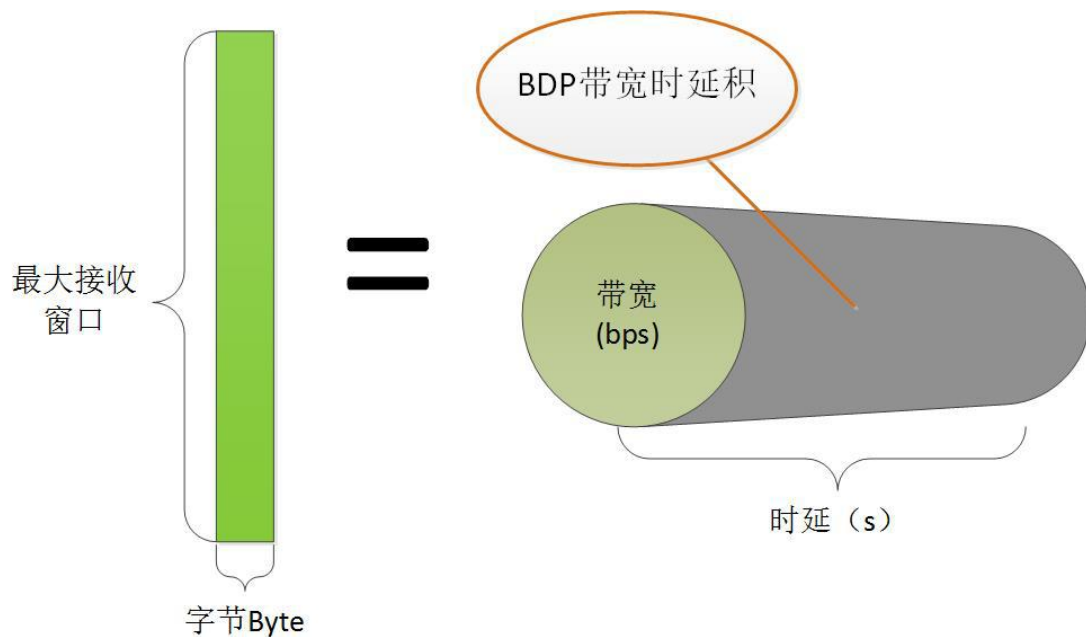
HTTP/1.1 的高延迟问题

- 高延迟带来页面加载速度的降低
 - 随着带宽的增加，延迟并没有显著下降
 - 并发连接有限
 - 同一连接同时只能在完成一个 HTTP 事务（请求/响应）才能处理下一个事务



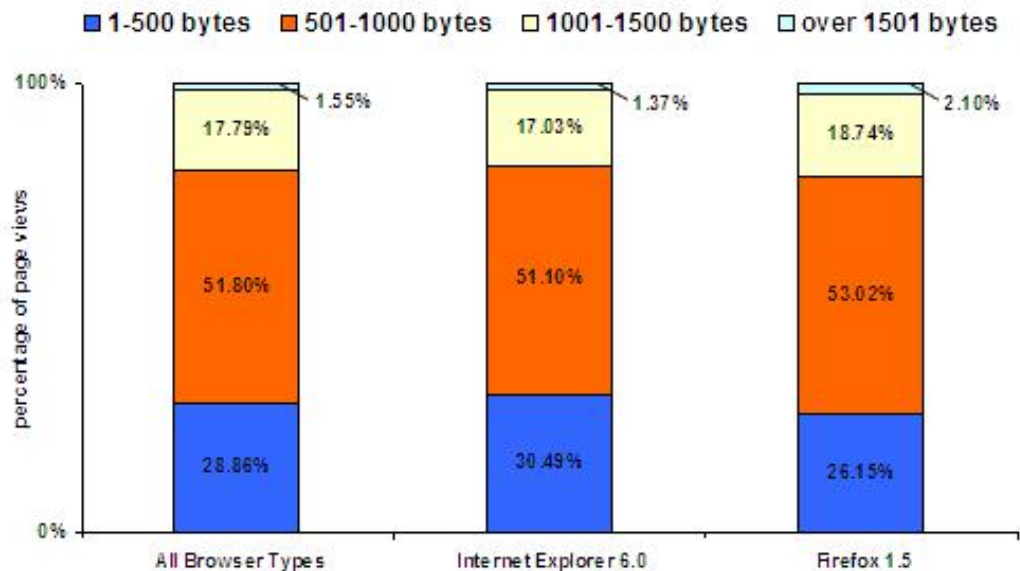
高延迟 VS 高带宽

- 单连接上的串行请求
- 无状态导致的高传输量（低网络效率）



无状态特性带来的巨大 HTTP 头部

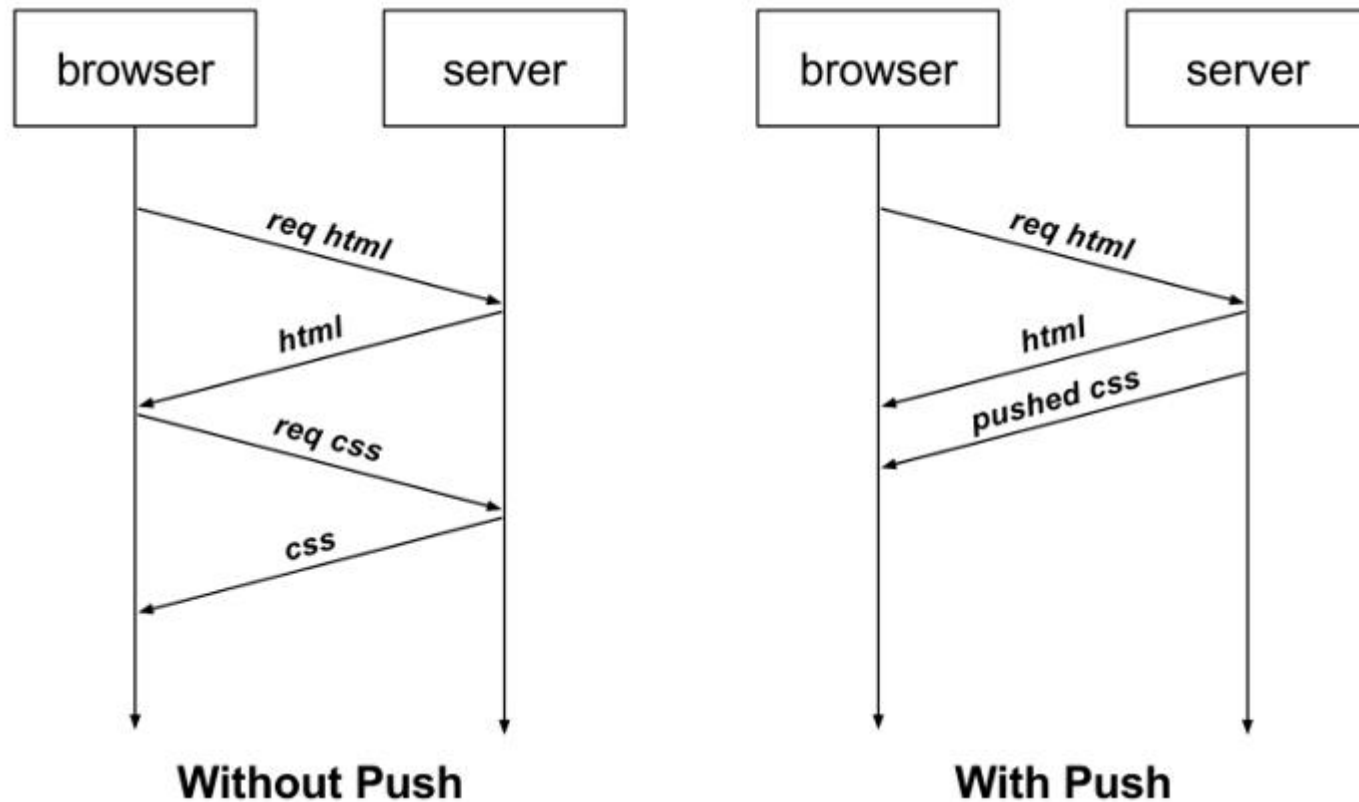
- 重复传输的体积巨大的 HTTP 头部



HTTP/1.1 为了解决性能问题做过的努力

- Spriting 合并多张小图为一张大图供浏览器 JS 切割使用
 - 不能区别对待
- Inlining 内联，将图片嵌入到 CSS 或者 HTML 文件中，减少网络请求次数
- Concatenation 拼接，将多个体积较小的 JavaScript 使用 webpack 等工具打包成 1 个体积更大的 JavaScript 文件
 - 1 个文件的改动导致用户重新下载多个文件
- Sharding 分片，将同一页面的资源分散到不同域名下，提升连接上限

HTTP/1.1 不支持服务器推送消息



第 2 课 HTTP/2 有哪些特性?

解决 HTTP/1 性能问题的 HTTP/2

- SPDY (2012-2016)
- HTTP2 (RFC7540, 2015.5)
 - 在应用层上修改，基于并充分挖掘 TCP 协议性能
 - 客户端向 server 发送 request 这种基本模型不会变。
 - 老的 scheme 不会变，没有 http2://。
 - 使用 http/1.x 的客户端和服务端可以无缝的通过代理方式转接到 http/2 上。
 - 不识别 http/2 的代理服务器可以将请求降级到 http/1.x

HTTP

SPDY

SSL

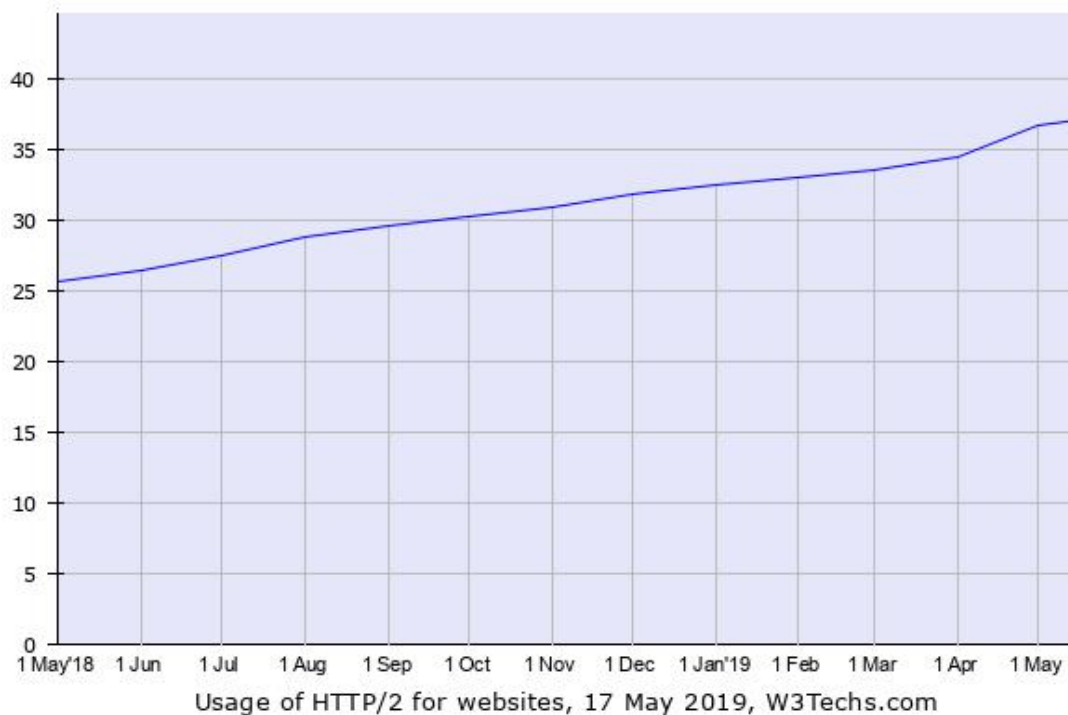
TCP

主流浏览器对 HTTP/2 的支持程度

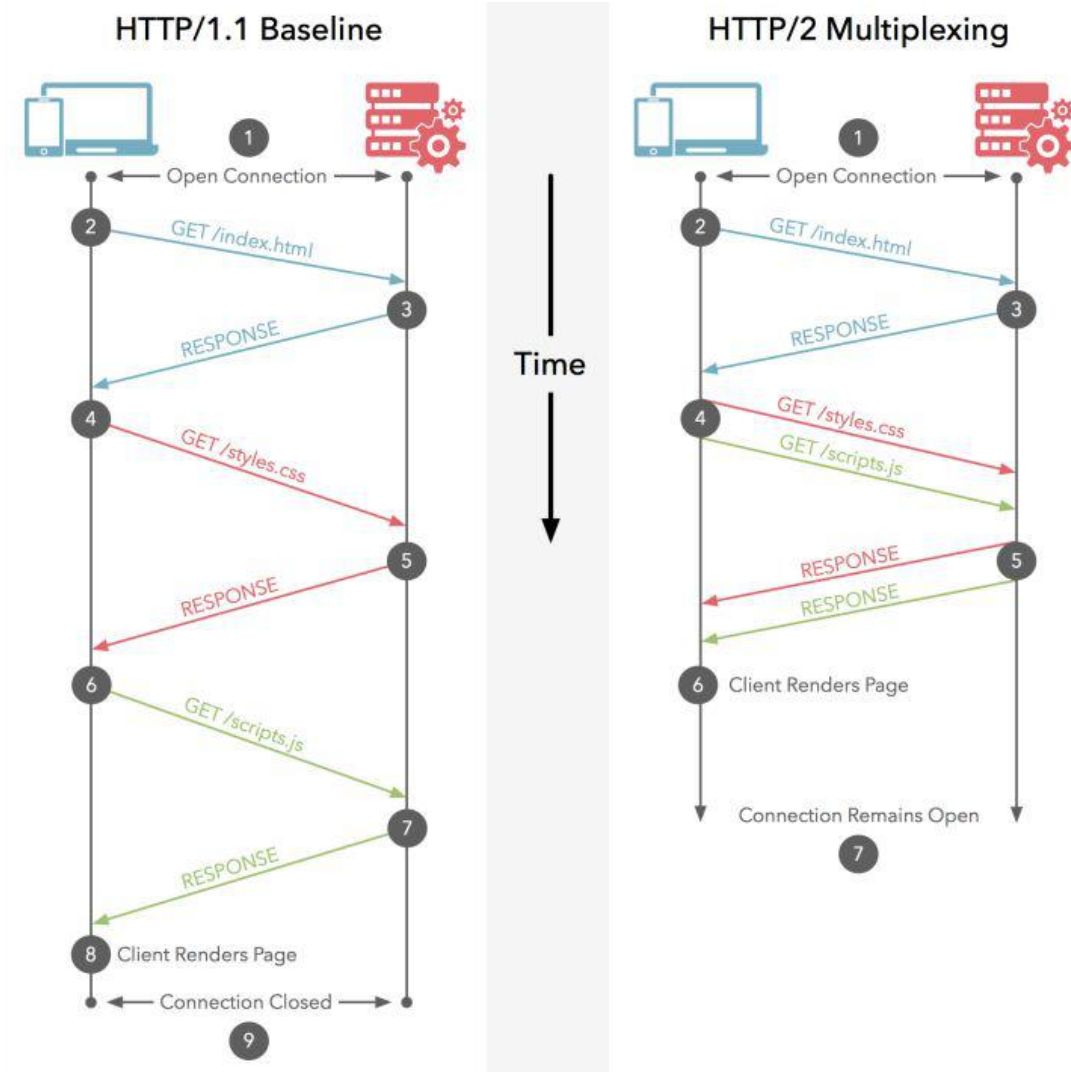
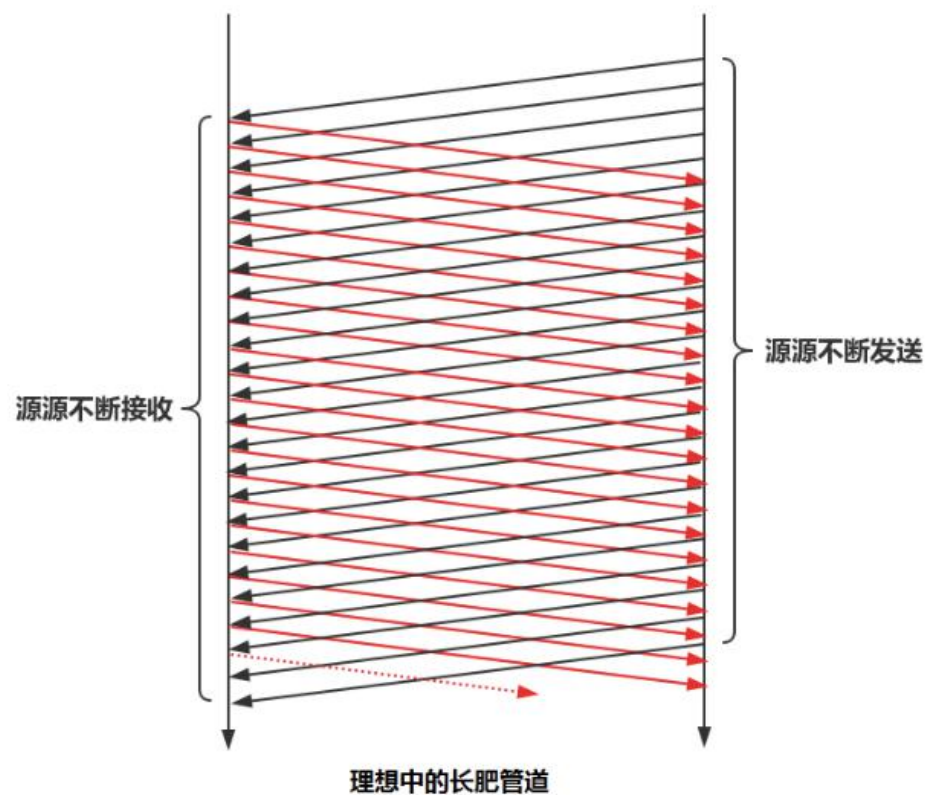
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49						
			51			9.2		4.4	
8	13	47	52			9.3		4.4.4	
11	14	48	53	9.1	39	10	all	52	51
		49	54	10	40				
		50	55	TP	41				
		51	56						

HTTP/2 的应用状况

- 截止 2019.5.17 号，互联网上使用 HTTP/2 协议的站点已经达到 37.2%
- 快速推广的原因
 - 未改变 HTTP/1.1 的语义
 - 基于 TCP，仅在应用层变动



多路复用带来的提升

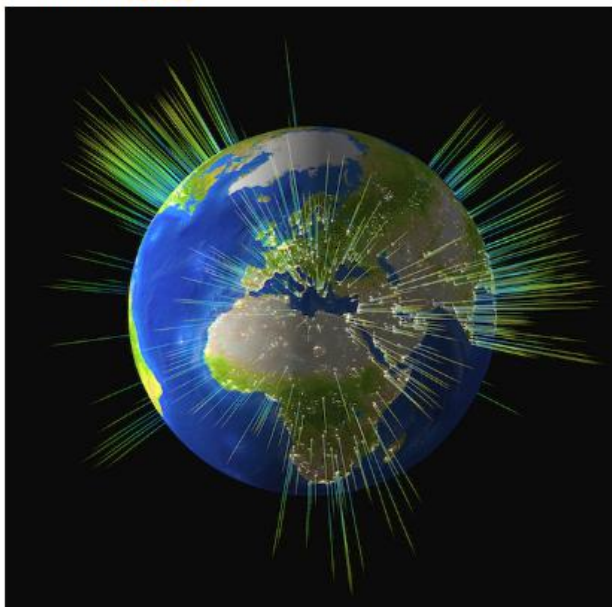


HTTP/2 的强大之处

- <https://http2.akamai.com/demo>

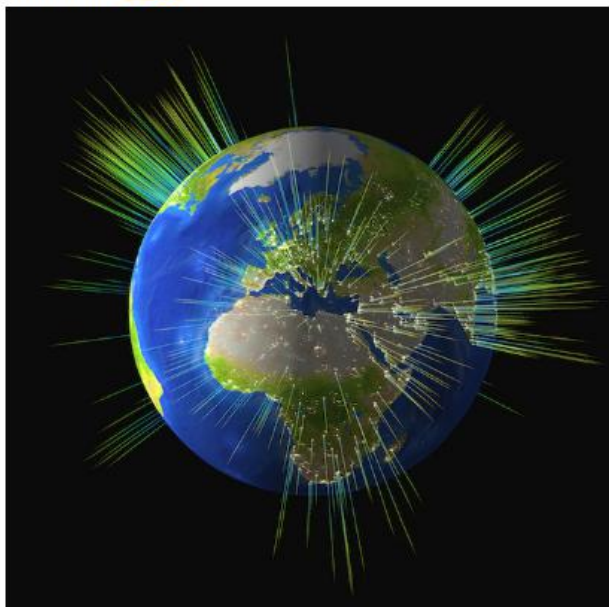
HTTP/1.1

Latency: 36ms
Load time: 14.18s



HTTP/2

Latency: 82ms
Load time: 2.28s



HTTP/2 主要特性

- 传输数据量的大幅减少
 - 以二进制方式传输
 - 标头压缩
- 多路复用及相关功能
 - 消息优先级
- 服务器消息推送
 - 并行推送

第 3 课 如何使用 Wireshark 解密 TLS/SSL 报文?

Chrome 浏览器检测 HTTP/2 插件

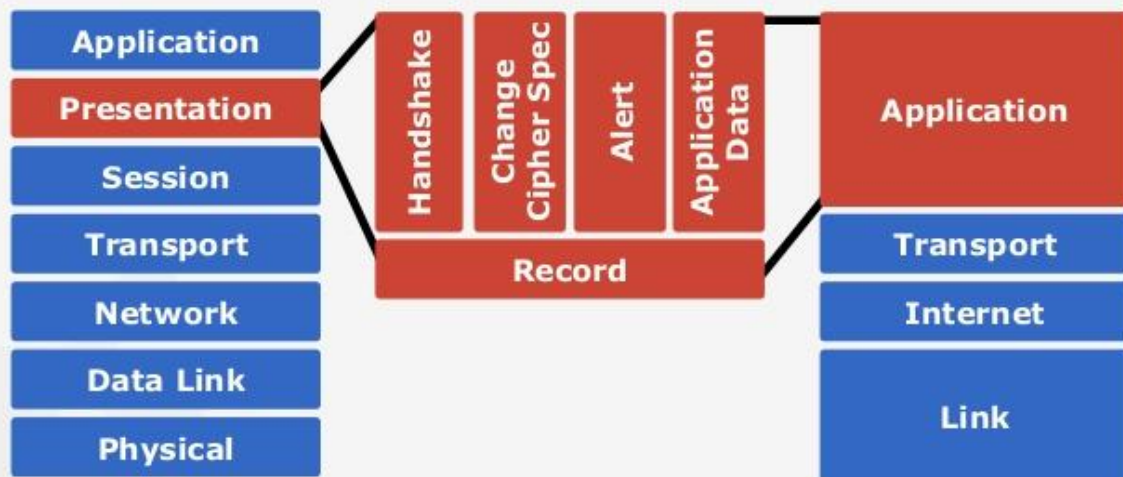
- HTTP/2 and SPDY indicator
 - <https://chrome.google.com/webstore/detail/http2-and-spdy-indicator/mpbpobfflnpcgagjijhmgnchggcjblin>

在 HTTP/2 应用层协议之下的 TLS 层

SSL/TLS in Common Models

ISO/OSI model

TCP/IP model



TLS1.2 的加密算法

- 常见加密套件



- 对称加密算法: AES_128_GCM

- 每次建立连接后, 加密密钥都不一样

- 密钥生成算法: ECDHE

- 客户端与服务器通过交换部分信息, 各自独立生成最终一致的密钥

Wireshark 如何解密 TLS 消息?

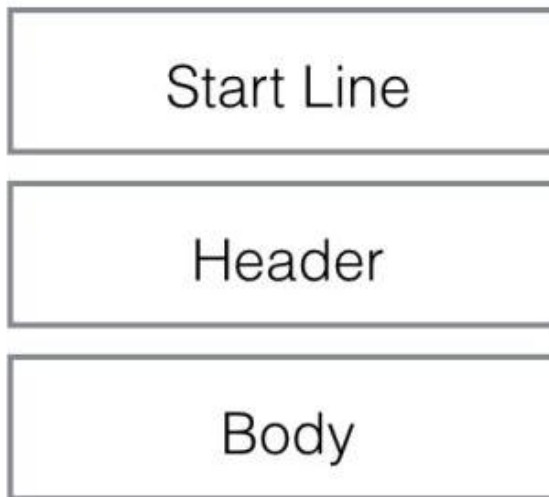
- 原理：获得 TLS 握手阶段生成的密钥
 - 通过 Chrome 浏览器 DEBUG 日志中的握手信息生成密钥
- 步骤
 - 配置 Chrome 输出 DEBUG 日志
 - 配置环境变量 SSLKEYLOGFILE
 - 在 Wireshark 中配置解析 DEBUG 日志
 - 编辑->首选项->Protocols->TLS/SSL
 - (Pre)-Master-Secret log filename

```
CLIENT_RANDOM afc4de62f9507f5783
ecf4bd78fd429c6ca6077a70afe94bad
7d
CLIENT_RANDOM 524de51b66f0d1d6e6
82fbc3e7fbf214cb408a4ded1f19aaf4
ea
CLIENT_HANDSHAKE_TRAFFIC_SECRET
d26d9369722978831a0d08ca0834878c
201e5a227b742b9ef41e0dee064b9f5c
SERVER_HANDSHAKE_TRAFFIC_SECRET
d26d9369722978831a0d08ca0834878c
b957b59ad738d46652723600113e3df1
```

二进制格式与可见性

- TLS/SSL 降低了可见性门槛
 - 代理服务器没有私钥不能看到内容

HTTP/1.x



HTTP/2



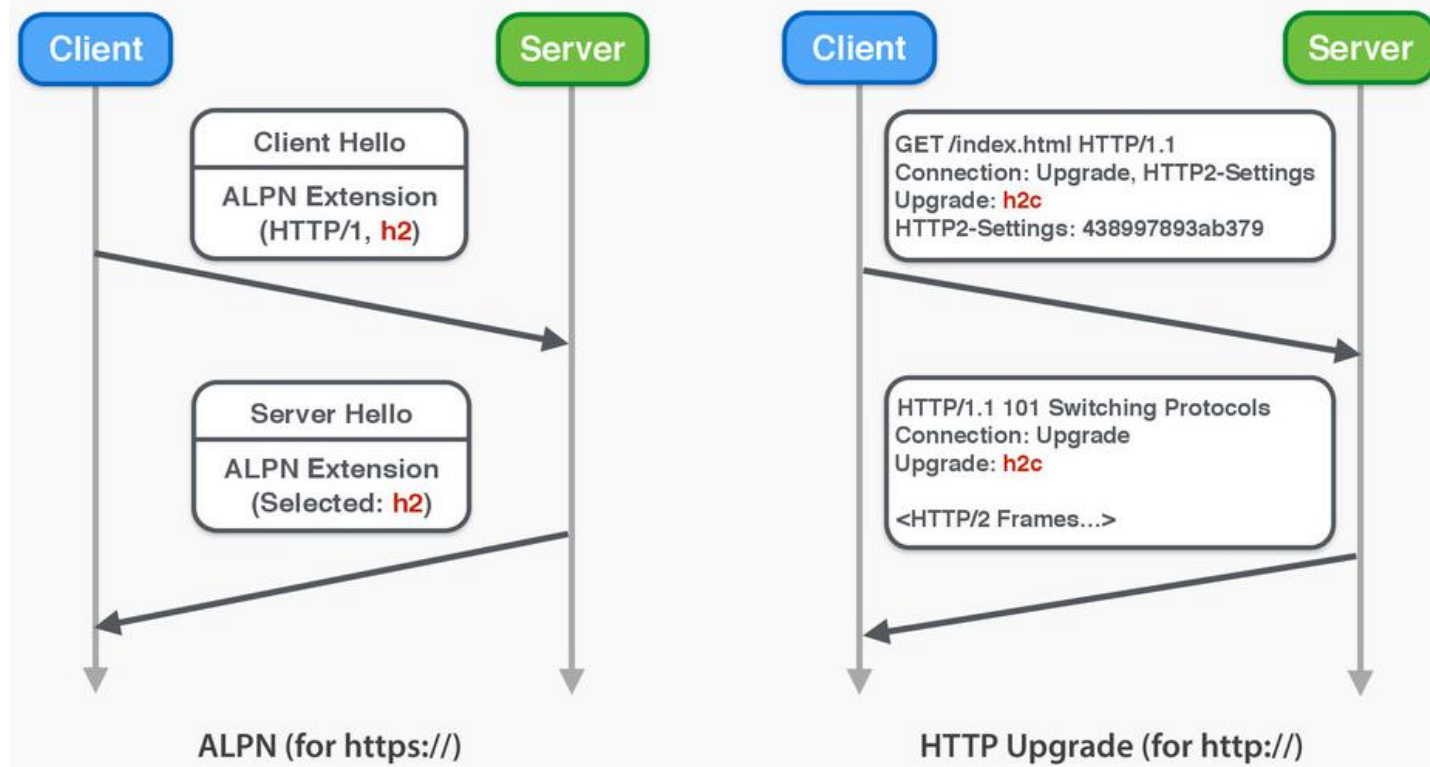
第 4 课 h2c: 如何从 http://升级到 HTTP/2 协议?

HTTP/2 是不是必须基于 TLS/SSL 协议?

- IETF 标准不要求必须基于TLS/SSL协议
- 浏览器要求必须基于TLS/SSL协议
- 在 TLS 层 ALPN (Application Layer Protocol Negotiation)扩展做协商, 只认 HTTP/1.x 的代理服务器不会干扰 HTTP/2
- shema: http://和 https:// 默认基于 80 和 443 端口
- h2: 基于 TLS 协议运行的 HTTP/2 被称为 h2
- h2c: 直接在 TCP 协议之上运行的 HTTP/2 被称为 h2c

h2 与 h2c

Protocol Negotiation



H2C: 不使用 TLS 协议进行协议升级 (1)

- 客户端测试工具: curl (7.46.0版本)
 - curl http://nghttp/2.org -http/2 -v

```
GET / HTTP/1.1\r\n
Host: nghttp2.org\r\n
User-Agent: curl/7.46.0\r\n
Accept: */*\r\n
Connection: Upgrade, HTTP2-Settings\r\n
Upgrade: h2c\r\n
HTTP2-Settings: AAMAAABkAAQAAP__\r\n
\r\n
```

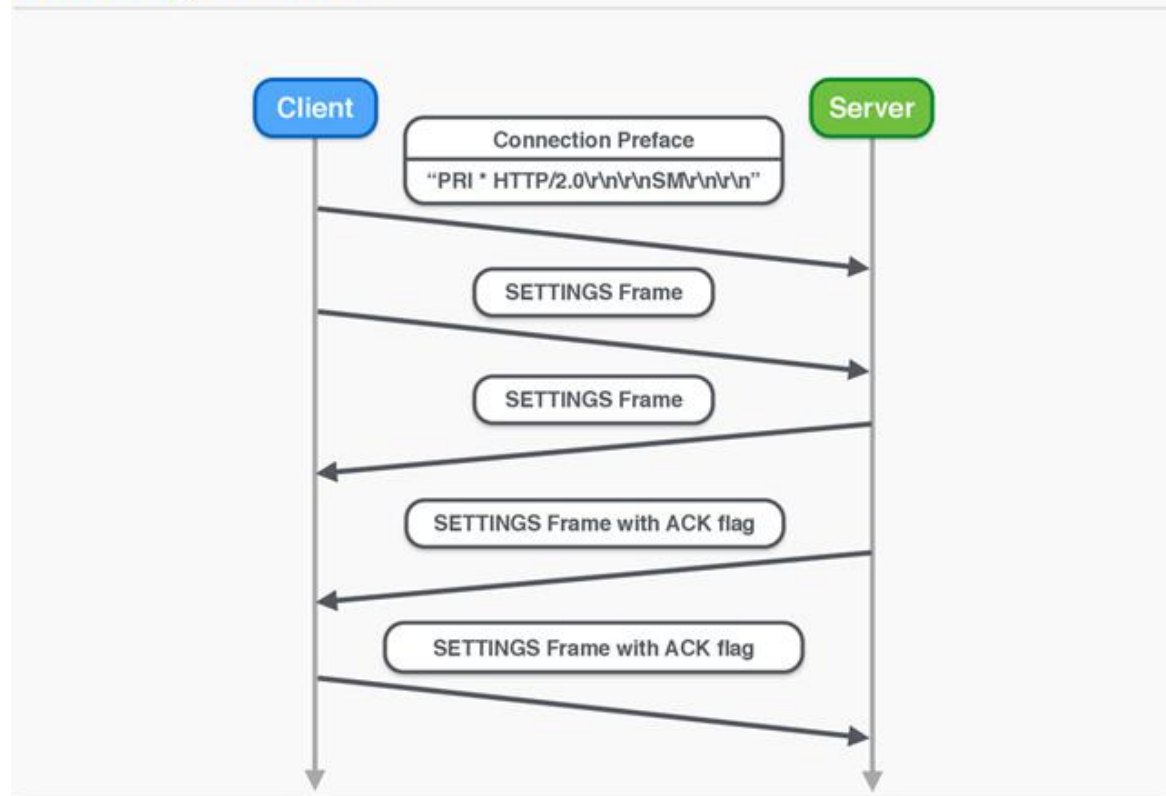
```
HTTP/1.1 101 Switching Protocols\r\n
Connection: Upgrade\r\n
Upgrade: h2c\r\n
\r\n
```

H2C: 客户端发送的 Magic 帧

- Preface (ASCII 编码, 12字节)
 - 何时发送?
 - 接收到服务器发送来的 101 Switching Protocols
 - TLS 握手成功后
 - Preface 内容
 - 0x505249202a20485454502f322e300d0a0d0a534d0d0a0d0a
 - PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
 - 发送完毕后, 应紧跟 SETTING 帧

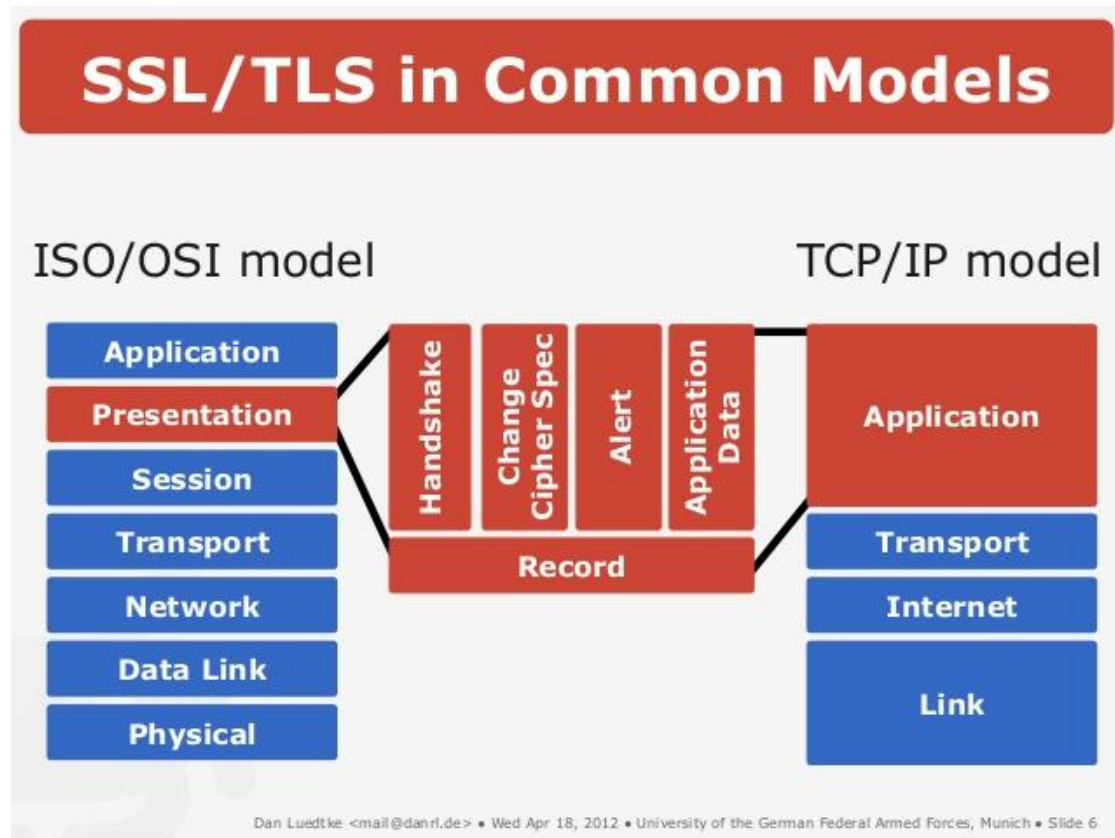
统一的连接过程

Starting HTTP/2



第 5 课 h2: 如何从 https://升级到 HTTP/2 协议?

在 HTTP/2 应用层协议之下的 TLS 层



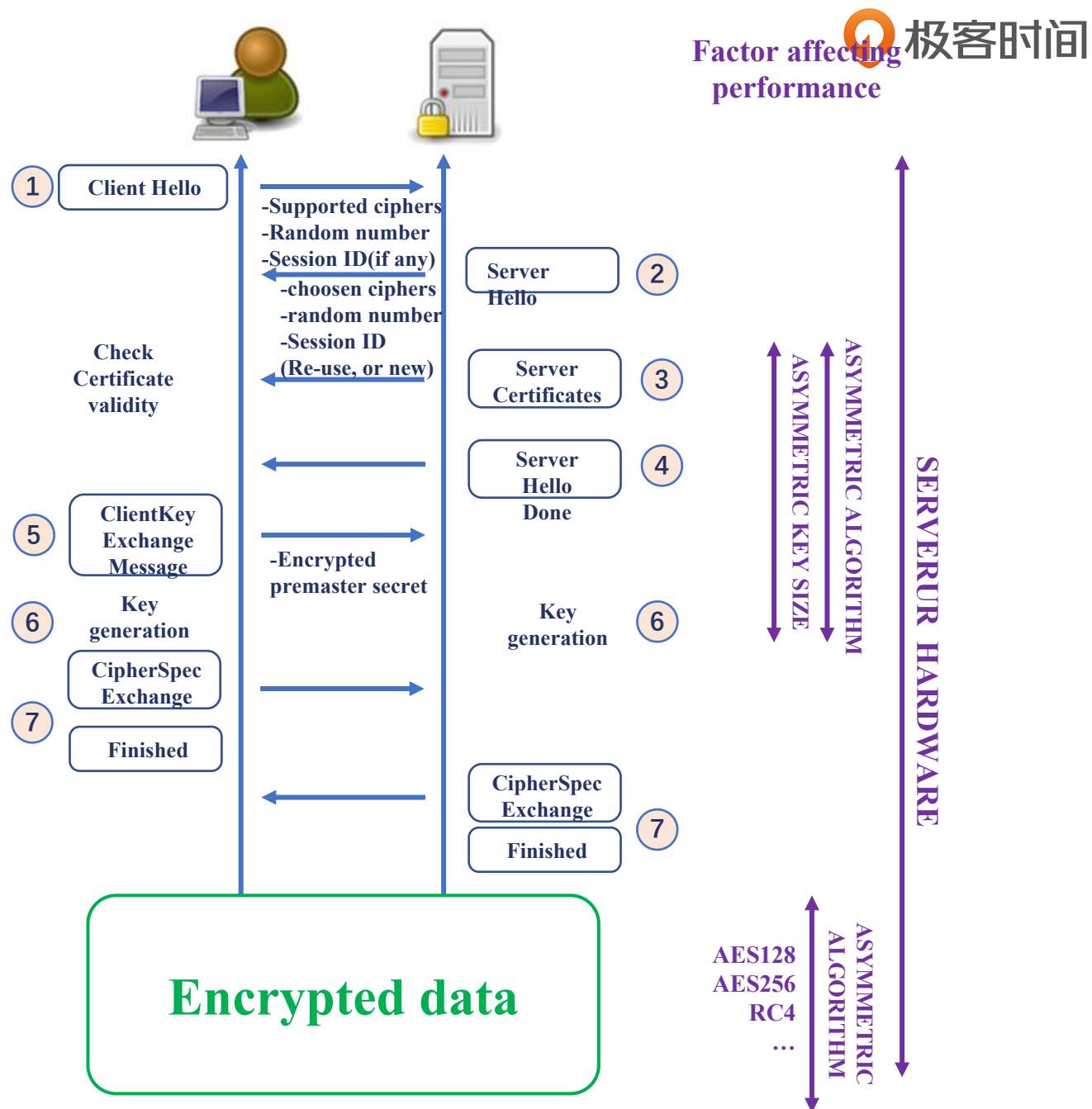
TLS 通讯过程

验证身份

达成安全套件共识

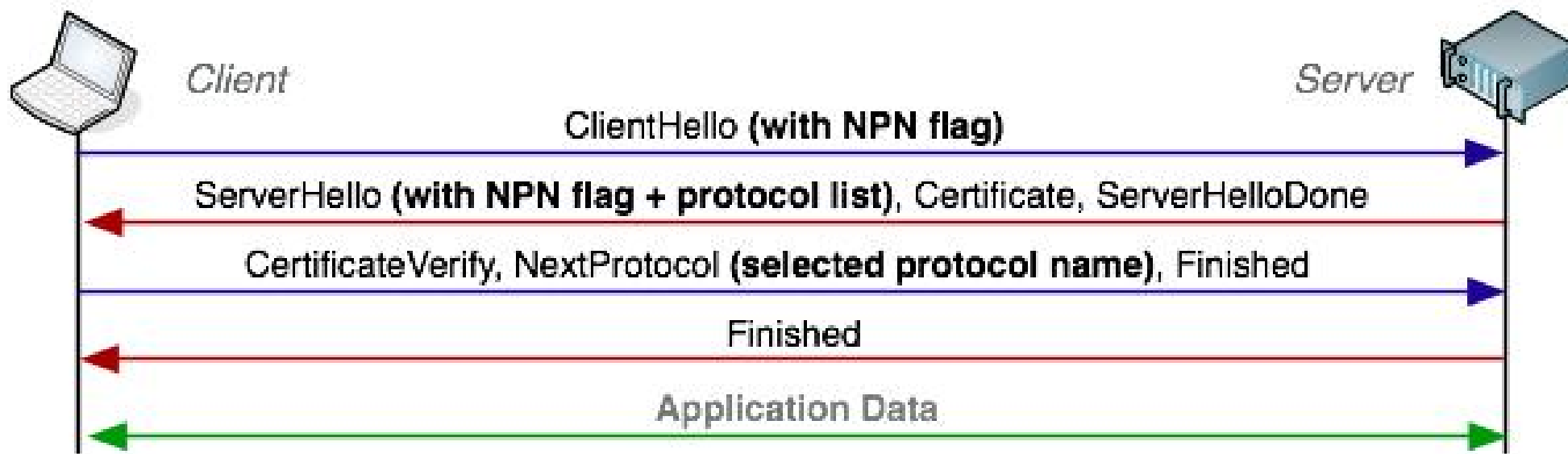
传递密钥

加密通讯



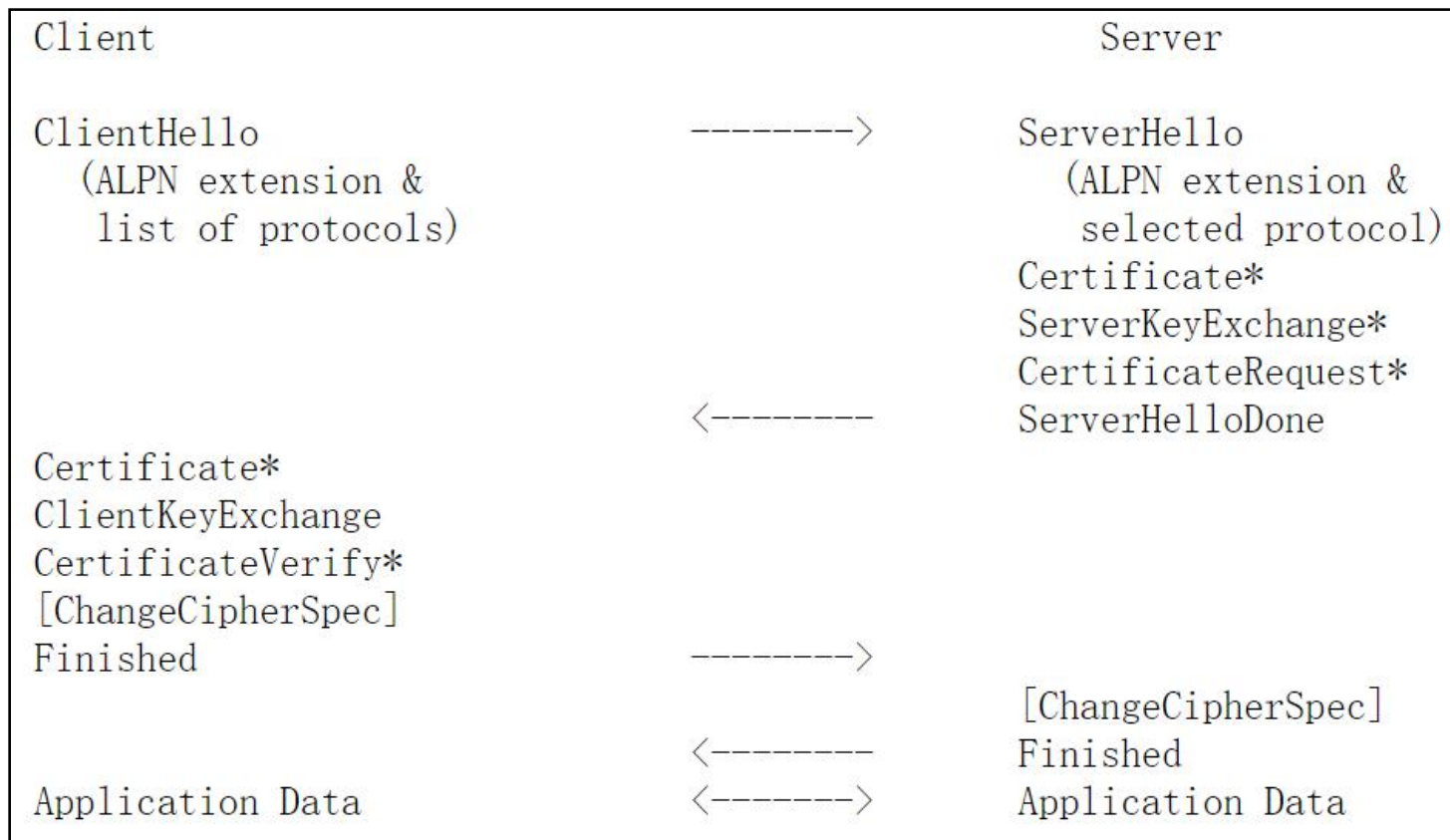
Next Protocol Negotiation (NPN)

- SPDY 使用的由客户端选择协议的 NPN 扩展



Application-Layer Protocol Negotiation Extension

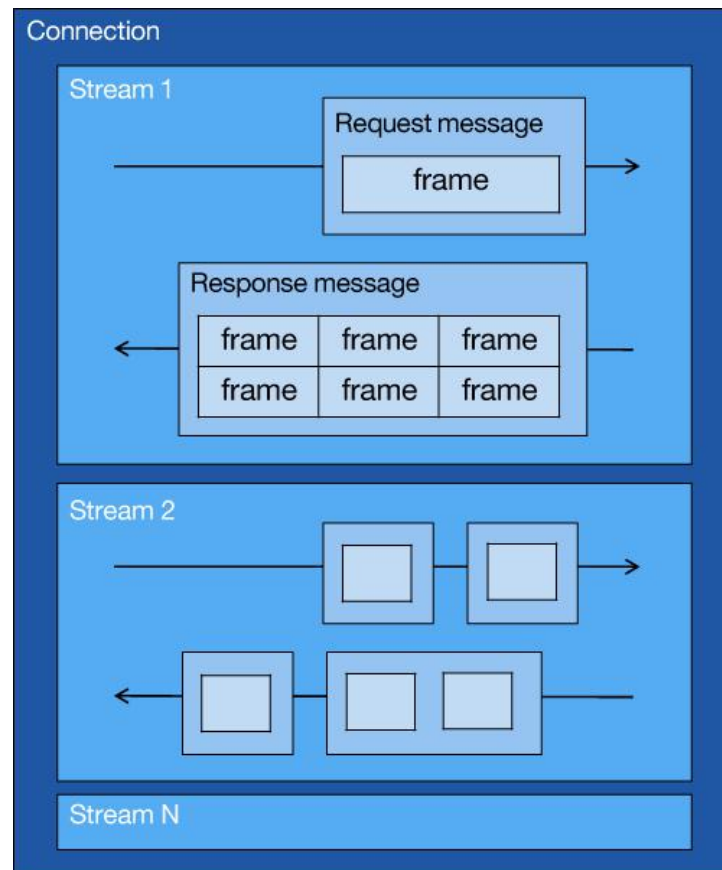
- RFC7301



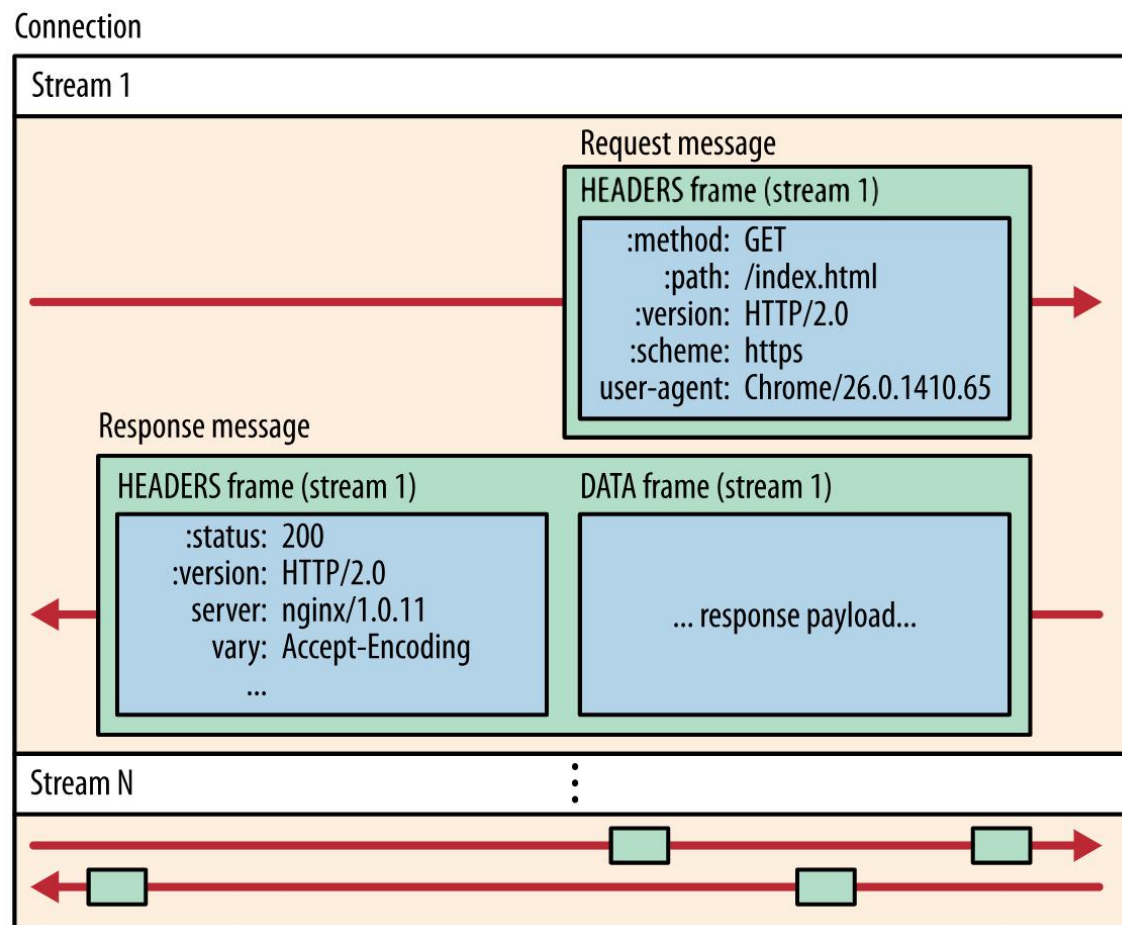
第 6 课 Stream、Message、Frame 间的关系

HTTP/2 核心概念

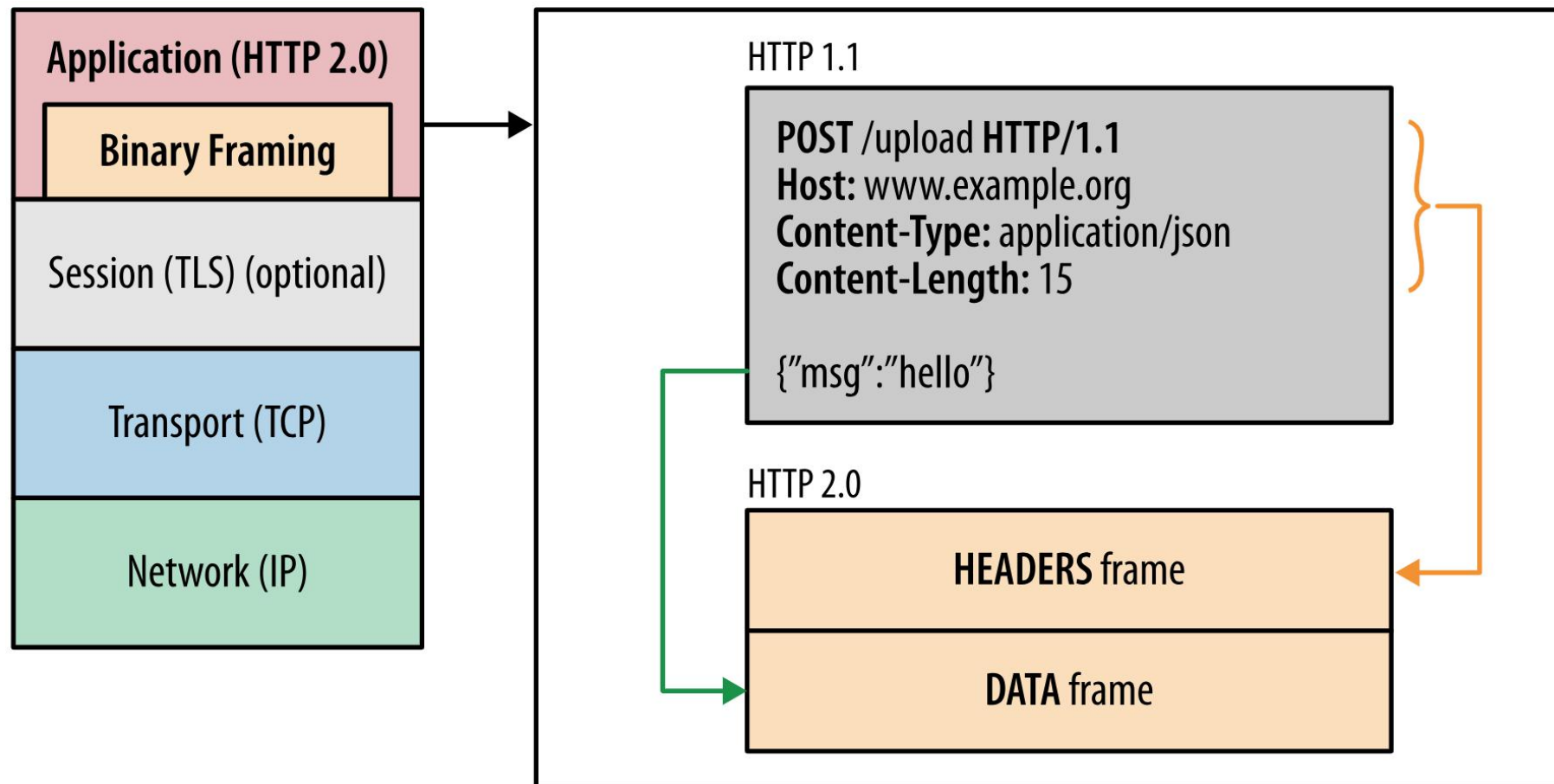
- 连接 Connection: 1个 TCP 连接, 包含一个或者多个 Stream
- 数据流 Stream: 一个双向通讯数据流, 包含 1 条或者多条 Message
- 消息 Message: 对应 HTTP/1 中的请求或者响应, 包含一条或者多条 Frame
- 数据帧 Frame: 最小单位, 以二进制压缩格式存放 HTTP/1 中的内容



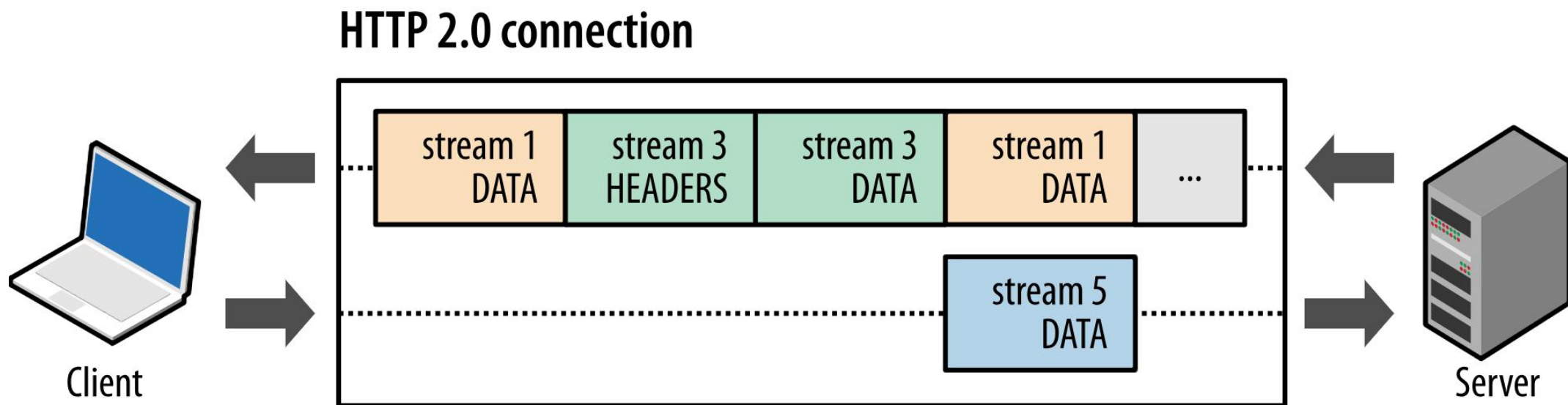
Stream、Message、Frame 间的关系



消息的组成：HEADERS 帧与 DATA 帧

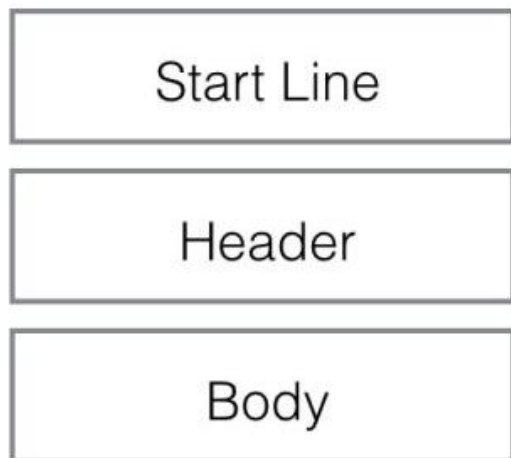


传输中无序，接收时组装

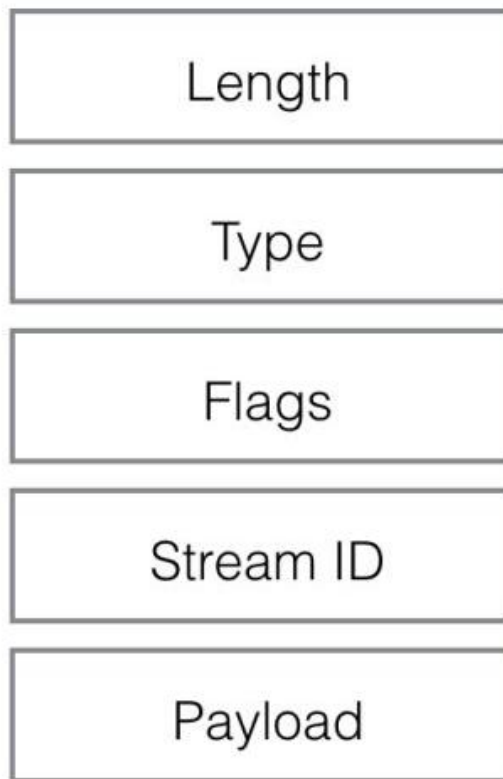


消息与帧

HTTP/1.x

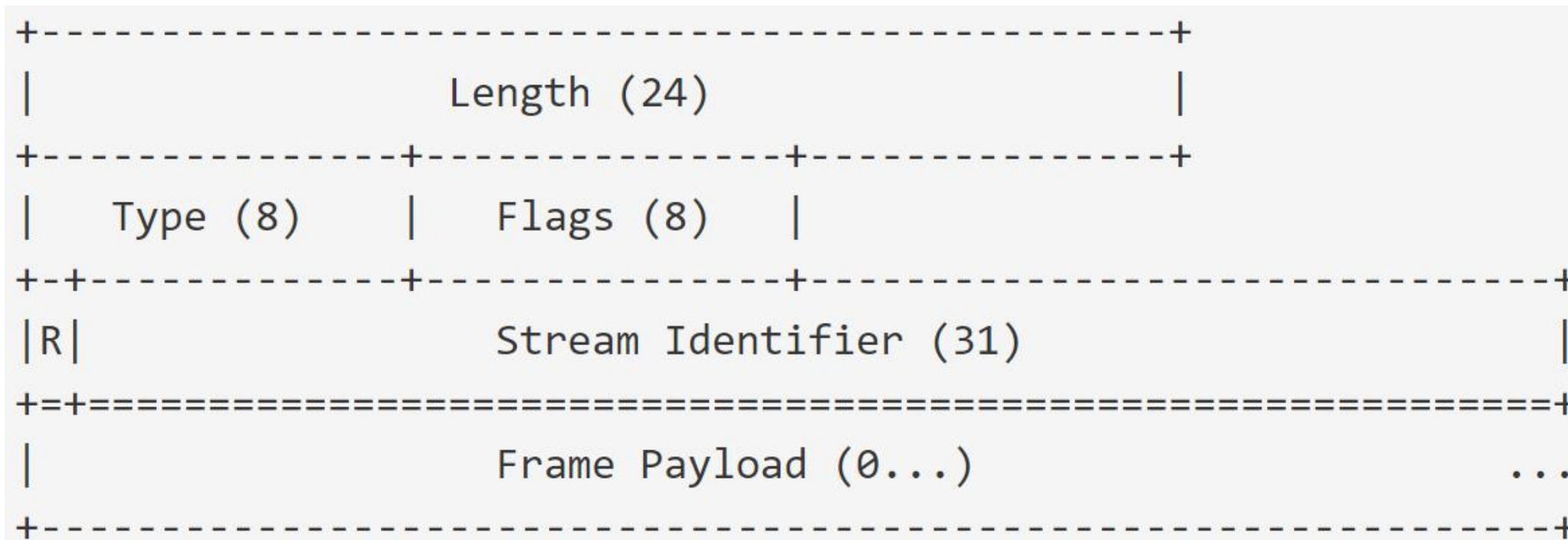


HTTP/2



第 7 课 消息帧的格式

9 字节标准帧头部



标准帧头与 HTTP



```
▼ HyperText Transfer Protocol 2
  ▼ Stream: HEADERS, Stream ID: 1, Length 198, 200 OK
    Length: 198
    Type: HEADERS (1)
    > Flags: 0x04
    0... .. = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
    Header Block Fragment: 886196c361be940baa681fa50400bea01ab8005c1
    [Header Length: 401]
    [Header Count: 13]
    > Header: :status: 200 OK
    > Header: date: Fri, 17 May 2019 04:00:23 GMT
    > Header: content-type: text/html
    > Header: last-modified: Thu, 18 Apr 2019 06:19:33 GMT
    > Header: etag: "5cb816f5-19d8"
    > Header: accept-ranges: bytes
    > Header: content-length: 6616
    > Header: x-backend-header-rtt: 0.024989
    .. ..
```

Frame 帧的类型



TYPE 类型:

- HEADERS: 帧仅包含 HTTP 标头信息。
- DATA: 帧包含消息的所有或部分有效负载。
- PRIORITY: 指定分配给流的重要性。
- RST_STREAM: 错误通知: 一个推送承诺遭到拒绝。终止流。
- SETTINGS: 指定连接配置。
- PUSH_PROMISE: 通知一个将资源推送到客户端的意图。
- PING: 检测信号和往返时间。
- GOAWAY: 停止为当前连接生成流的停止通知。
- WINDOW_UPDATE: 用于管理流的流控制。
- CONTINUATION: 用于延续某个标头碎片序列。

帧长度 Length

- 0 至 2^{14} (16,384) -1
 - 所有实现必须可以支持 16KB 以下的帧
- 2^{14} (16,384) 至 $2^{24}-1$ (16,777,215)
 - 传递 16KB 到 16MB 的帧时，必须接收端首先公布自己可以处理此大小
 - 通过 SETTINGS_MAX_FRAME_SIZE 帧 (Identifier=5) 告知

Setting 设置帧格式

- type=0x4
- 设置帧并不是“协商”，而是发送方向接收方通知其特性、能力

