

# 计算机网络

## 第二章 应用层协议及网络编程

徐敬东 张建忠

[xujd@nankai.edu.cn](mailto:xujd@nankai.edu.cn)

[zhangjz@nankai.edu.cn](mailto:zhangjz@nankai.edu.cn)

计算机网络与信息安全研究室

**总体目标：**理解应用层协议与进程通信模型，掌握Socket编程方法，学习典型的应用层协议

理解客户/服务器模型和对等计算模型，初步了解传输层服务及对应用层的支持

掌握基于套接字的网络编程方法，理解数据报式套接字和流式套接字的功能

掌握域名系统构成和解析过程，理解根域名服务器在国家网络基础设施中的重要作用

掌握Web服务的特点、HTTP 1.0和1.1的工作机制及所面临的性能问题，以及HTTP/2的优化机制和解决的关键问题

掌握内容分发网络所解决的问题和基本工作机制，理解两种基本的重定向方法，了解理解动态自适应流媒体协议的基本思想

2.1 应用协议与进程通信模型

2.2 传输层服务对应用的支持

2.3 Socket编程

2.4 域名系统

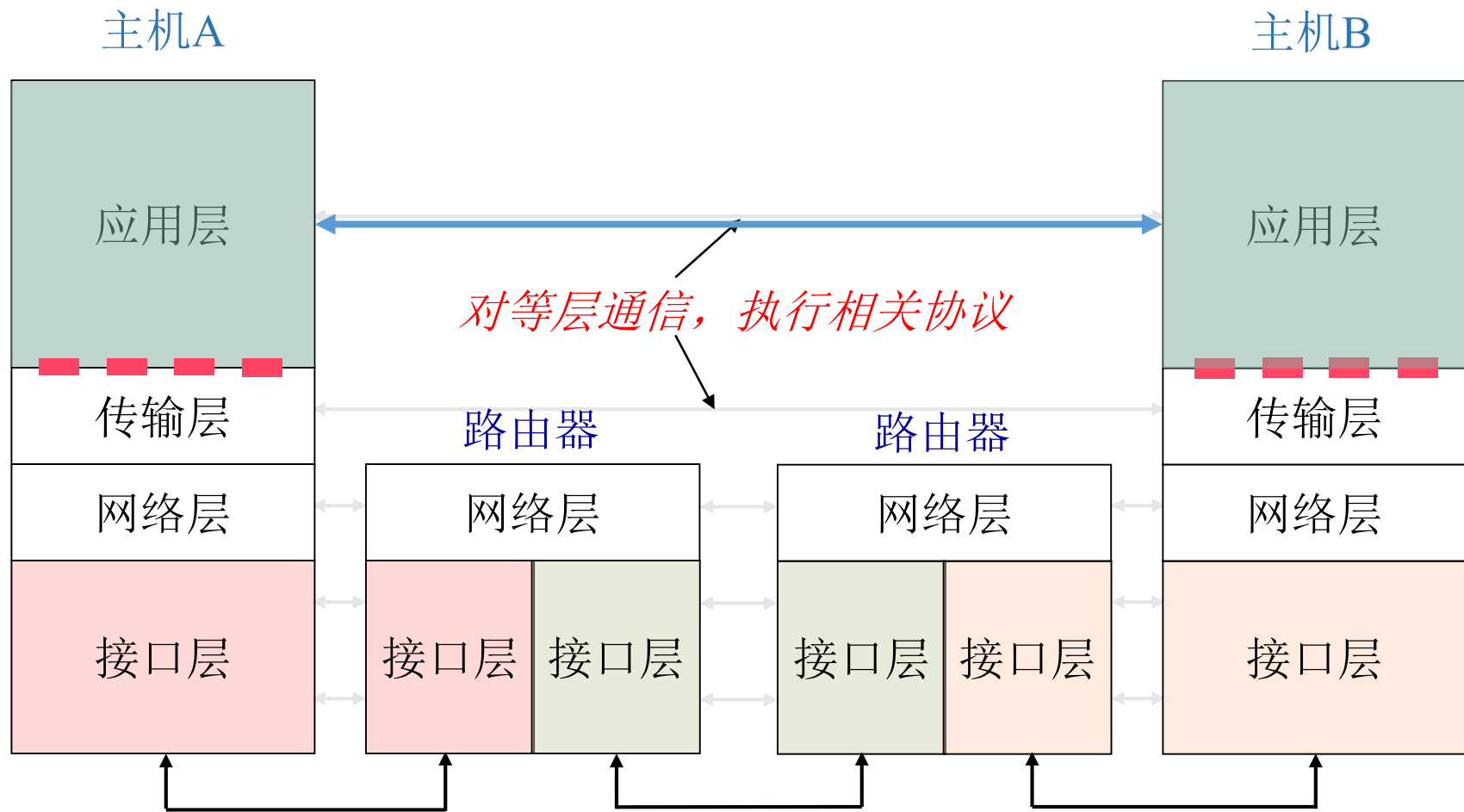
2.5 传统的应用层服务与协议

2.6 Web服务与HTTP协议

2.7 内容分发网络CDN

2.8 动态自适应流媒体协议DASH

## TCP/IP体系结构



接口层通常包括数据链路层和物理层

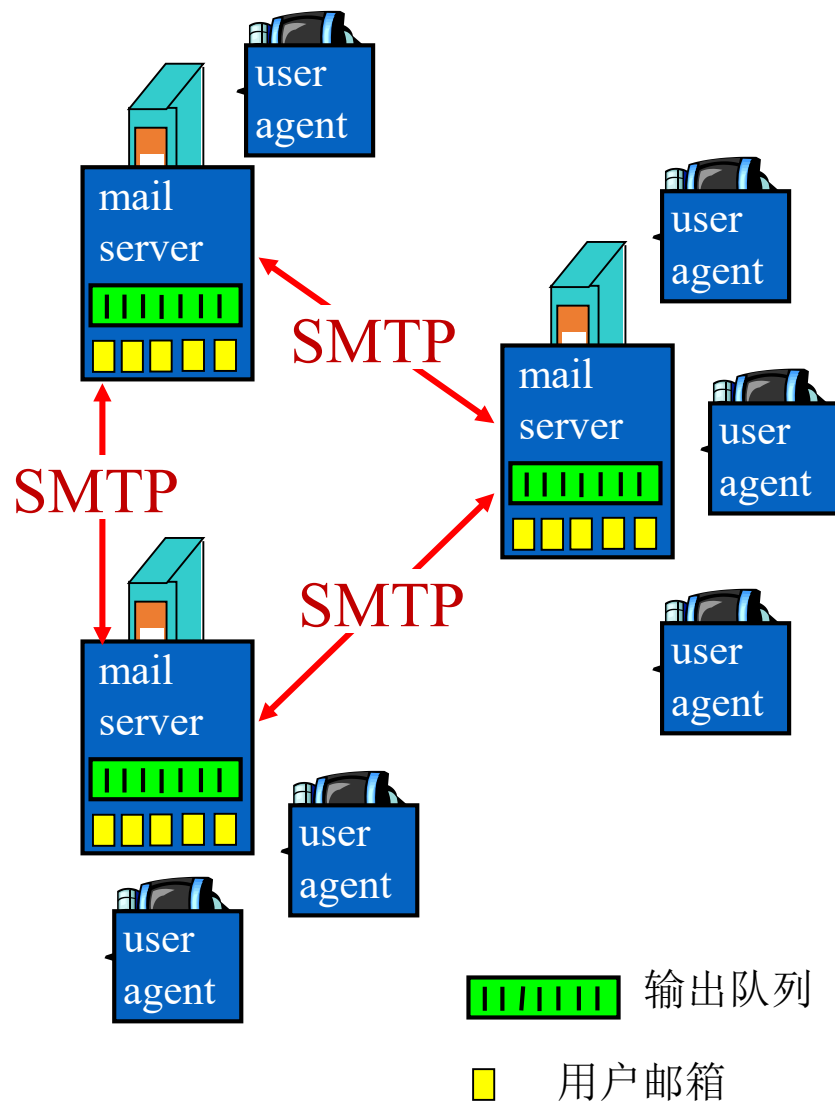
### 电子邮件系统

#### 用户代理 (接口)

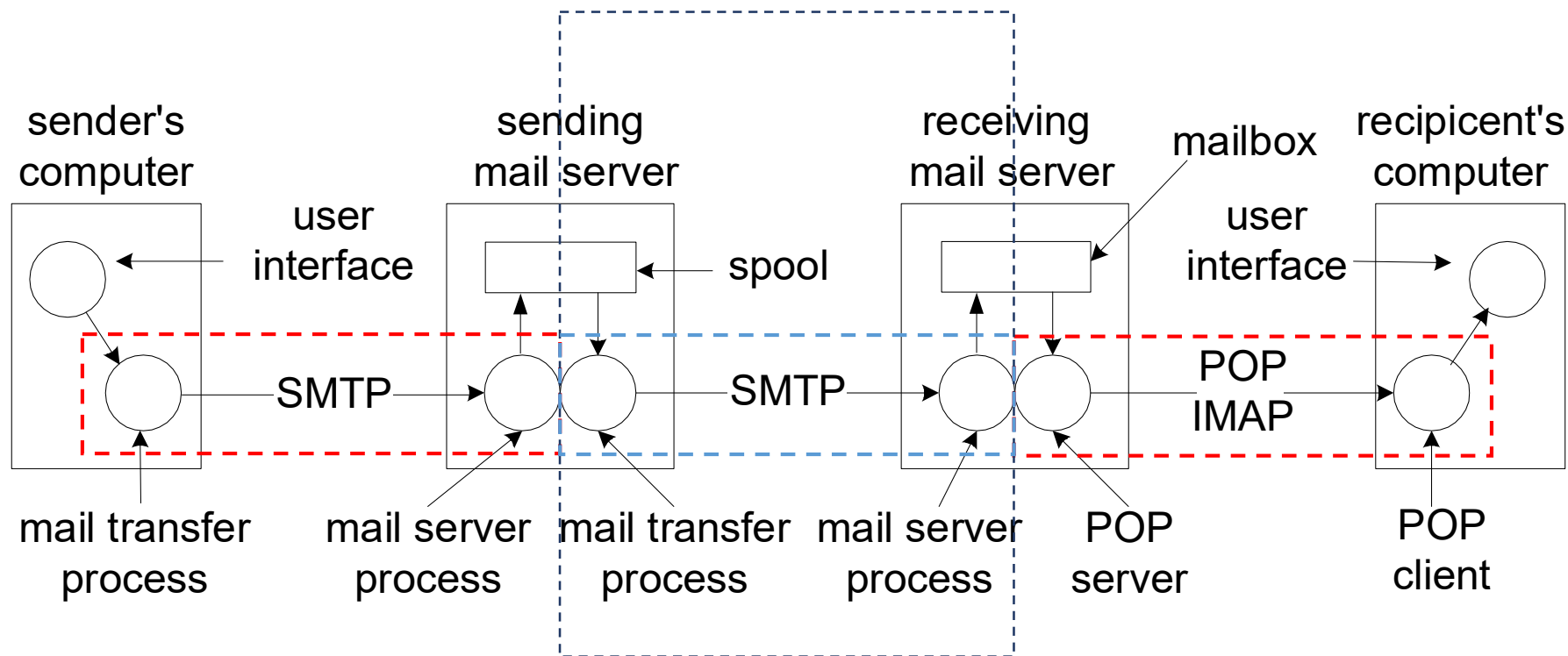
- 编辑和发送邮件
- 接收、读取和管理邮件
- 无统一标准

#### 邮件服务器

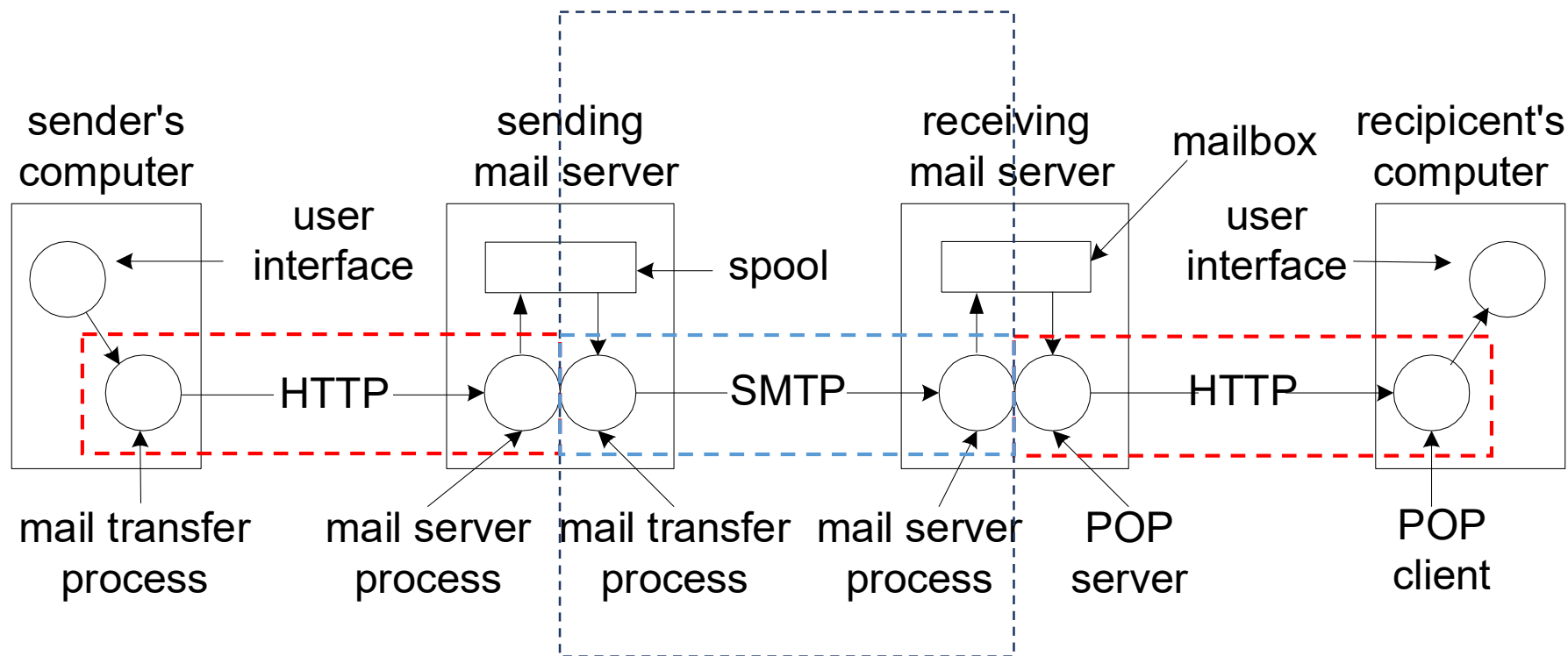
- **邮箱**：保存用户收到的消息
- **消息输出队列**：消息的发送队列
- **SMTP协议**：邮件服务器之间传递邮件使用的协议
  - smtp客户：发送邮件端
  - smtp服务器：接收邮件端



### 客户/服务器模型 (SMTP/POP/IMAP)



### 客户/服务器模型 (SMTP/HTTP)

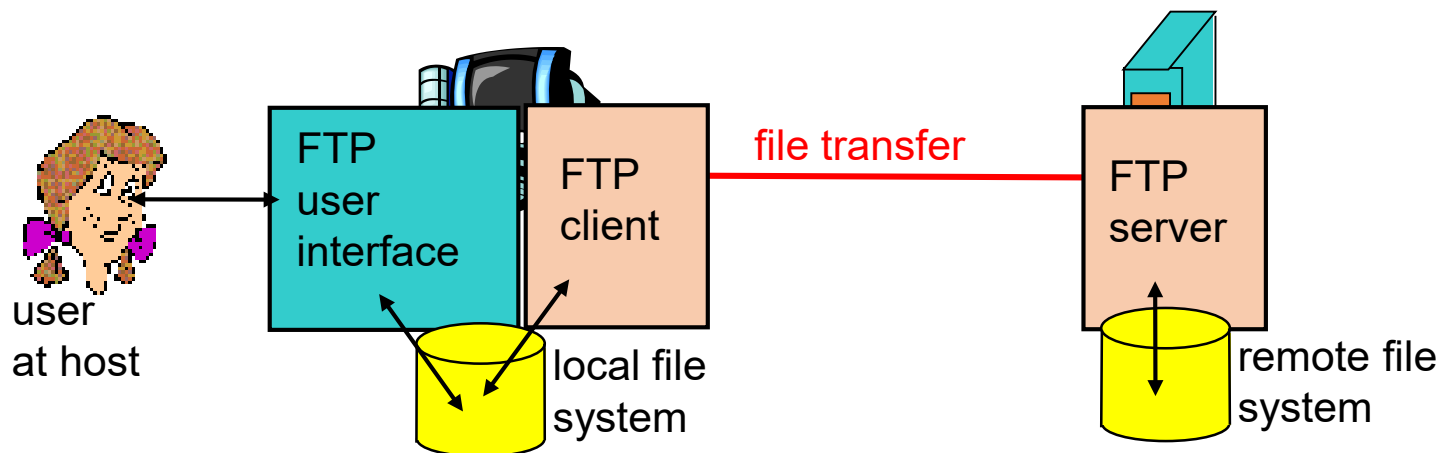


### SMTP: 简单邮件传输协议

- 利用TCP可靠地从SMTP客户向SMTP服务器传递邮件
- SMTP的3个阶段：连接建立、邮件传送、连接关闭
- 命令/响应
  - 命令：ASCII字符串
  - 响应：状态码+短语
- 消息为7位ASCII
- RFC 821: SMTP

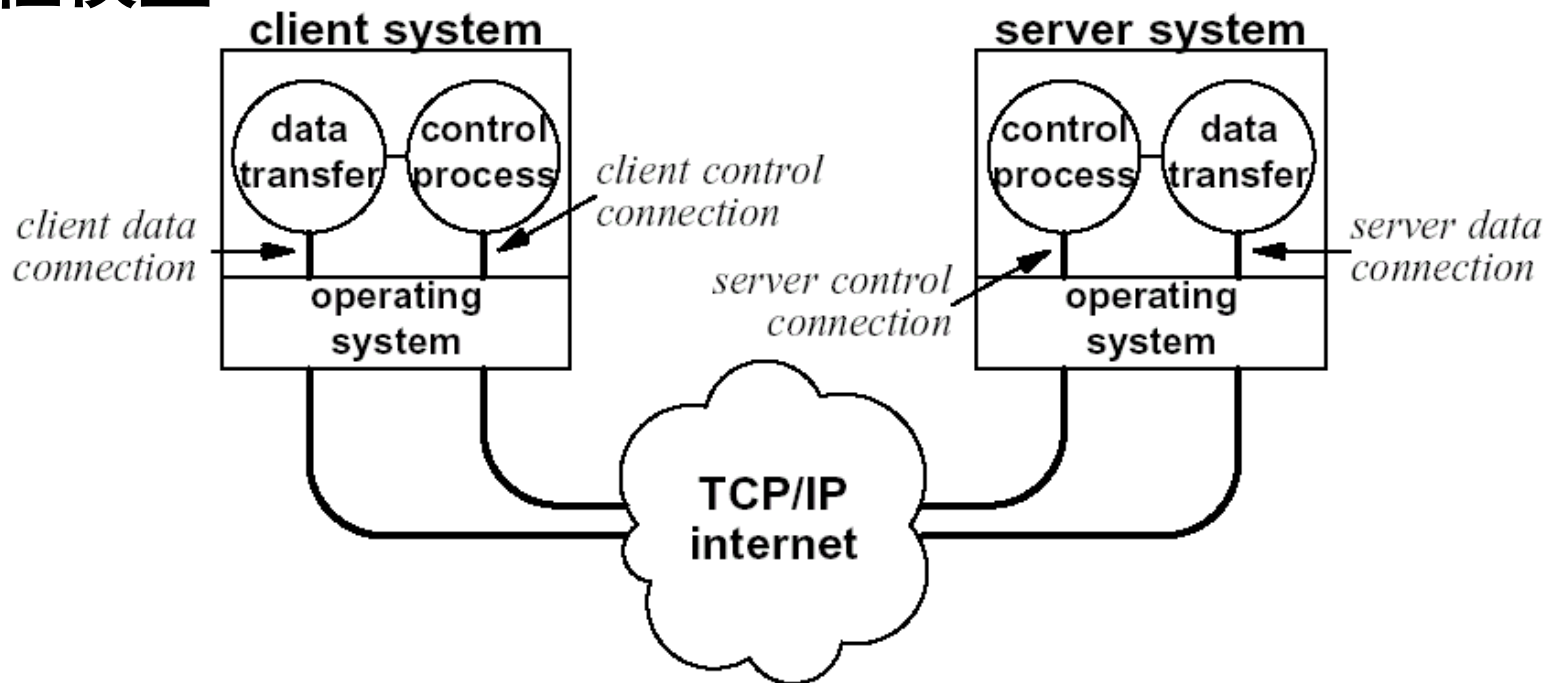


### FTP: 文件传输协议 (RFC 959)



- 基于TCP/IP的文件传输系统
- 客户/服务器模型
  - *client*: 初始化传输（无论上传还是下载）
  - *server*: 远端
- 客户使用TCP协议连接远端服务器

### FTP进程模型



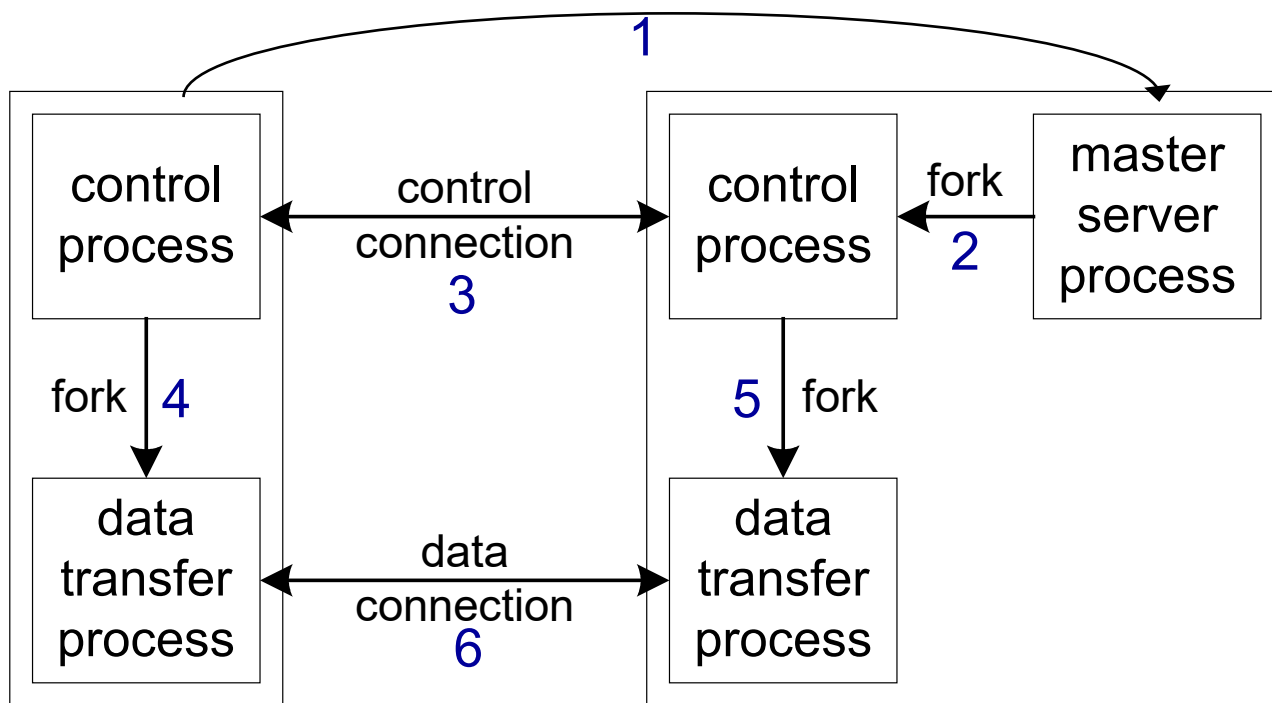
#### 控制连接

- 带外控制
- TCP的21端口
- 客户和服务端之间维护的一个FTP会话
- 用于客户和服务端之间交换命令和响应
- 在整个会话期间保持活跃

#### 数据传输连接

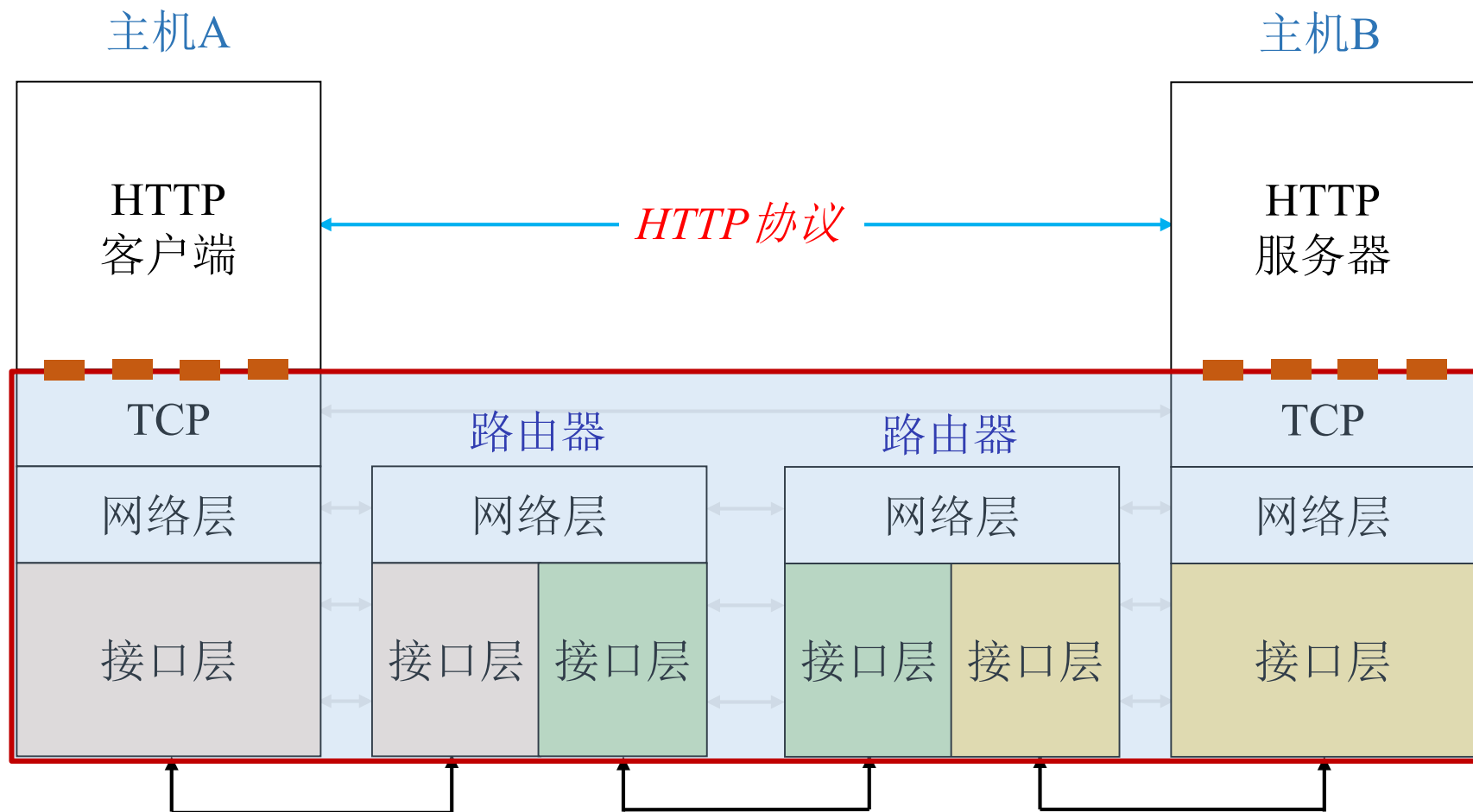
- TCP的20端口
- 用于传输数据
- 客户在一个会话上向服务器传输多个请求
- 每个文件请求都会建立一个数据连接
- 数据传输结束后，释放数据连接

### FTP进程模型（续）

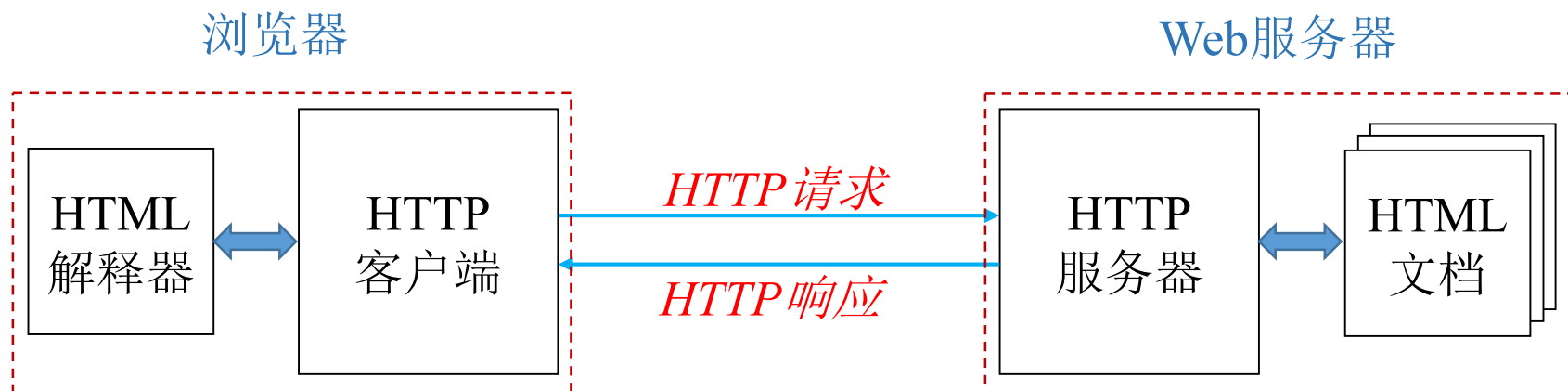


- Master server process: **daemon** at port **21**
- Client control process: random port
- Server control process: port **21**
- Server data transfer process: port **20**
- Client data transfer process: random port

### Web客户/服务器模型



### Web客户/服务器模型（续）



#### ■ 服务器

- ▶ Web 页面（HTML 文档）：包含到多种对象的链接
- ▶ 对象：可以是 HTML 文档、图像文件、视频文件、声音文件、脚本文件等
- ▶ 对象用 URL（统一资源定位符）编址：**协议类型://主机名//路径和文件名**

#### ■ 客户端

- ▶ 发出请求、接收响应、解释 HTML 文档并显示；
- ▶ 有些对象需要浏览器安装插件

# HTTP协议

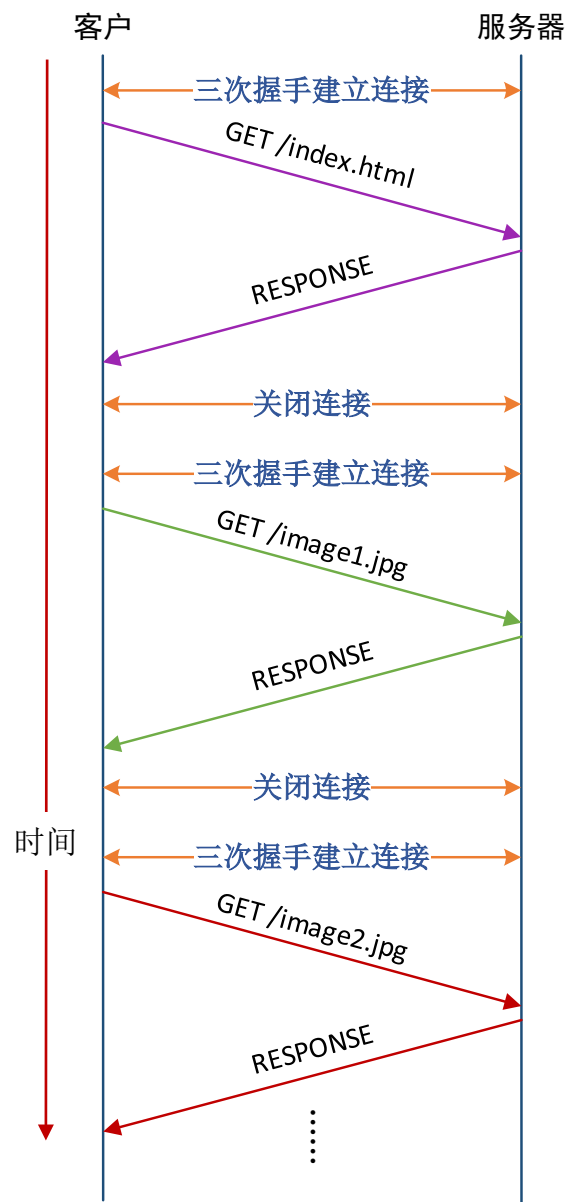
## ■ HTTP (HyperText Transfer Protocol)

- ▶ 传输层通常使用TCP协议，缺省使用TCP的80端口
- ▶ HTTP为无状态协议，服务器端不保留之前请求的状态信息
  - ✓ 无状态协议：效率低、但简单
  - ✓ 有状态协议：维护状态相对复杂，需要维护历史信息，在客户端或服务器出现故障时，需要保持状态的一致性等
- ▶ HTTP标准
  - ✓ HTTP/1.0: RFC 1945 (1996年)
  - ✓ HTTP/1.1: RFC 2616 (1999年)
  - ✓ HTTP/2: RFC 7540 (2015年)、RFC 8740 (2020年)
  - ✓ HTTP/3: RFC 9114 (2022年)

### HTTP/1.0示例

#### ■ 假设用户输入URL

- ▶ `http://www.nankai.edu.cn/computer/index.html`
- ▶ 页面包含10幅jpg图像



### 非持久连接和持久连接

思考：持久连接中的TCP连接何时关闭？

#### 非持久连接

##### ■ HTTP/1.0缺省为非持久连接

- ▶ 服务器接收请求、给出响应、关闭TCP连接

##### ■ 获取每个对象需要两阶段

- ▶ 建立TCP连接
- ▶ 对象请求和传输

##### ■ 每次连接需要经历TCP慢启动阶段

#### 持久连接

##### ■ HTTP/1.1缺省为持久连接

- ▶ 在相同的TCP连接上，服务器接收请求、给出响应；再接收请求、给出响应；响应后保持连接

##### ■ HTTP/1.1支持流水线机制

- ▶ 需要按序响应

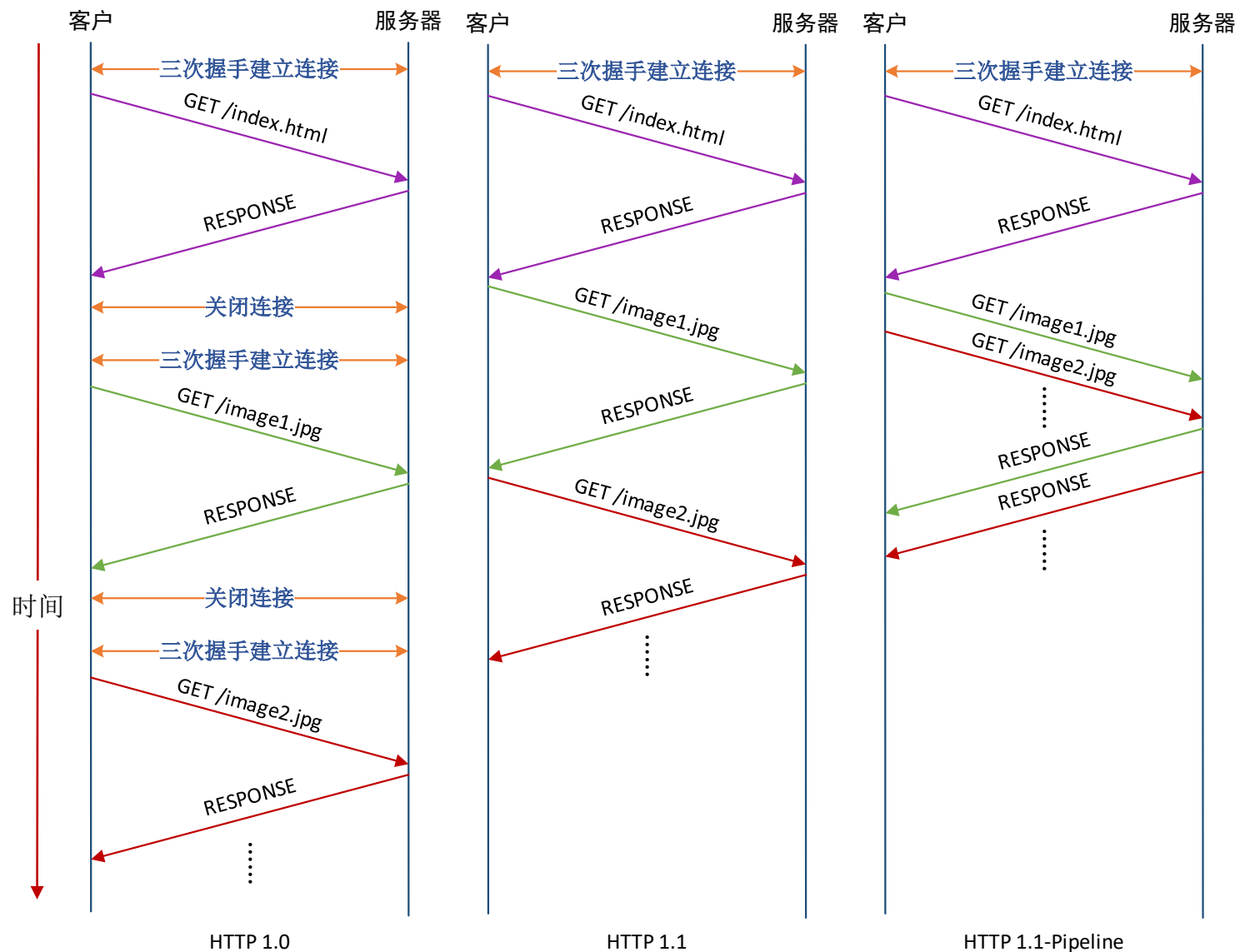
##### ■ 经历较少的慢启动过程，减少往返时间

- ▶ 降低响应时间



# 2.6 Web服务与HTTP协议

## HTTP/1.0与HTTP/1.1比较



- HTTP两种报文： 请求(request)、 响应(response)
- HTTP请求报文： 采用ASCII， 数据部分采用MIME格式

请求体  $\longrightarrow$  **【POST和PUT方法包含请求体】**

HTTP/1.1请求方法: GET, POST, HEAD, PUT, DELETE

### HTTP报文类型

#### ■ HTTP响应报文：数据部分采用MIME格式

响应行(状态码和解释)

HTTP/1.1 200 OK\r\n

Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n

Server: Apache/2.0.52 (CentOS)\r\n

Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n

ETag: "17dc6-a5c-bf716880"\r\n

Accept-Ranges: bytes\r\n

Content-Length: 2652\r\n

Keep-Alive: timeout=10, max=100\r\n

Connection: Keep-Alive\r\n

Content-Type: text/html; charset=ISO-8859-1\r\n

\r\n

data data data data data ...

回车字符  
换行字符

响应头

响应体，例如请  
求的HTML文档

### HTTP报文类型

#### ■ 典型的状态码

##### 200 OK

- 请求成功，被请求的对象包含在该响应的数据部分

##### 301 Moved Permanently

- 请求的对象被移走，新的位置在响应中通过**Location:** 给出

##### 400 Bad Request

- 服务器不能解释请求报文

##### 404 Not Found

- 服务器中找不到请求的文档

##### 505 HTTP Version Not Supported

- 服务器不支持相应的HTTP版本

Intel(R) 82567LM Gigabit Network Connection [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]

FileEditViewGoCaptureAnalyzeStatisticsTelephonyToolsInternalsHelp

Filter: http Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
813	43.946687	192.168.1.101	66.103.80.47	HTTP	181	GET /cgi-bin/alive?0001088 HTTP/1.1
816	43.996668	66.103.80.47	192.168.1.101	HTTP	60	HTTP/1.1 200 OK (text/plain)
826	44.457577	192.168.1.101	204.9.163.166	HTTP	333	POST /api/v1.0/pnr?language=EN&plugin=F
828	44.507171	204.9.163.166	192.168.1.101	HTTP	271	HTTP/1.1 200 OK
835	45.629833	192.168.1.101	128.119.245.12	HTTP	489	GET /wireshark-labs/INTRO-wireshark-fi
837	45.646802	128.119.245.12	192.168.1.101	HTTP	434	HTTP/1.1 200 OK (text/html)
838	45.670226	192.168.1.101	128.119.245.12	HTTP	429	GET /favicon.ico HTTP/1.1
839	45.687572	128.119.245.12	192.168.1.101	HTTP	564	HTTP/1.1 404 Not Found (text/html)
840	45.724273	192.168.1.101	128.119.245.12	HTTP	459	GET /favicon.ico HTTP/1.1
841	45.739188	128.119.245.12	192.168.1.101	HTTP	564	HTTP/1.1 404 Not Found (text/html)
847	48.670194	192.168.1.101	128.119.245.12	HTTP	459	GET /favicon.ico HTTP/1.1
848	48.689680	128.119.245.12	192.168.1.101	HTTP	564	HTTP/1.1 404 Not Found (text/html)

Frame 835: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)

Ethernet II, Src: HonHaiPr\_Od:ca:8f (00:22:68:0d:ca:8f), Dst: Cisco-Li\_45:1f:1b (00:22:6b:45:1f:1b)

Internet Protocol Version 4, Src: 192.168.1.101 (192.168.1.101), Dst: 128.119.245.12 (128.119.245.12)

Transmission Control Protocol, Src Port: 57522 (57522), Dst Port: http (80), Seq: 1, Ack: 1, Len: 435

Hypertext Transfer Protocol

GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\nHost: gaia.cs.umass.edu\r\nUser-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.22) Gecko/20110902 Firefox/3.6.22 (.NET CLR 3.5.30729)\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n\r\n

0000 00 22 6b 45 1f 1b 00 22 68 0d ca 8f 08 00 45 00 . "kE... " h.....E.

0010 01 db 29 13 40 00 80 06 00 00 c0 a8 01 65 80 77 ..).@... ..e.w

0020 f5 0c e0 b2 00 50 ca 16 89 b3 d9 41 b1 83 50 18 ....P.. ...A..P.

0030 40 29 39 5f 00 00 47 45 54 20 2f 77 69 72 65 73 @)9...GE T/wires

0040 68 61 72 6b 2d 6c 61 62 73 2f 49 4e 54 52 4f 2d hark-lab s/INTRO-

0050 77 60 72 65 72 68 61 72 6b 2d 66 60 65 65 21 2e wireshark file1

Frame (frame), 489 bytes

Packets: 850 Displayed: 132 Marked: 0 Dropped: 0

Profile: Default

2022/10/11

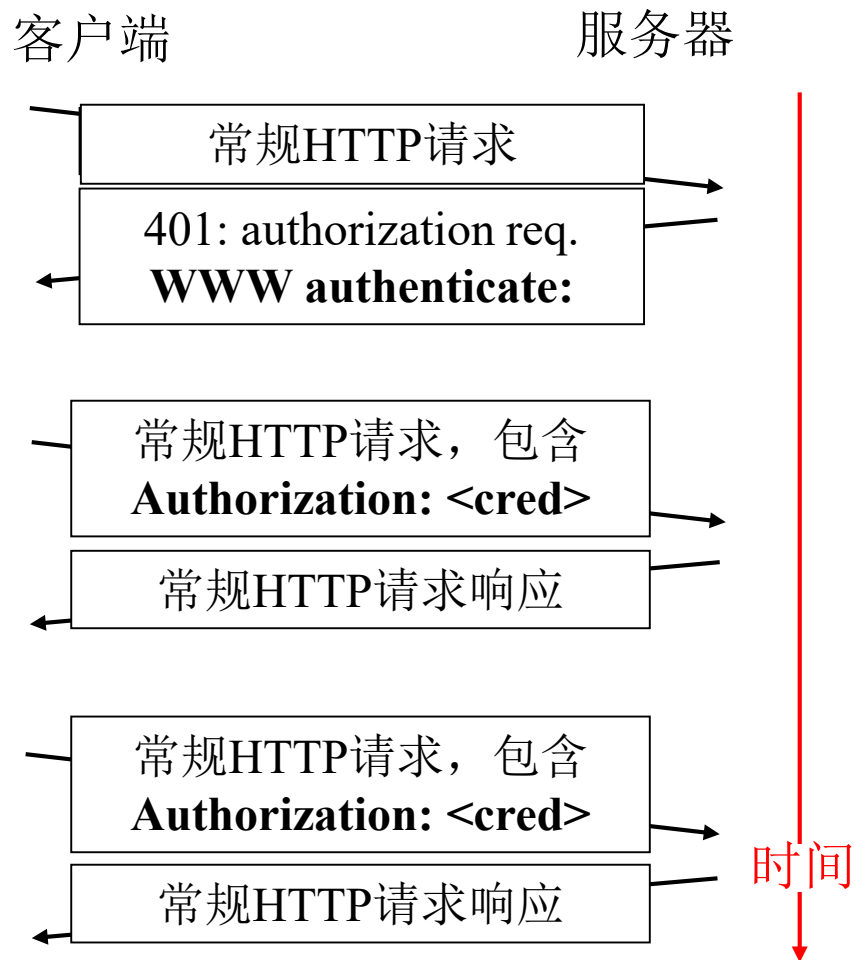
计算机网络与信息安全研究室

21

### 用户-服务器交互：认证

■ 认证：控制对服务器内容的访问

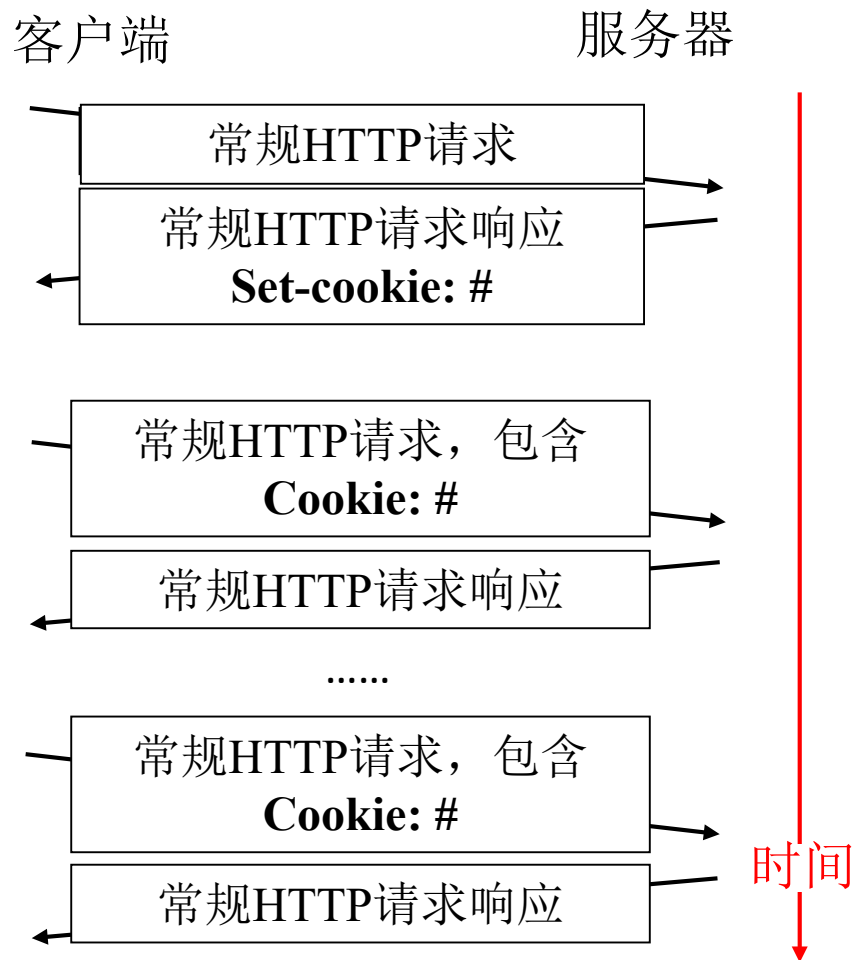
- ▶ 认证方法：通常使用“名字-口令”
- ▶ 无状态：客户端需要在每个请求中携带认证信息
- ▶ 每个请求头中包含 **authorization:**
- ▶ 如果请求头中无 **authorization:**，则服务器拒绝访问，并在响应头中包含 **WWW authenticate:**





### 用户-服务器状态: Cookies

- 服务器使用cookies保持状态
  - ▶ HTTP响应头中使用**set-cookie:**
    - 选择的cookie号具有唯一性
  - ▶ 后继的HTTP请求中使用**cookie:**
  - ▶ **Cookie文件**保存在用户的主机中, 由用户主机中的浏览器管理
  - ▶ Web服务器建立后端数据库, 记录用户信息
  - ▶ 例如:
    - **Set-Cookie:** SID=31d4d96e407aad42;  
Path=/; Domain=example.com
    - **Cookie:** SID=31d4d96e407aad42



### Web缓存机制：客户端缓存

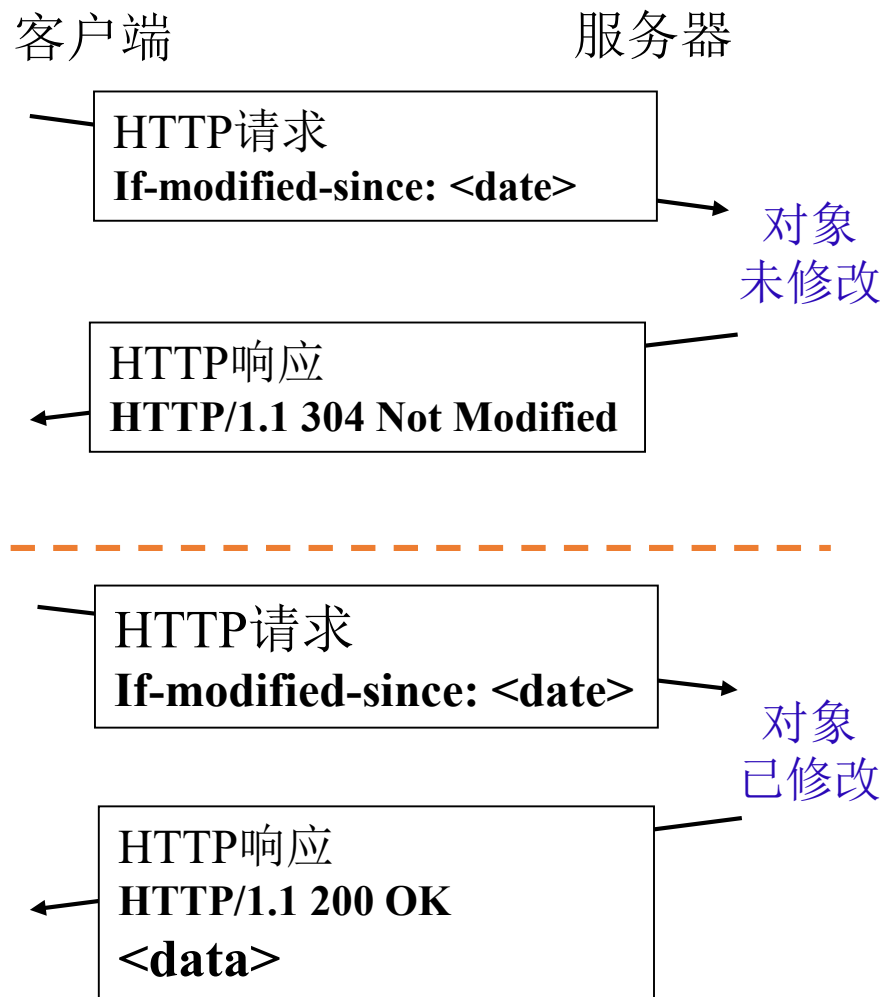
■ **目标**：如果被请求的对象在客户端缓存有最近版本，则不需要发送该对象

■ **客户端**：在发送的HTTP请求中指定缓存的时间，请求头包含

**If-modified-since: <date>**

■ **服务器**：如果缓存的对象是最新的，在响应时无需包含该对象，响应头包含

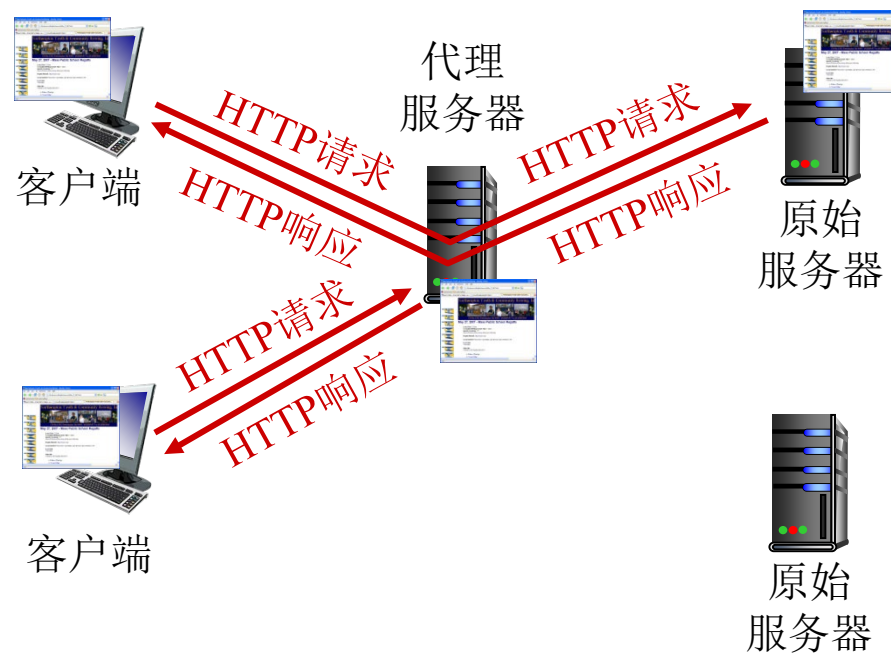
**HTTP/1.1 304 Not Modified**





### Web缓存机制：代理服务器缓存

- **目标：**由代理服务器进行缓存，尽量减少原始服务器参与
- 用户设置浏览器：通过代理服务器进行Web访问
- 浏览器将所有的HTTP请求发送到代理服务器
  - ▶ 如果缓存中有被请求的对象，则直接返回对象
  - ▶ 否则，代理服务器向原始服务器请求对象，再将对象返回给客户端



降低时延、减少网络流量

### HTTP发展现状

#### ■ HTTP/1.0 (1996)

- ▶ 无状态，非持久连接

#### ■ HTTP/1.1 (1999)

- ▶ 支持长连接和流水线机制
- ▶ 缓存策略优化、部分资源请求及断点续传

#### ■ HTTPS: HTTP+TLS (2008)

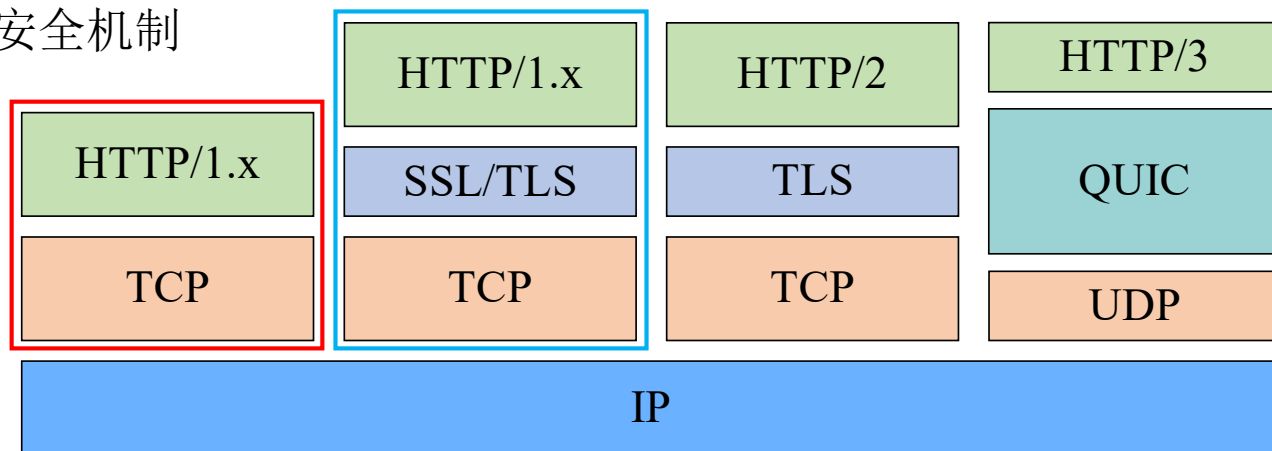
- ▶ 增加SSL/TLS (TLS 1.2) 层，在TCP之上提供安全机制

#### ■ HTTP/2.0 (2015、2020)

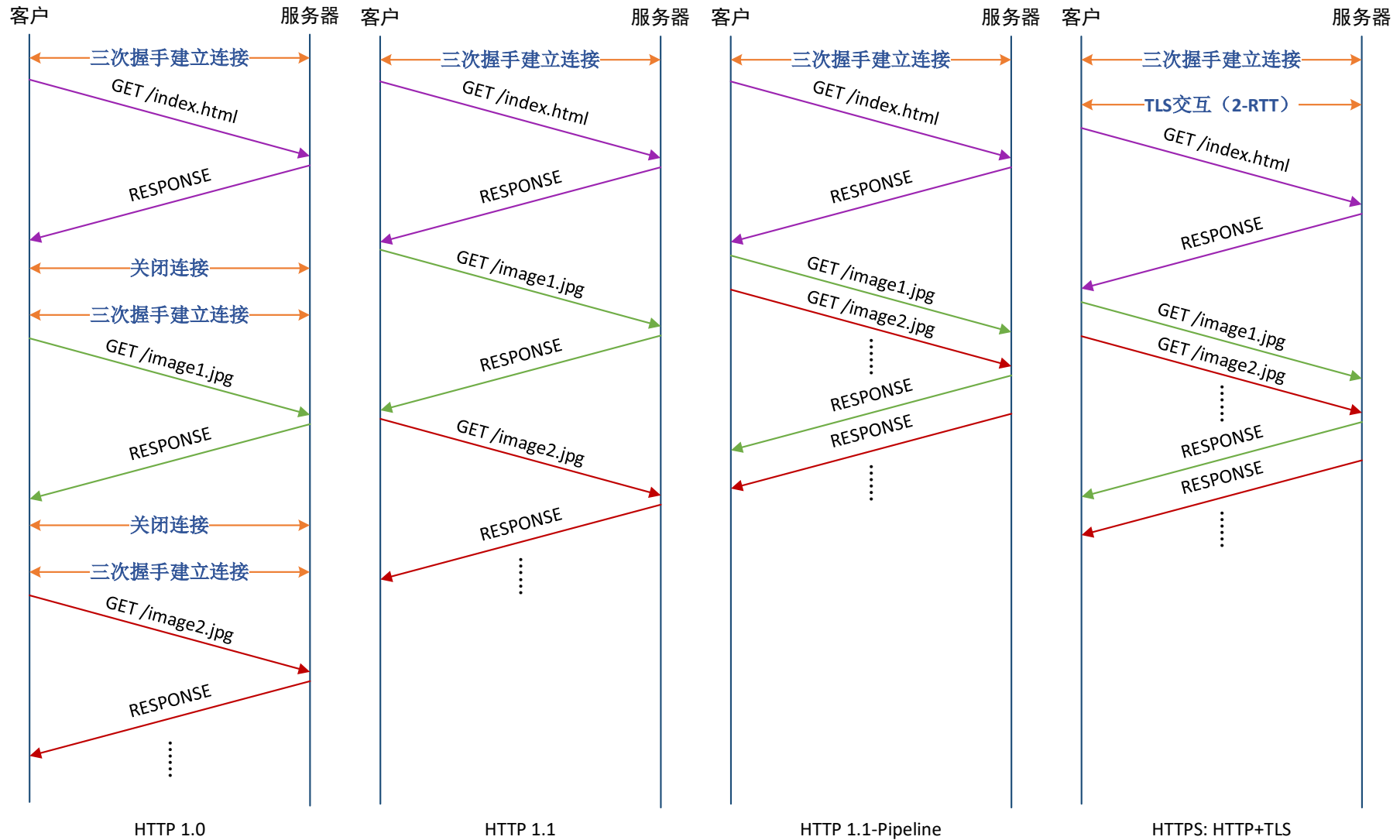
- ▶ 目标：提高带宽利用率、降低延迟
- ▶ 增加二进制格式、TCP多路复用、头压缩、服务端推送等功能

#### ■ HTTP/3.0 (2022)

- ▶ 运行在QUIC+UDP之上



## 2.6 Web服务与HTTP协议



例：页面index.html包含10幅图像

# HTTP 1.1存在的问题

## ■ HTTP 1.1的问题

### ▶ 队头阻塞问题

- 基于文本协议的问答有序模式，先请求的必须先响应

















### ▶ 传输效率问题

- 文本格式、冗长重复的头部等

## ■ HTTP 1.1队头阻塞的解决策略

### ▶ 浏览器建立多个TCP连接

- 一般最多可以建立6个TCP连接
- 通过不同TCP连接传送的请求没有响应顺序的要求
- 耗费较多的计算和存储资源

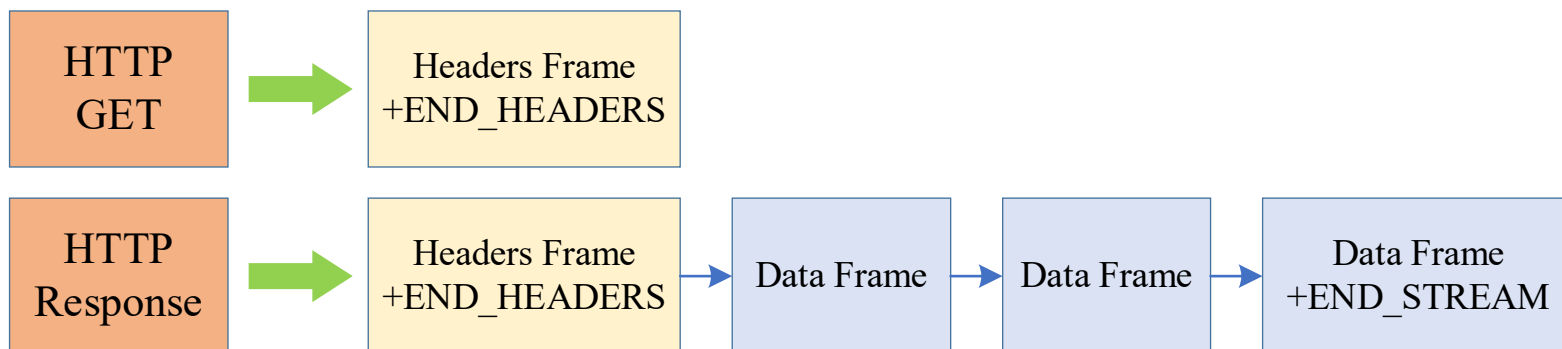
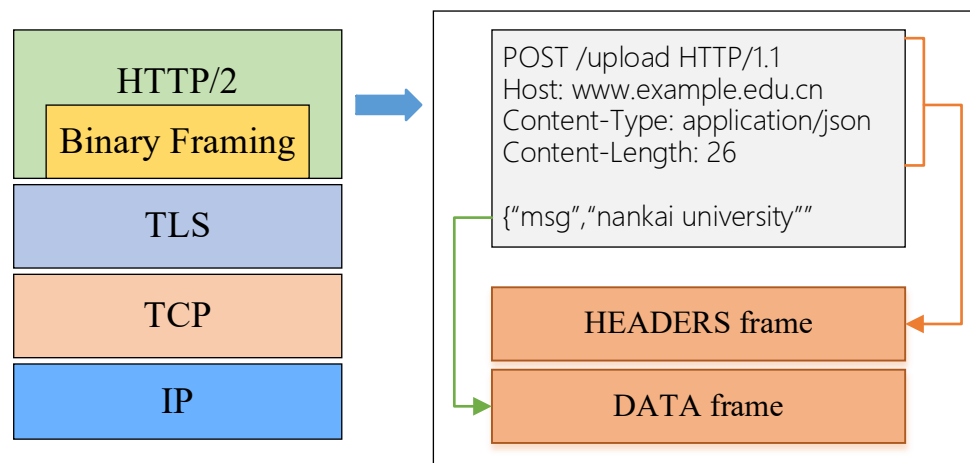
Name	Status	Type	Initiator	Size	Time	Waterfall
tile-4.png	200	png	h2_demo_frame...	1.3 KB	223 ms	
tile-5.png	200	png	h2_demo_frame...	1.3 KB	223 ms	
aksb.min...	200	script	h2_demo_frame...	5.0 KB	153 ms	
tile-13.p...	200	png	h2_demo_frame...	1.3 KB	155 ms	
tile-63.p...	200	png	h2_demo_frame...	3.7 KB	157 ms	
tile-33.p...	200	png	h2_demo_frame...	1.3 KB	168 ms	
tile-40.p...	200	png	h2_demo_frame...	1.3 KB	163 ms	
tile-31.p...	200	png	h2_demo_frame...	1.3 KB	166 ms	
tile-74.p...	200	png	h2_demo_frame...	3.1 KB	168 ms	
tile-44.p...	200	png	h2_demo_frame...	2.0 KB	248 ms	
tile-47.p...	200	png	h2_demo_frame...	1.3 KB	260 ms	
tile-41.p...	200	png	h2_demo_frame...	1.3 KB	271 ms	
tile-68.p...	200	png	h2_demo_frame...	2.3 KB	266 ms	
tile-43.p...	200	png	h2_demo_frame...	2.4 KB	272 ms	
tile-6.png	200	png	h2_demo_frame...	1.3 KB	283 ms	
tile-72.p...	200	png	h2_demo_frame...	1.3 KB	353 ms	

### HTTP 2.0协议基本原理

#### ■ 二进制分帧传输

- ▶ 不改变HTTP原有的语义
- ▶ 将HTTP请求和响应分割成帧，采用二进制编码
- ▶ 帧为最小传输单位

#### ■ 最常用的 HTTP 请求 / 响应的帧形式



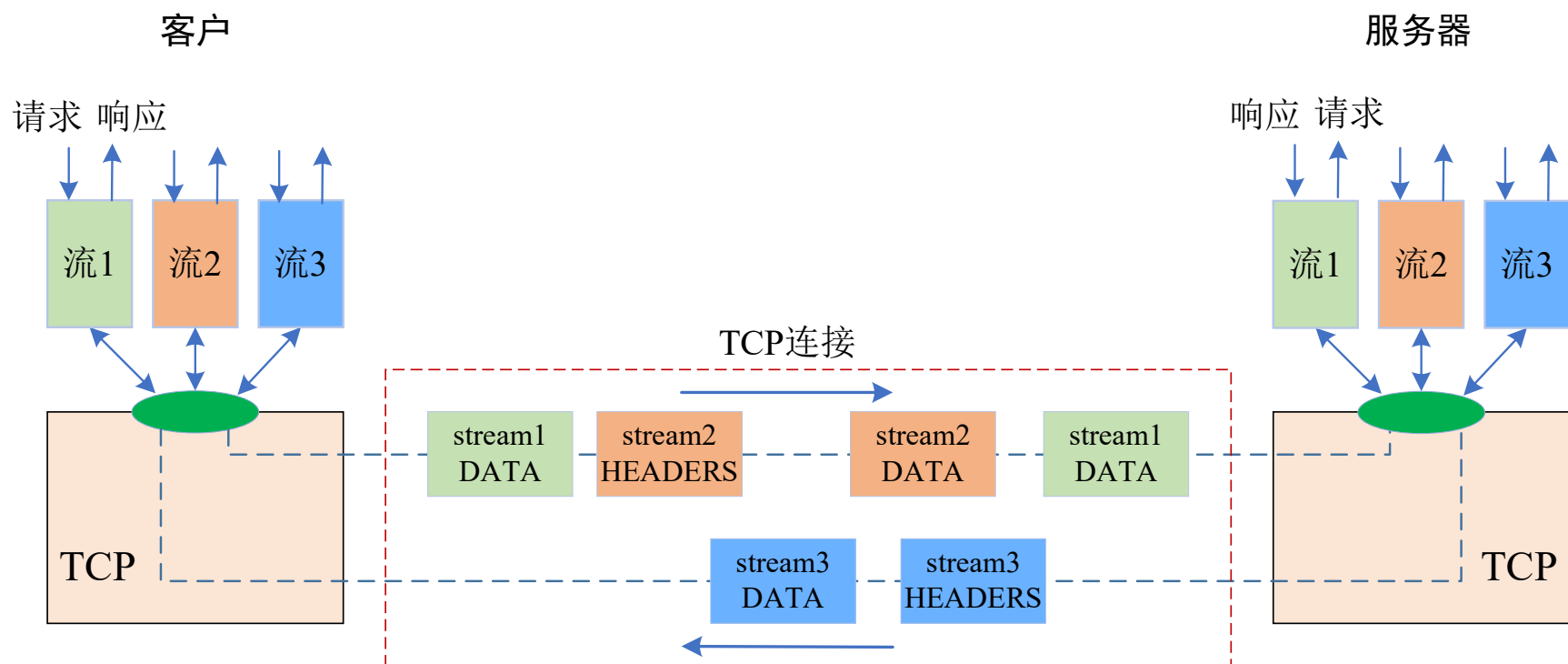
### HTTP 2.0协议基本原理（续）

■ TCP连接复用：提高连接利用率，解决HTTP的队头阻塞问题

- ▶ 消息(Message)：HTTP一次请求或响应，包含一个或多个帧
- ▶ 流(Stream)：简单看成一次请求和应答，包含多个帧
- ▶ 每个TCP连接中可以承载多个流，不同流的帧可以交替穿插传输
- ▶ 流的创建与标识
  - Stream ID：标识一个流。客户端创建的流，ID为奇数；服务器创建的流，ID为偶数；0x00和0x01用于特定场景；Stream ID 不能重复使用，如果一条连接上ID分配完，会新建一条连接。接收端通过Stream ID进行消息的组装。
  - 流创建：发送和接收到HEADERS帧（包含新Stream ID）时创建
  - 流优先级：可以依据重要性为流设置不同的优先级（1~256），在HEADERS帧中承载

### HTTP 2.0协议基本原理（续）

#### ■ TCP连接复用（示例）



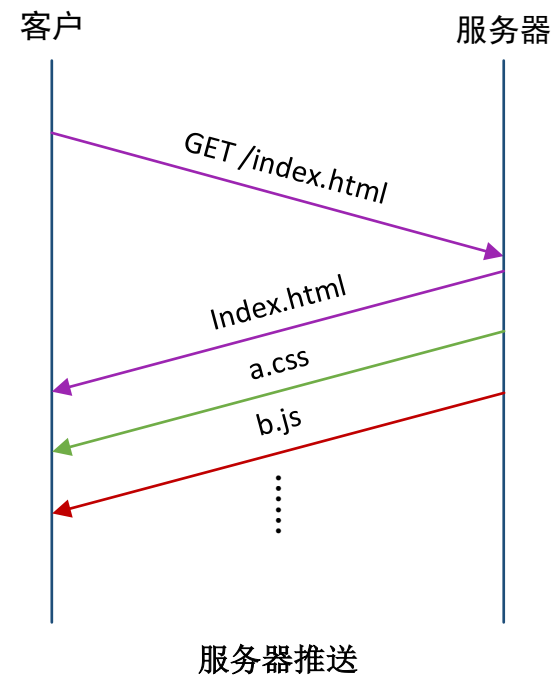
### HTTP 2.0协议基本原理（续）

#### ■ 服务器推送：提高响应速度

- ▶ 服务器在请求之前先推送响应信息到客户端，推送的响应信息可以在客户端被缓存

#### ■ HTTP头压缩（HPACK）

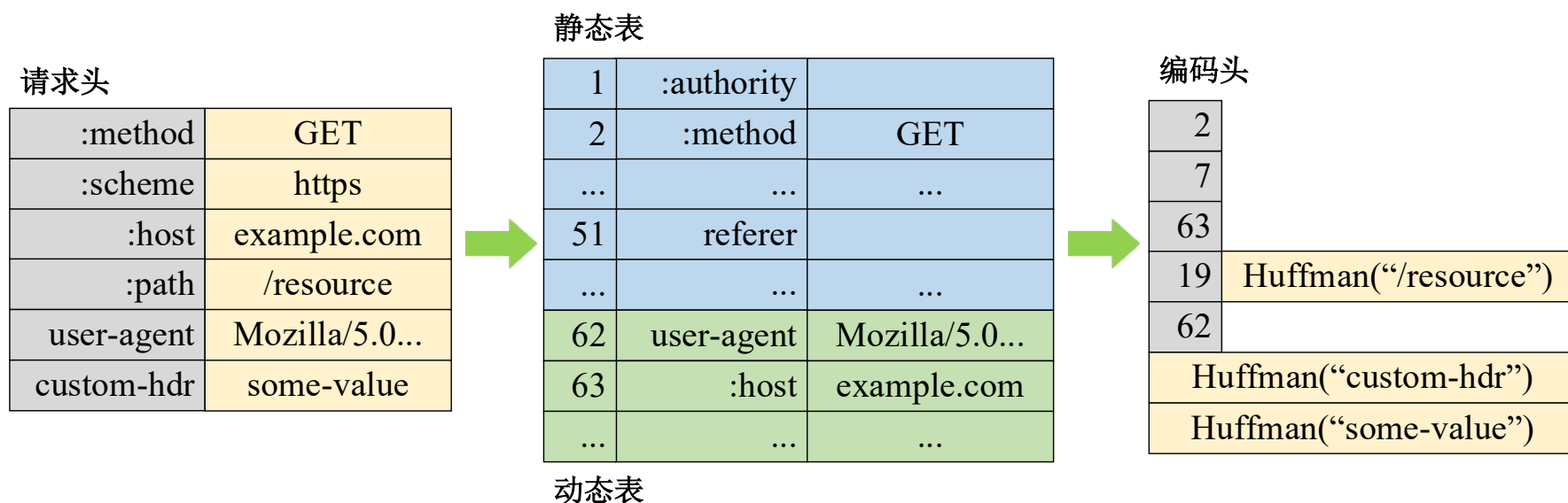
- ▶ 请求头由大量的键值组成，多个请求的键值重复程度很高
- ▶ 静态表：定义通用HTTP头域，常用键值无需重复传送，直接引用内部字典的整数索引
- ▶ 动态表：两边交互发现新的头域，添加到动态表
- ▶ 自定义键值：采用Huffman编码





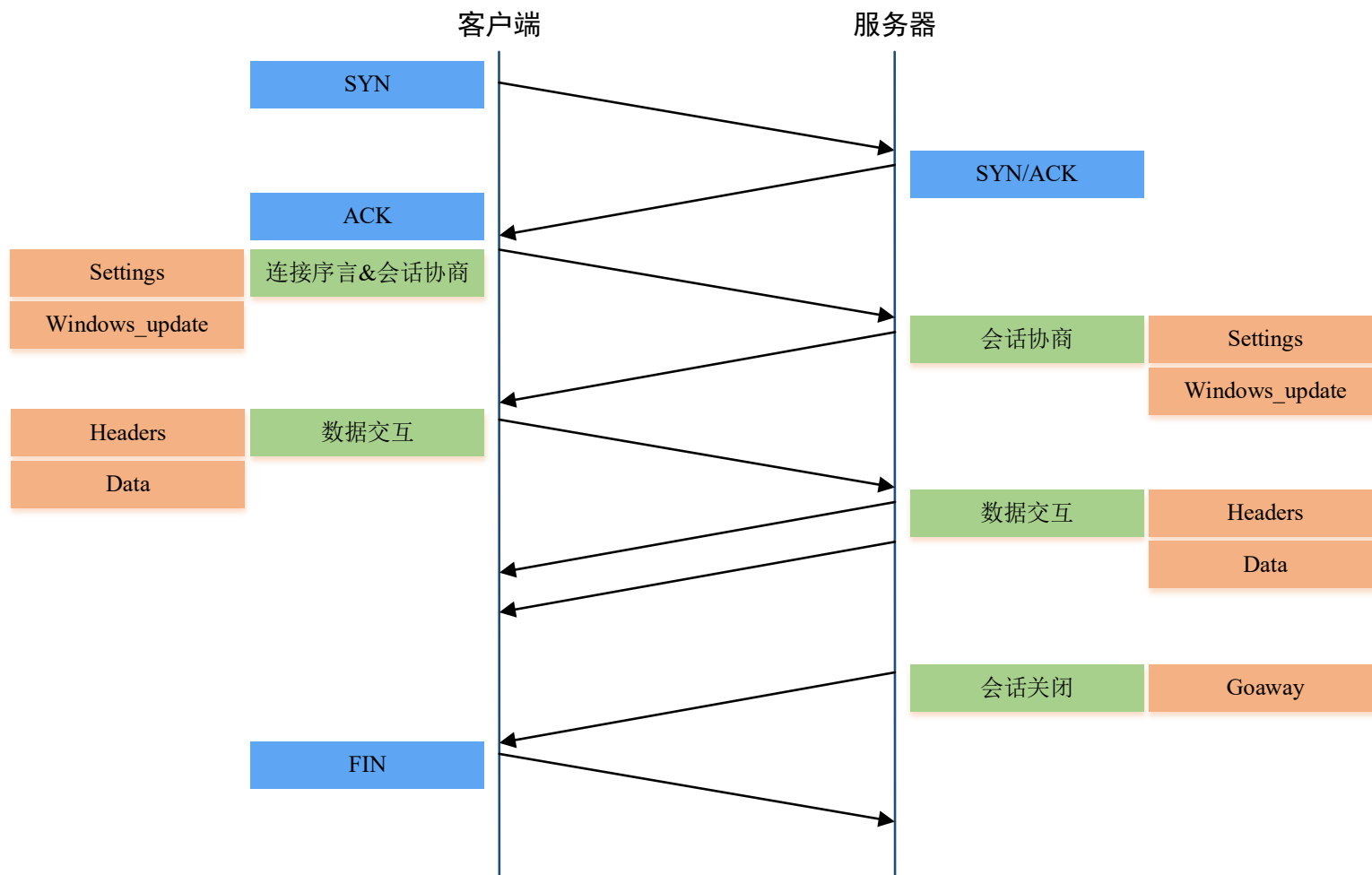
### HTTP 2.0协议基本原理（续）

#### ■ HTTP头压缩示例



### HTTP 2.0协议基本原理（续）

#### ■ HTTP连接流程



# HTTP 2.0协议基本原理（续）



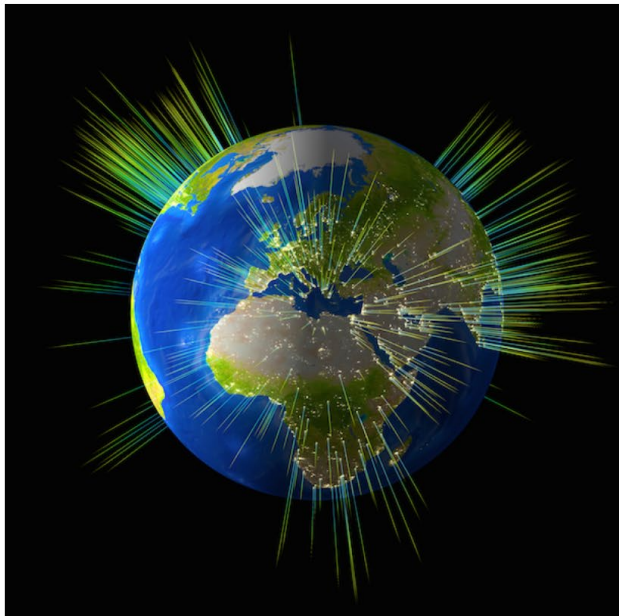
HTTP/2 is the future of the Web, and it is here!

Your browser supports HTTP/2!

This is a demo of HTTP/2's impact on your download of many small tiles making up the Akamai Spinning Globe.

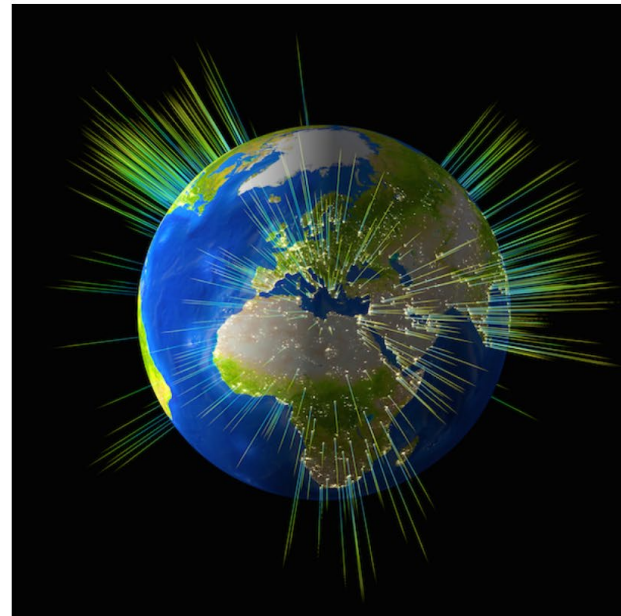
HTTP/1.1

Latency: 138ms  
Load time: 9.68s



HTTP/2

Latency: 131ms  
Load time: 1.63s



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

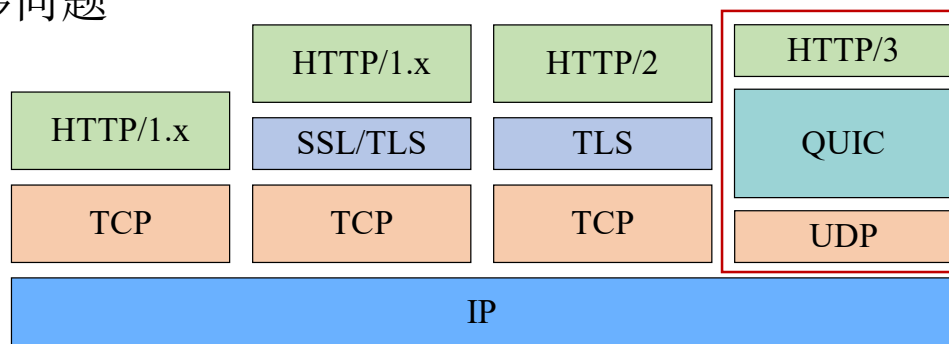
### HTTP 2.0协议基本原理（续）

#### ■ HTTP 2.0协议解决的问题

- ▶ 通过引入流机制，解决了HTTP队头阻塞问题，提高了传输效率
- ▶ 通过二进制编码、头压缩机制提高了网络带宽利用率
- ▶ 通过服务器推送，加快了页面响应速度

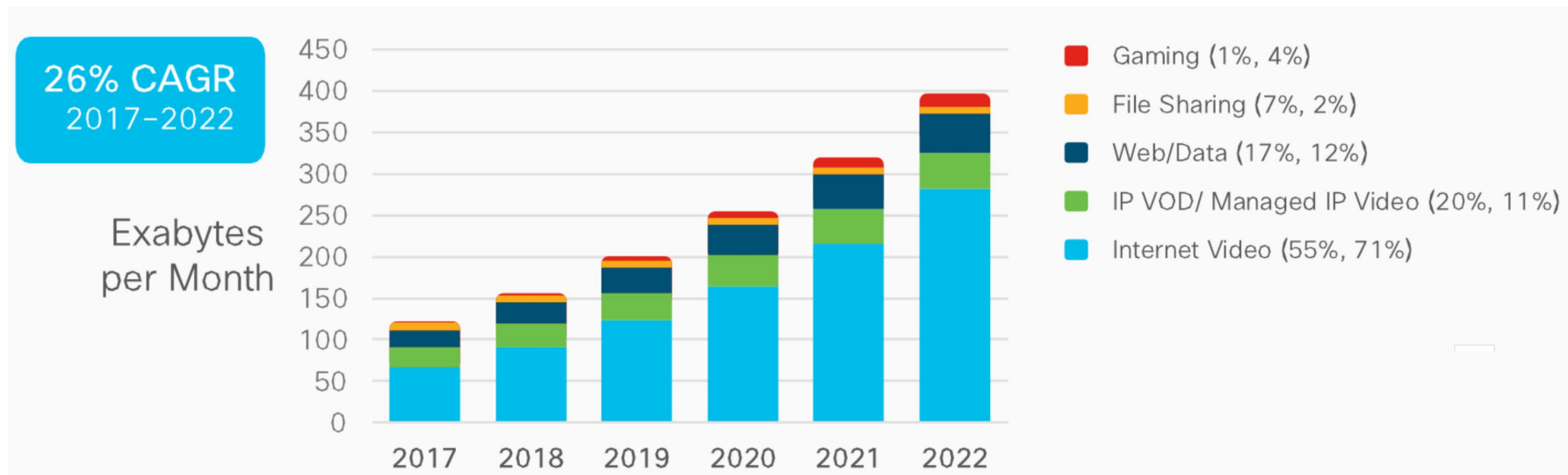
#### ■ HTTP 2.0协议没有解决的问题

- ▶ TCP+TLS的多次交互，造成启动延迟问题
- ▶ 移动主机和多宿主机的连接迁移问题
- ▶ TCP队头阻塞问题



### 互联网IP流量发展趋势

全球IP流量（按应用分类）



■ IP Video 流量到2022年将占82%

参考：Cisco Visual Networking Index: Forecast and Trends, 2017-2022 (white paper), Cisco, Feb. 2019

关注文章Waiting Times In Quality Of Experience For Web Based Services关于QoE的观点

### 内容分发网络概述

CDN (Content Distribution Network)

- 基本思想源于MIT对Web服务瞬间拥塞问题的解决（1998）
  - ▶ 一种Web缓存系统，靠近网络边缘（用户）提供内容服务
  - ▶ 目前提供更丰富的服务，包括静态内容、流媒体、用户上传视频等
- 主要优点
  - ▶ 降低响应时延，避免网络拥塞
  - ▶ 避免原始服务器过载及防止DDoS攻击
  - ▶ 分布式架构，具有良好的可扩展性
  - ▶ 对用户透明，无需用户感知

### 内容分发网络概述（续）

#### ■ 关键问题

- ▶ CDN服务器如何布局
- ▶ 在哪里缓存、缓存哪些内容
- ▶ 如何进行重定向，将请求调度到较近或负载较轻的CDN服务器

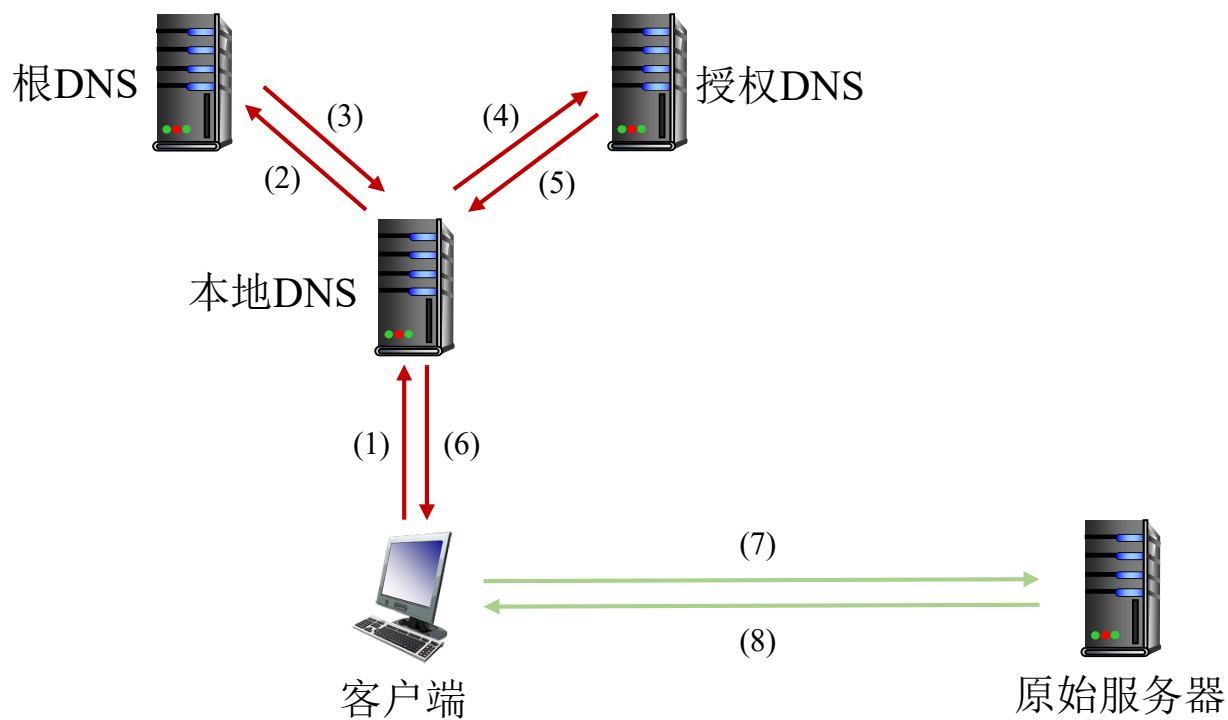
#### ■ CDN服务提供

- ▶ CDN服务提供商：Akamai、蓝讯、世纪互联、网宿等
- ▶ 互联网内容提供商（ICP）：腾讯、百度等
- ▶ 互联网服务提供商（ISP）：移动、联通、电信等

- ✓ 腾讯云CDN：2100+节点覆盖国内移动、联通、电信及十几家中小型运营商，以及全球50+国家地区，全网带宽120Tbps+
- ✓ Akamai: 240,000部署在 > 120 个国家内 (2015)

### CDN基本原理

#### ■ 传统Web服务访问



URL: `www.example.com`



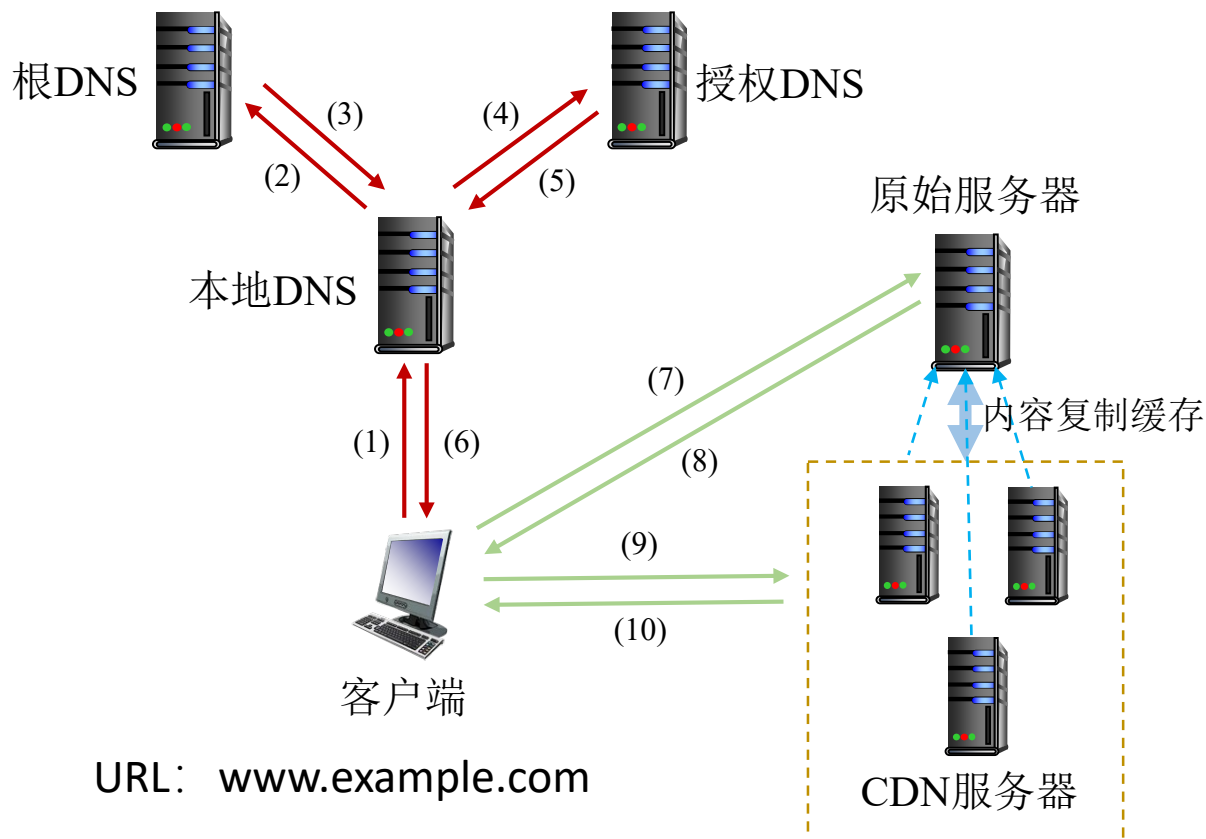
# 2.7 内容分发网络

## CDN基本原理

### CDN的实现机制（1）

#### HTTP重定向

- ✓ 原始服务器决策CDN服务器
- ✓ HTTP响应：状态码30X, **Location**:指明新的位置



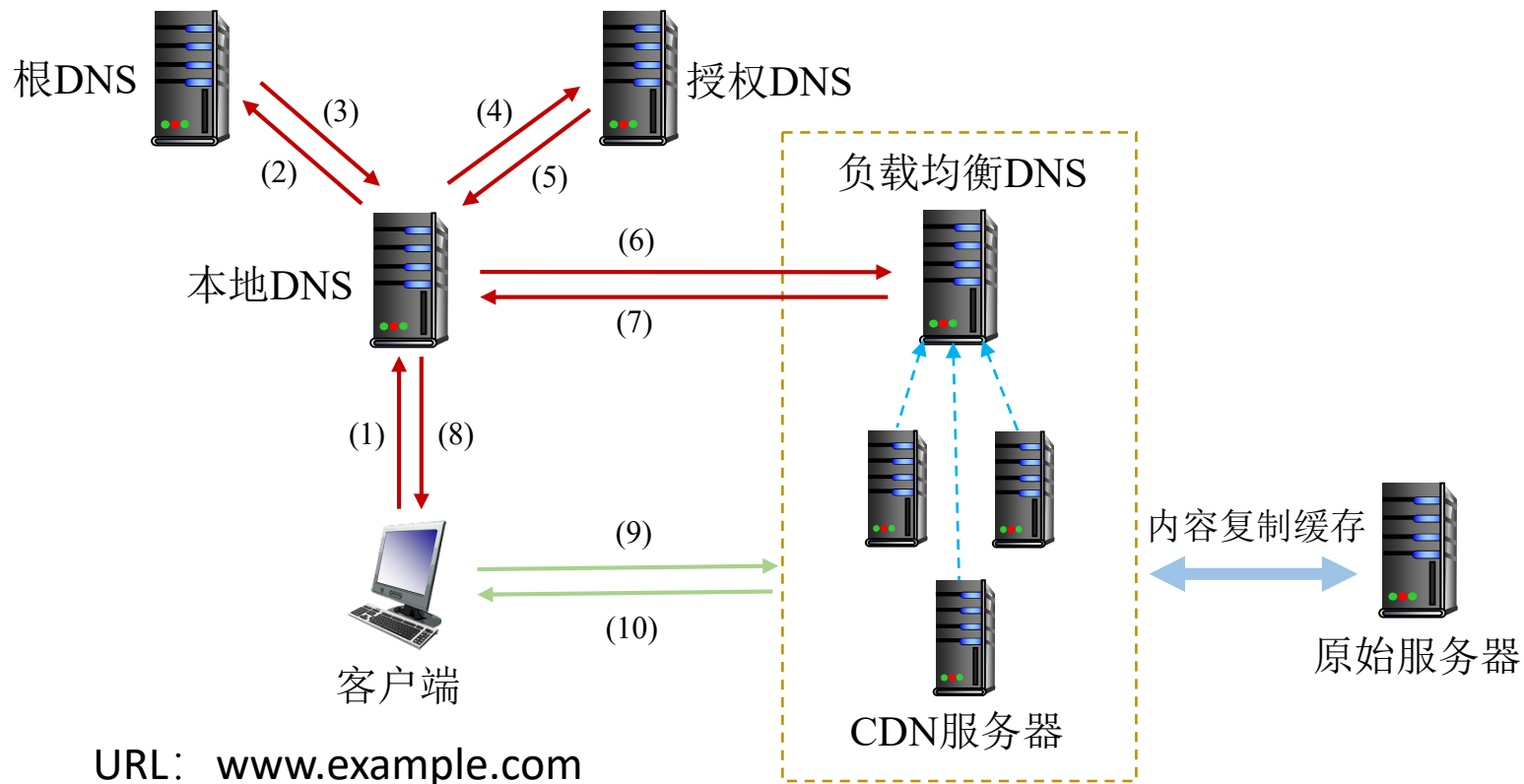
# 2.7 内容分发网络

## CDN基本原理

### CDN的实现机制（2）

#### DNS辅助

- ✓ 负载均衡DNS负责决策CDN服务器选择
- ✓ 负载均衡DNS需要收集CDN服务器的位置和负载情况
- ✓ 如果找不到被请求的对象，需要从原始服务器获取



扩展：CDN服务器的层次化组织→**移动边缘缓存**→**移动边缘计算**

### DASH概述

#### ■ DASH (Dynamic Adaptive Streaming **over HTTP**)

- ▶ MPEG组织制定的标准（标准号ISO/IEC 23009-1）
  - 也称MPEG-DASH
- ▶ 基于HTTP的流媒体传输协议
  - 基于HTTP渐进式下载（**类流媒体**）
- ▶ 类似协议：
  - HTTP Live Streaming（HLS），苹果公司
  - HTTP Dynamic Streaming（HDS），Adobe公司
  - Microsoft Smooth Streaming（MSS），微软公司

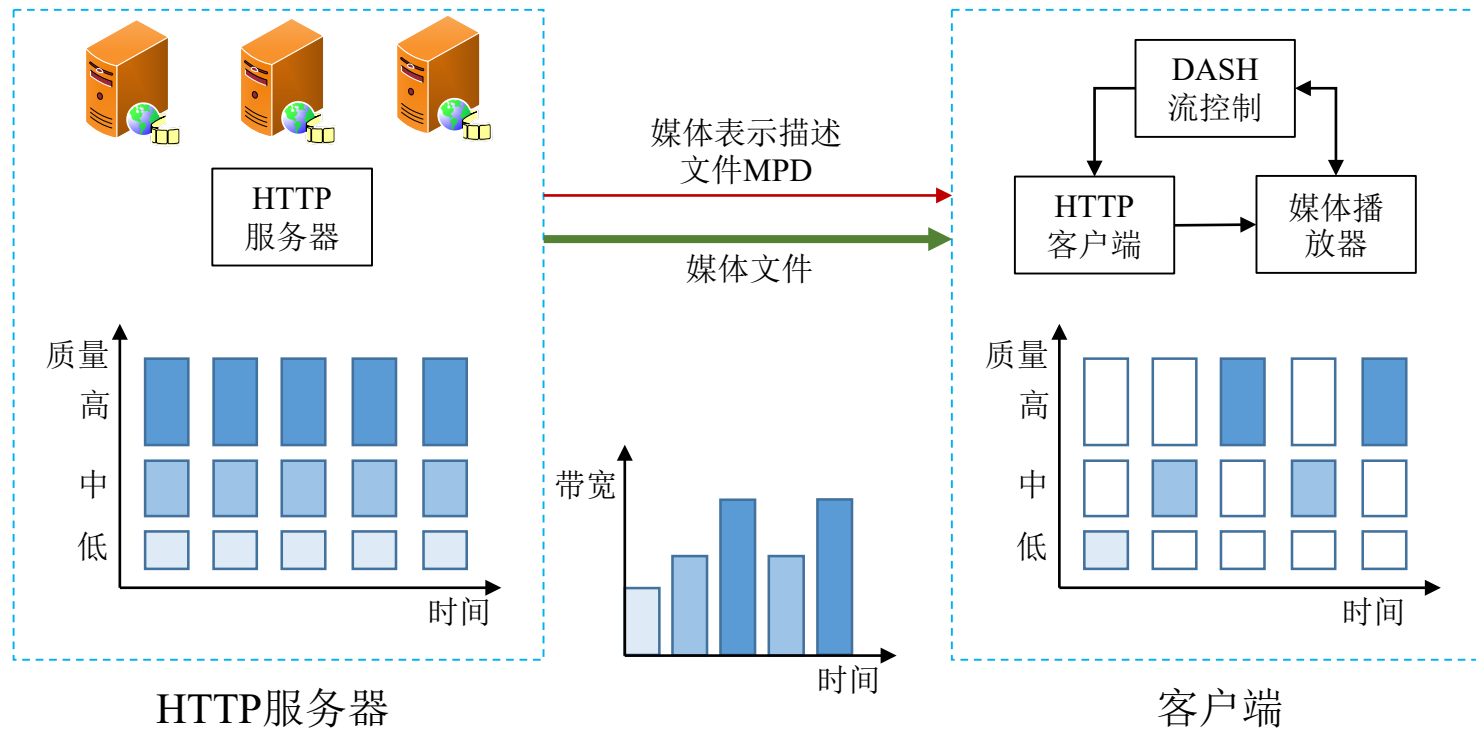
### DASH概述

#### ■ 基本思想:

- ▶ 完整视频被拆分为固定时长 (2s-10s)、不同码率的视频片段(segment)
- ▶ 视频片段与媒体表示描述 (Media Presentation Description, MPD) 文件一同存放于DASH服务器
- ▶ 客户端根据自身设备性能、当前网络条件、客户端缓冲大小等自适应选择一种视频码率进行下载

# 2.8 动态自适应流媒体协议

## DASH基本原理

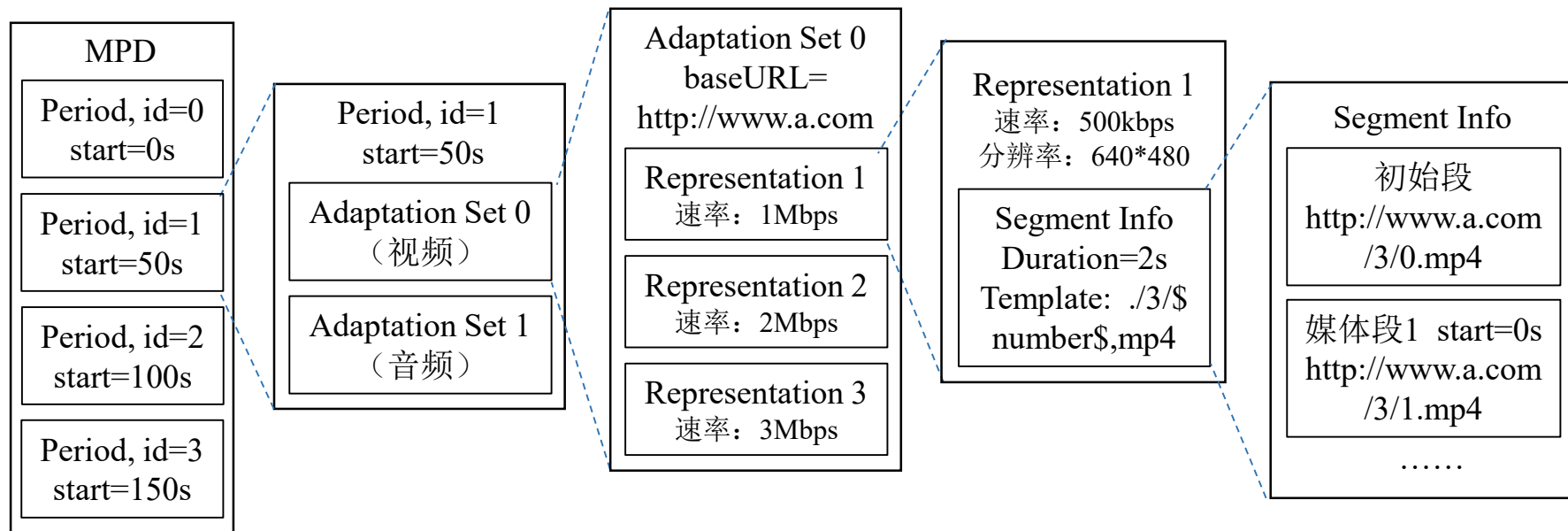


- 例如：HTTP服务器中保存有高中低三种质量的视频片段，DASH客户端评估网络状况，通常在保证**视频流畅**的前提下，获取最高质量的视频片段

### DASH基本原理（续）

#### ■ MPD (Media Presentation Description) 文件

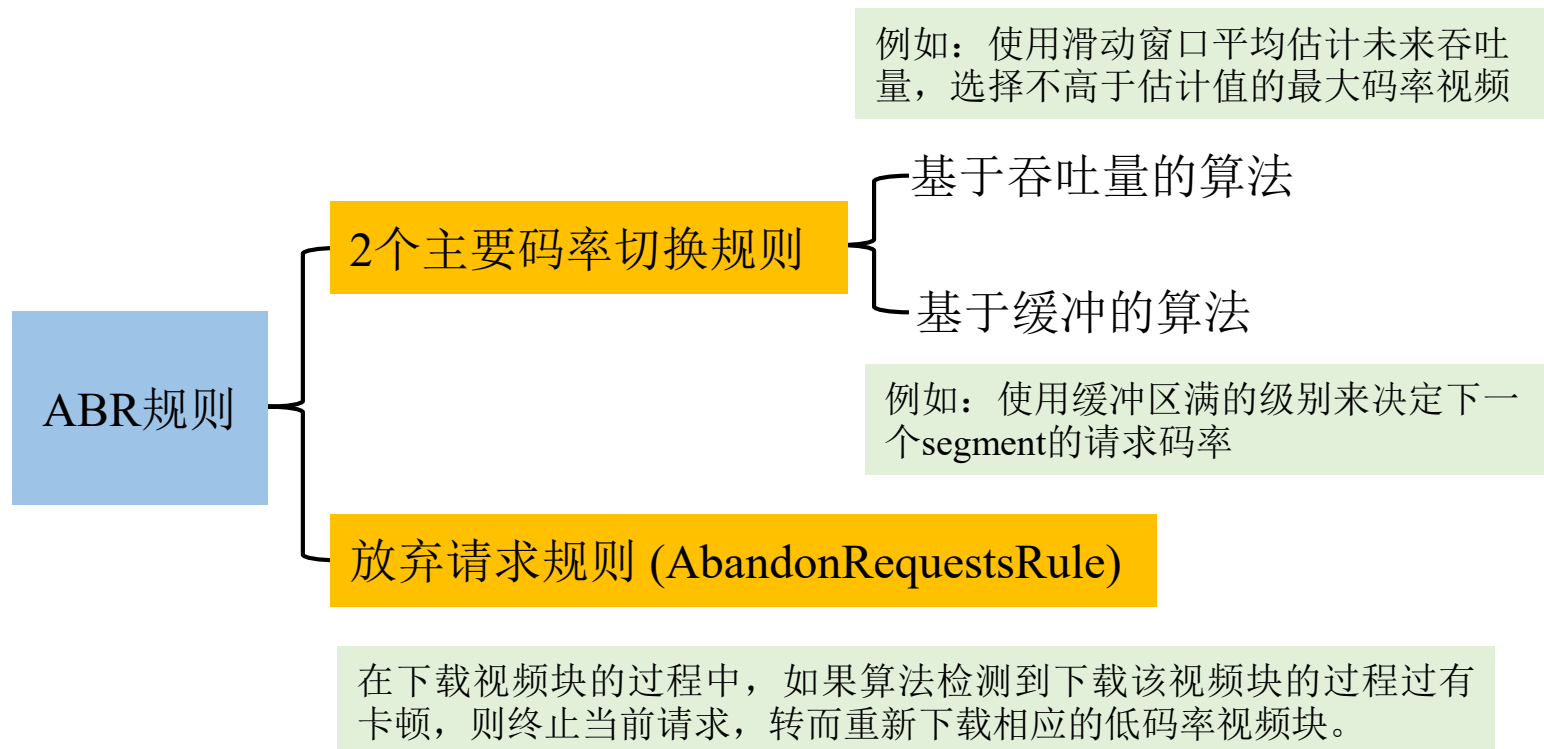
- ▶ 一种 XML 文件，描述了DASH流媒体中视频/音频文件信息



### DASH基本原理

典型的DASH开源播放器dash.js

#### ■ 自适应码率（Adaptive bitrate, ABR）规则



扩展：考虑多DASH服务器同时传输

- 应用协议与进程通信模型
- 传输层服务对应用的支持
- Socket编程
- 域名系统DNS
- 电子邮件服务与协议
- 文件传输服务与协议
- Web 服务与HTTP协议
- 内容分发网络CDN
- 动态自适应流媒体协议DASH