Lecture 1

网络协议的概念

- A **network protocol** defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other events.

- 网络协议定义了在两个和多个通信实体之间交换信息的格式和顺序，以及在消息或其他事件的传输和/或接收上这些通信实体所采取的动作。

总结

- 课程管理
- 网络协议定义了在两个和多个通信实体之间交换信息的格式和顺序，以及在消息或其他事件的传输和/或接收上这些通信实体所采取的动作。
- 过去的互联网:
  - 事实:
    - 互联网始于 1960 年代后期的 ARPANET。
    - 初始链路带宽只有 50 kbps。
    - 1969年底主机数只有4台。
  - 过去的互联网带来的影响:
    - ARPANET由ARPA赞助 → 好的设计从失败中诞生（失败是成功之母）
    - 初始的IMPs设计非常简单 → 保证网络的简单有效
    - 许多网络的结合 → 需要使用网络来连接其他网络
- 目前的互联网:
  - 连接到互联网的主机数量约为 10 亿台。
  - 当前互联网的骨干网带宽约为40/100 Gbps。
  - 互联网是分层的，各个ISP通过PoP和IXP互连。
  - 需要处理规模性、复杂性、去中心化、安全性等问题。

Lecture 2

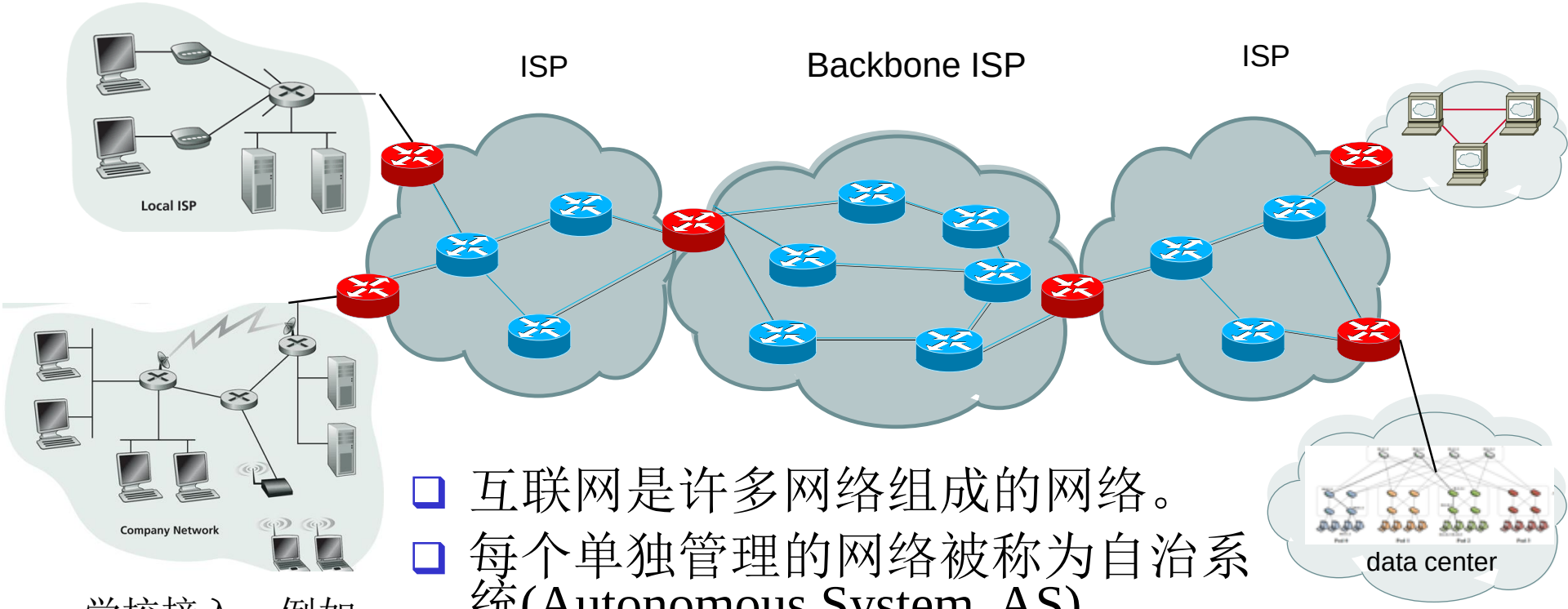
回顾

- 网络协议定义了在两个和多个通信实体之间交换信息的格式和顺序，以及在消息或其他事件的传输和/或接收上这些通信实体所采取的动作。
- 关键的互联网里程碑事件及其影响:
  - ARPANET 由ARPA赞助 → 好的设计从失败中诞生
  - 最初的IMPs是由一家小公司制造的→ 保证网络的简单有效
  - 许多网络的结合 → 网络互连:需要使用网络来连接其他网络
  - 商业化→ 支持分布式自治系统的架构

互联网物理基础设施

用户接入

- 电缆, 光纤, DSL, 无线



学校接入，例如

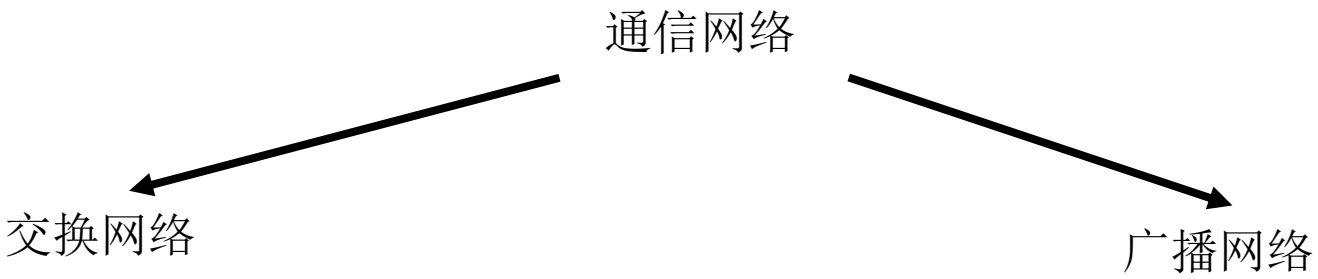
- 以太网
- 无线

- 互联网是许多网络组成的网络。
- 每个单独管理的网络被称为自治系统(Autonomous System, AS)。

通用复杂性

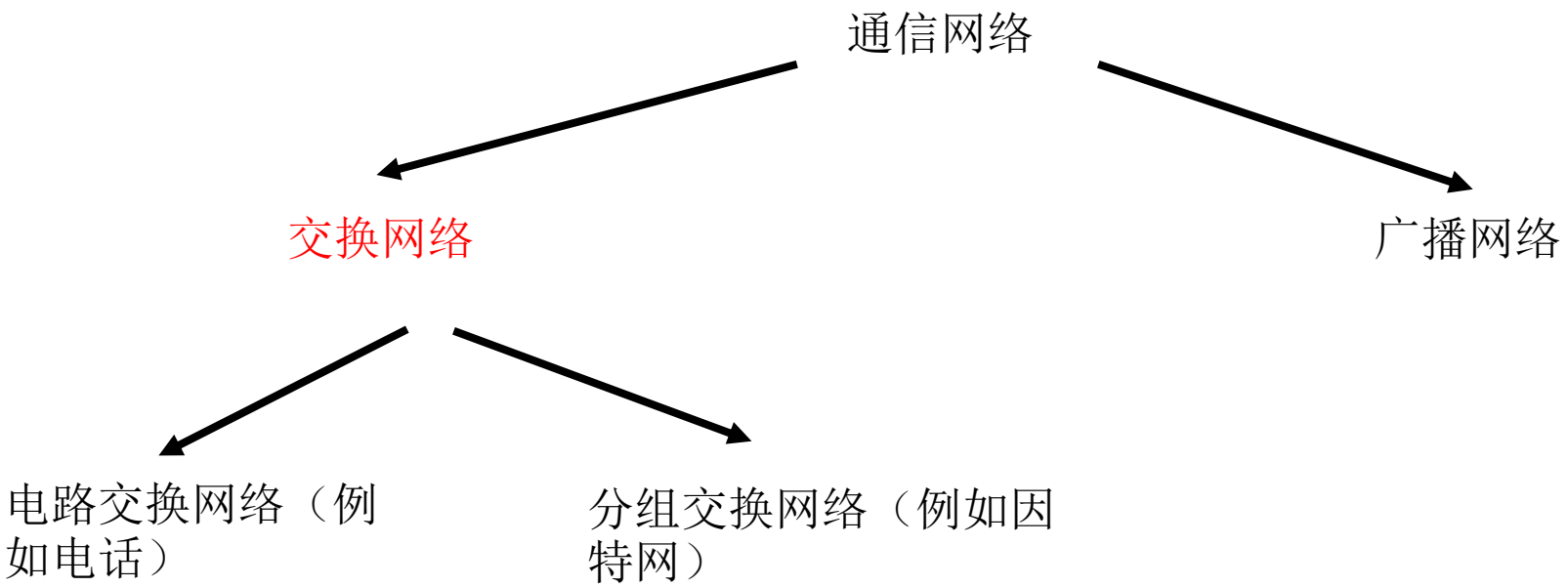
- 在高度组织的系统中，复杂性主要来自于设计策略，这些策略旨在为其环境和组件中的不确定性提供鲁棒性。
  - 可扩展性是对整个系统规模和复杂性变化的鲁棒性。
  - 进化性是指对各种（通常是长期）时间尺度上的巨大变化的鲁棒性。
  - 可靠性是指对组件故障的鲁棒性。
  - 效率是对资源稀缺的鲁棒性。
  - 模块化是对组件重新布置的鲁棒性。

通信网络的分类



- 广播网络
  - 节点共享一个公共通道; 一个节点发送的信息可以被网络中其他所有的节点接收
  - 例子: TV, radio（无线电，收音机）
- 交换网络
  - 信息可以被节点中一个小的子集（通常只有一个）所接收

交换网络分类



- 电路交换：每个呼叫/对话的专用网络：
  - 例如电话和蜂窝语音
- 分组交换：数据通过网络以离散的“块”的形式发送：
  - 例如因特网和蜂窝移动数据

Lecture 3

电路交换：过程

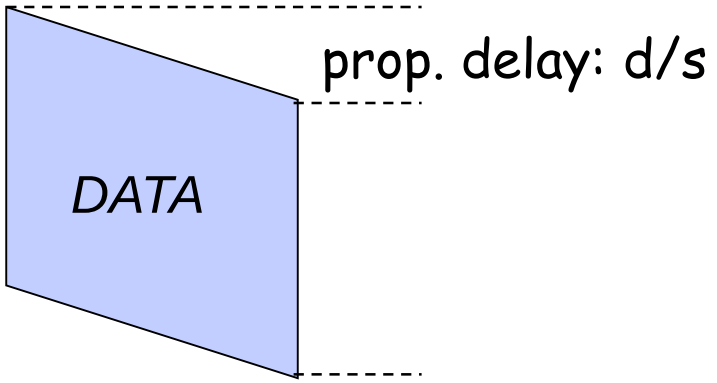
- 三个阶段
  - 电路建立
  - 数据传输
  - 电路终止

电路交换网络中的延迟计算

- 传播延迟（ Propagation delay ）：第一个比特从源头到目的地的延迟

Propagation delay:

- d = length of physical link
- s = propagation speed in medium (~2×10^5 km/sec)

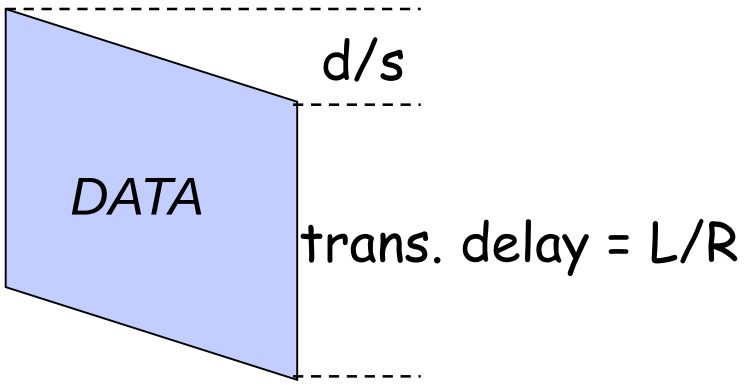


电路交换网络中的延迟计算

- 传输延迟（Transmission delay）:以线路速率将数据传入链路的时间

Transmission delay:

- R = reserved bandwidth (bps)
- L = message length (bits)



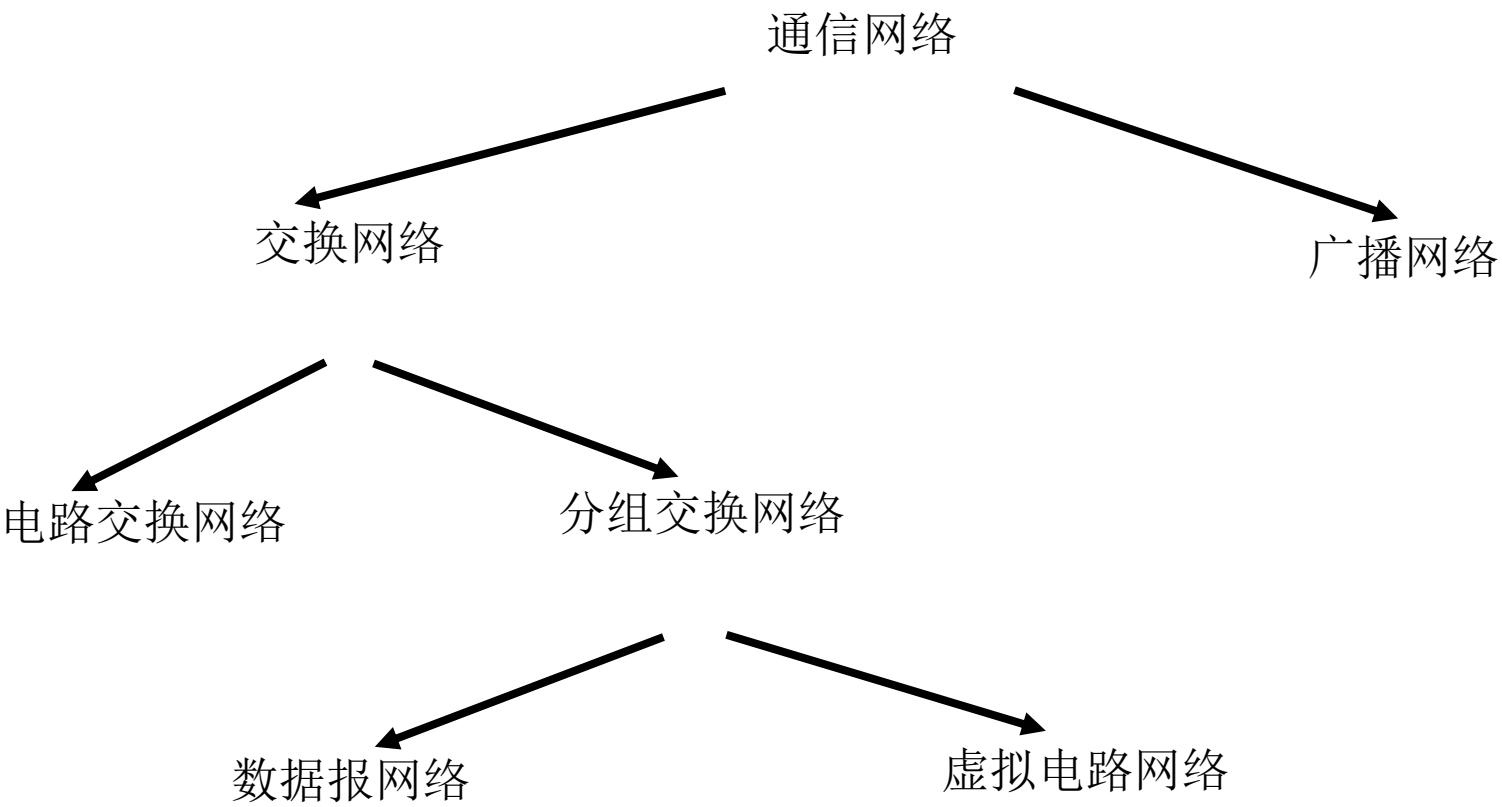
排队论

- 策略:
  - 建模系统状态
    - 如果我们知道系统在每个状态下花费的时间百分比，我们就可以得到许多基本问题的答案:一个新请求需要等待多长时间才能得到服务?
- 系统状态随事件变化:
  - 介绍状态转移图
  - 关注于均衡: 状态趋势既不增长也不收缩(关键问题：如何定义均衡)
- 我们的方法: 我们对极其精确的建模不感兴趣，但是想要定量的直觉

总结：
分组交换 vs. 电路交换

- 分组交换相对于电路交换的优势
  - 分组交换相对于电路交换的最重要的优势是统计多路复用，因此带宽使用效率更高
- 分组交换的缺点
  - 潜在的拥塞: 数据包延迟和高丢失
    - 需要保证可靠数据传输和拥塞控制的协议
    - 分组交换网络可以在保证服务质量（QoS）的同时仍然获得统计多路复用的优势，但这增加了很多复杂性
  - 包头的开销
  - 每个数据包的处理开销

通信网络分类总结



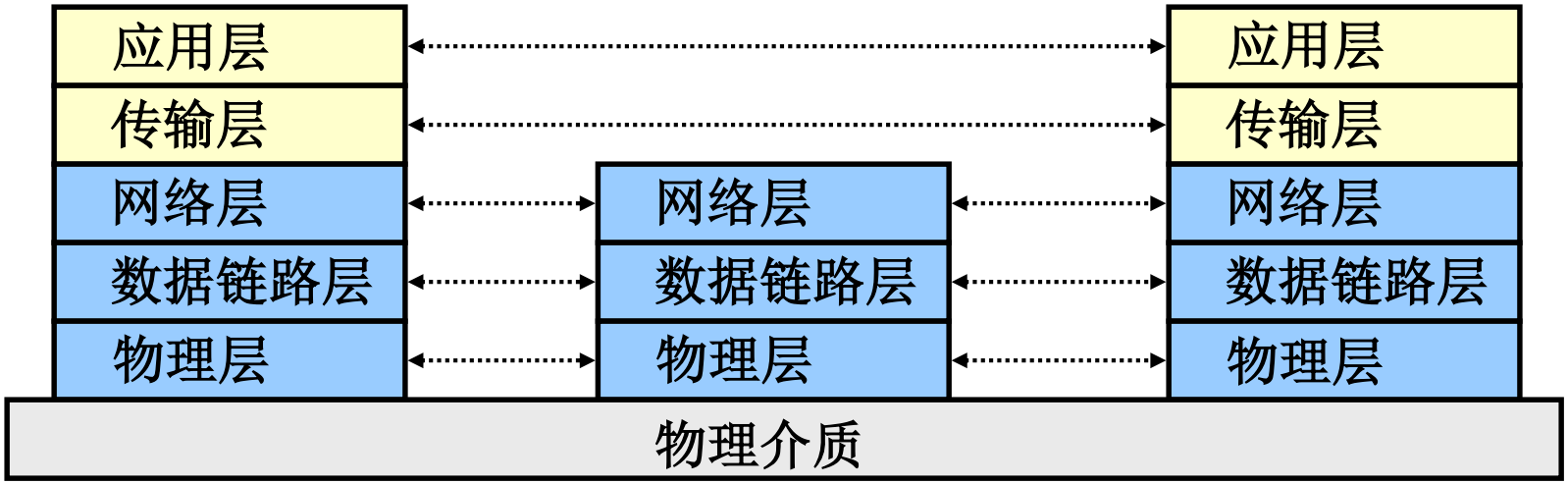
Lecture 4

回顾: 电路交换 vs. 分组交换

	电路交换	分组交换
资源使用	使用单个分区带宽	使用整个链路带宽
预留/设置	需要预留(设置延迟)	不需要预留
资源竞争	忙信号(会话丢失)	拥塞 (长延迟和包丢失)
收费	时间	数据包
包头	没有包头	每个包都有包头
快速链路处理	快速	每个包都需要处理

什么是分层？

- 一种将网络系统组织成一系列逻辑上不同的实体的技术，这样一个实体提供的服务完全基于前一个（较低层次）实体提供的服务 。



ISO/OSI 概念

- ISO – 国际标准组织
- OSI – 开放系统互连

- 服务 – 说明一个层的作用
- 接口 – 说明如何访问服务
- 协议 – 指定服务该如何实现
  - 一组管理两个或多个对等点之间通信的规则和格式

端到端原则意味着什么？

- 应用最了解需求，将功能放在尽可能高的层中
- 在较低层实现功能之前要三思而后行，即使您认为它对应用程序有用

总结: 端到端原则

- 如果上层能做到，就不要在下层做-- 越高的层越知道它想要什么
- 如果它在较低层添加功能

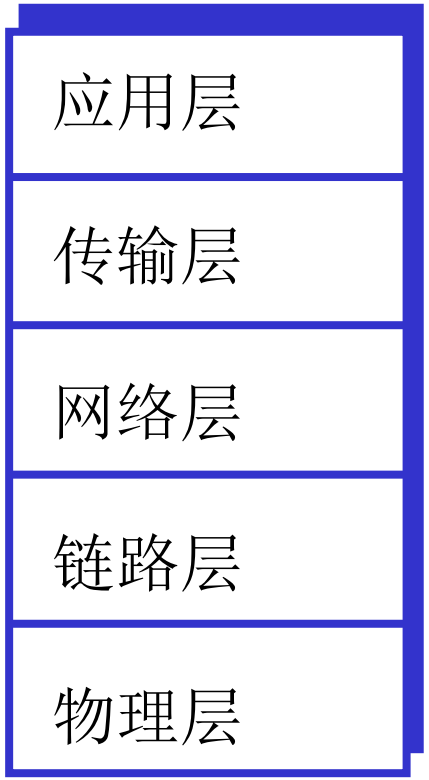
  （1）被大量应用（当前的或者未来的）使用来提高性能,

  （2）不会（过于）伤害到其他应用，且

  （3）不会增加（太多）复杂性/开销
- 实际的权衡, 例如,
  - 允许下层有多个接口（一个提供功能，一个不提供）

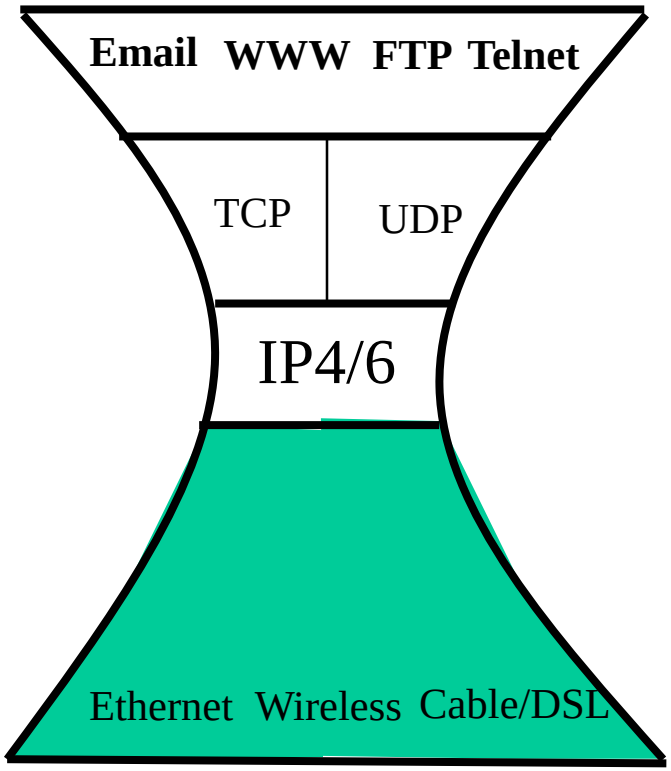
网络协议层

- 五层
  - 应用层: 应用
    - ftp, smtp, http, p2p, IP电话, 区块链, MapReduce, …
  - 传输层: 主机到主机的数据传输
    - tcp (可靠的), udp (不可靠的)
  - 网络层: 数据报从源头到目的地的路由
    - ipv4, ipv6
  - 链路层: 相邻网络元素之间的数据传输
    - 以太网, 802.11, 电缆, DSL, …
  - 物理层: 电线上的比特
    - 电缆, 无线, 光纤



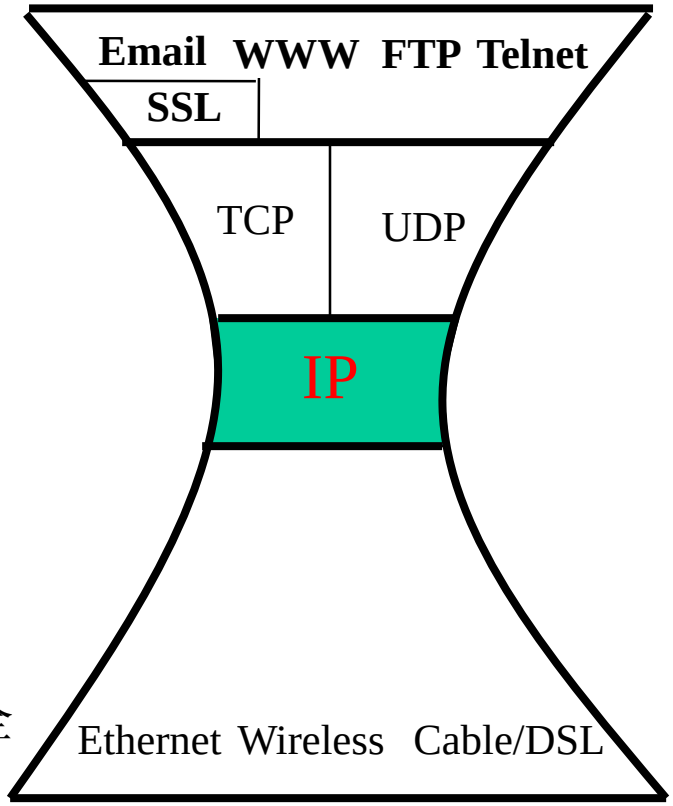
链路层 (以太网)

- 服务（对网络层）
  - 多路复用/多路解复用
    - 从/到网络层
  - 错误检测
  - 多重访问控制
    - 仲裁对共享介质的访问
- 接口
  - 将帧发送到可以直接访问的对等方



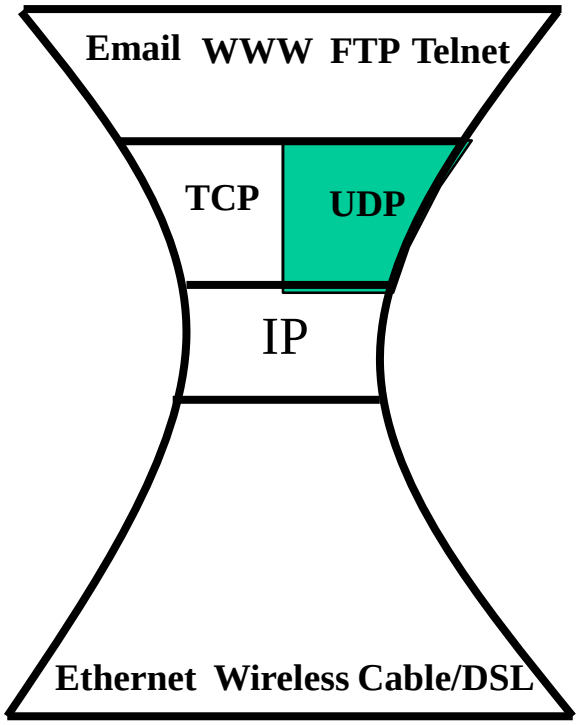
网络层: IP

- 服务 (对传输层)
  - 多路复用/多路解复用 从/到传输层
  - 碎片化和重组: 将一个段分成多个数据包
    - 在IPv6中删除了
  - 错误检测
  - 路由: 尽最大努力将数据包从源头发送到目的地
  - 特定的QoS/CoS
  - 不提供可靠性或预留
- 接口:
  - 使用特定的QoS/CoS将数据包发送给指定全局目的地（传输层）的对等方



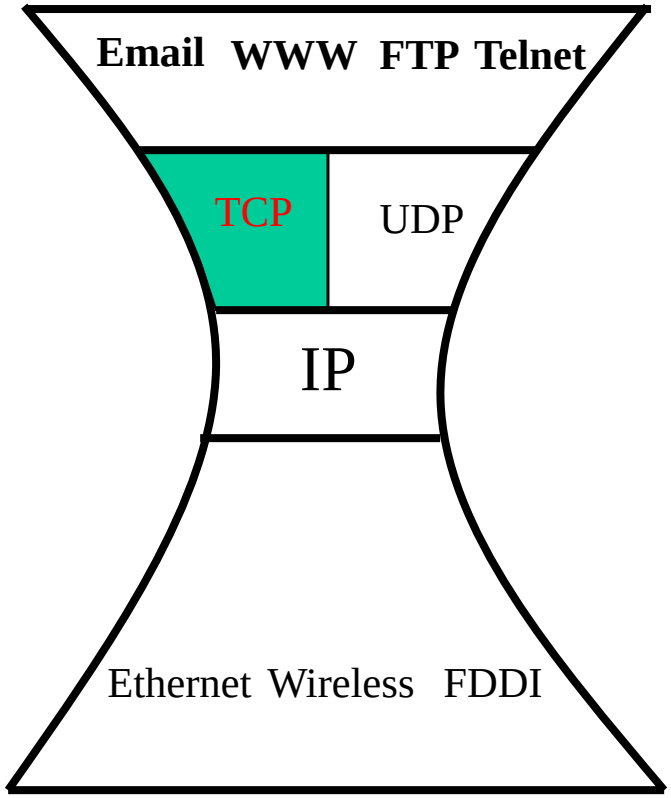
传输层: UDP

- 无连接的服务
- 不提供: 连接建立, 可靠性, 流控制, 拥塞控制, 时间或者带宽保证
  - 为什么会有UDP?



传输层: TCP

- 服务
  - 多路复用/多路解复用
  - 可靠的传输
    - 在发送和接收过程之间
    - 发送方和接收方所需要的设置: 面向连接的服务
  - 流控制: 发送方不会淹没接收方
  - 拥塞控制: 当网络过载时限制发送方
  - 错误检测
  - 不提供 时间和最小带宽的保证
- 接口:
  - 向（应用层）对等方发送数据包



Lecture 5

客户端-服务器模式

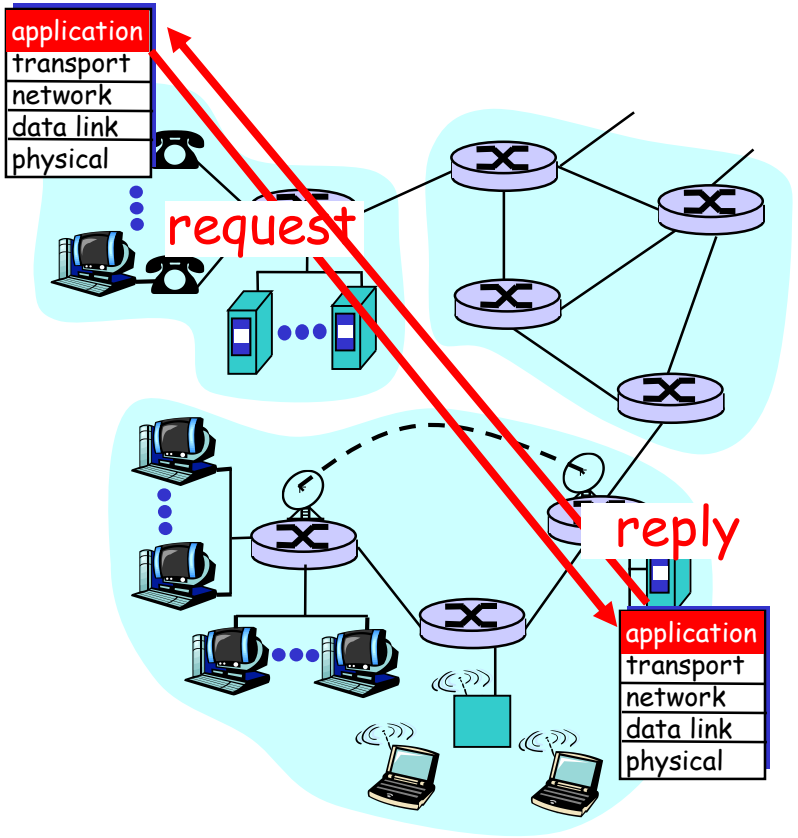
典型的网络应用程序有两部分：客户端和服务器

客户端 (C):

- 发起与服务器的连接 (“首先发言”)
- 通常从服务器请求服务
- 对于Web来说, 客户端在浏览器中实现; 对于电子邮件来说, 客户端在邮件阅读器中

服务器 (S):

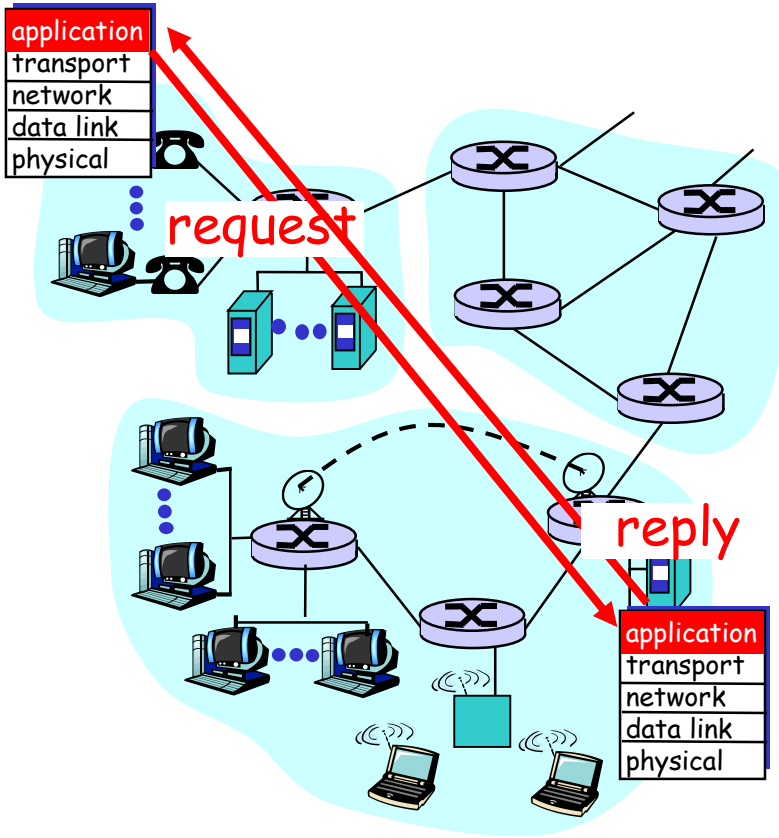
- 向客户提供请求的服务
- 例如, Web服务器发送请求的网页, 邮件服务器发送电子邮件



客户端–服务器模式：关键问题

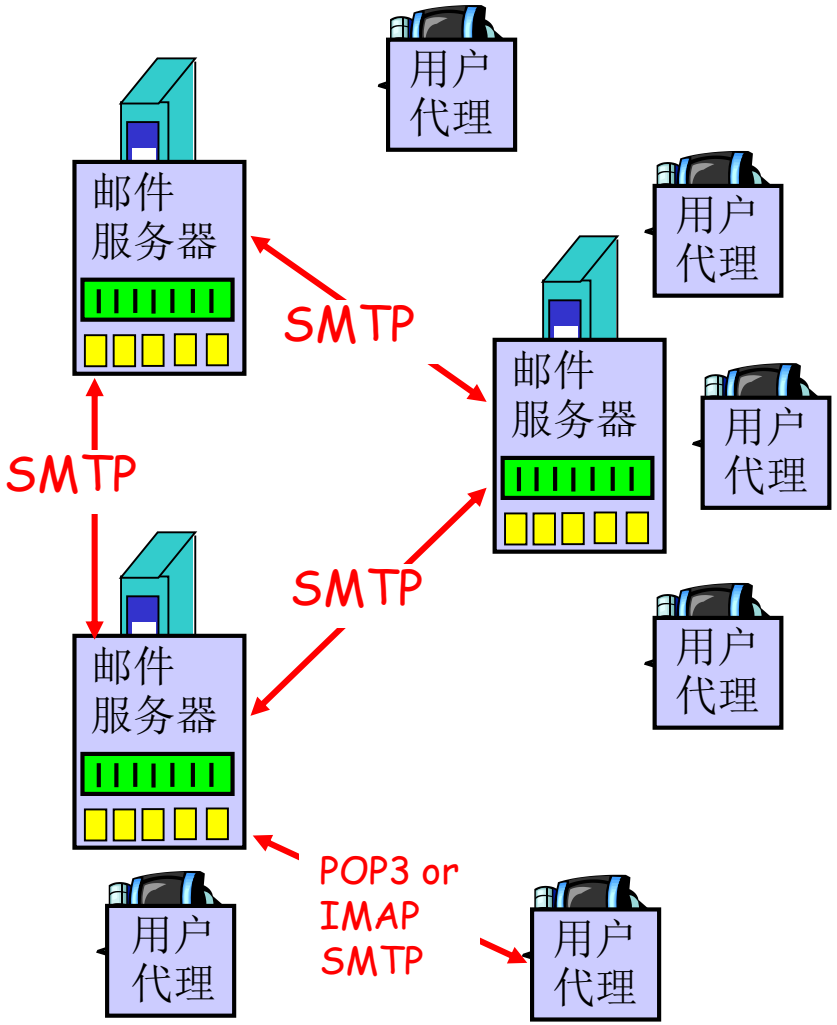
关于客户端-服务器(C-S)应用的关键问题

- 应用程序是否可扩展?
- 应用程序是否可伸缩?
- 应用程序如何处理服务器故障（变得鲁棒）?
- 应用程序如何处理安全问题?



Lecture 6

回顾: 电子邮件应用



电子邮件的主要设计特点

- 对不同的功能使用具有独立的协议
  - 邮件获取 (e.g., POP3, IMAP)
  - 邮件传输 (SMTP)
- 信封和消息体的独立使用(端到端的参数)
  - 信封: 简单/基础的请求去实现传输协议;
  - 消息体: 通过ASCII头和消息体实现细粒度的控制
    - MIME 类型作为自描述数据类型
- 响应中的状态码可以让信息易于解析

DNS 消息流:
两种类型的查询

递归查询:

- 完全由被连接的域名服务器解析域名

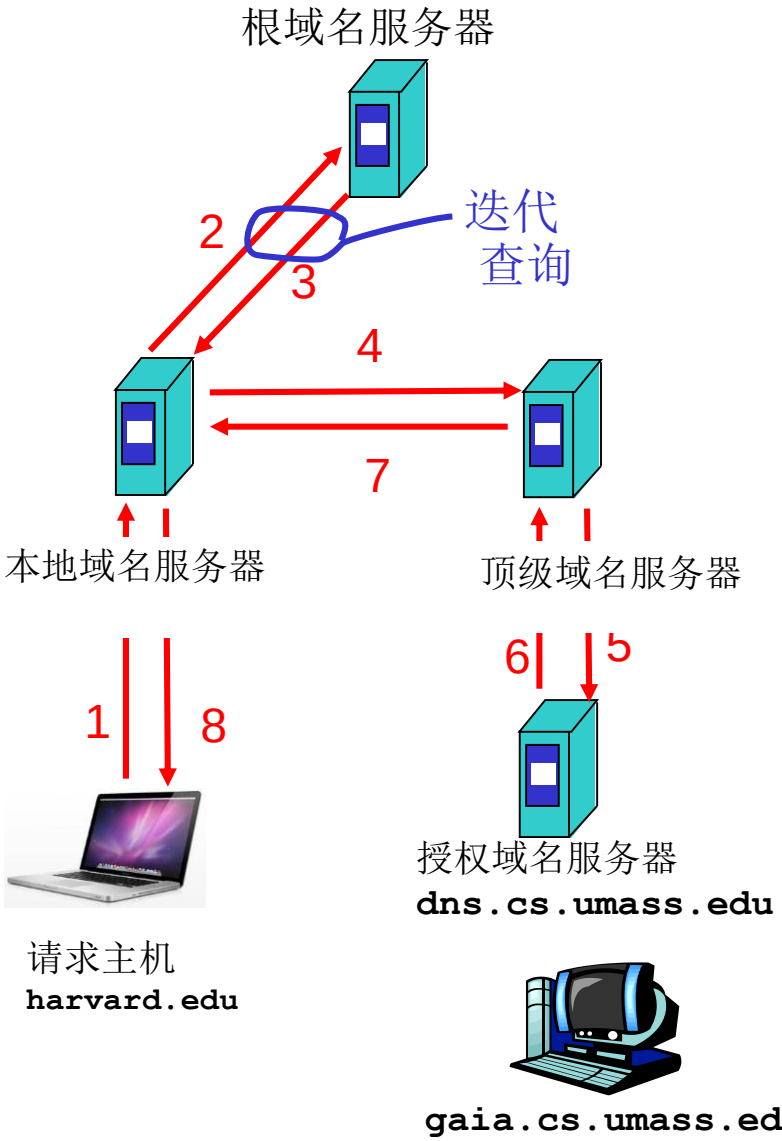
迭代查询:

- 被连接的服务器会返回下一个可连接进行查询服务器的域名
  - “我不知道域名是什么，但我可以向服务器询问”

典型的DNS消息流: 混合案例

- 主机只知道本地域名服务器
- 本地域名服务器是从DHCP获取域名信息, 或者被配置得到, e.g. /etc/resolv.conf

- 本地域名服务器帮助解析域名
- 本地域名服务器的好处(经常被叫做 解析器)
  - 简化客户端
  - 缓存/重用结果

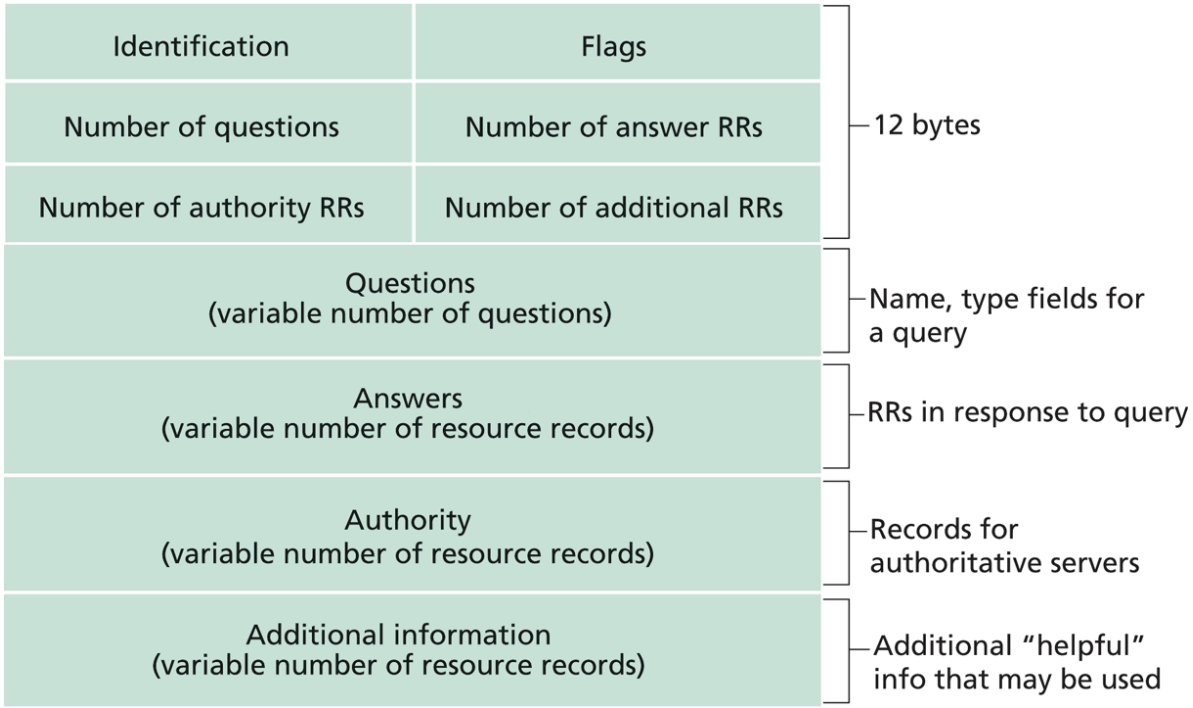


Lecture 7

https://www.ietf.org/rfc/rfc1035.txt

回顾: DNS 协议, 消息

一些特征: 通常使用 UDP (也可以使用 TCP); 查询和回复的消息具有相同的消息格式; *以名称的长度/内容编码; 简单的压缩; 作为服务器推送的附加消息*



Lecture 8

TCP 面向连接的多路分解

- TCP 套接字用四元组标识:
  - 源IP地址
  - 源端口号
  - 目的IP地址
  - 目的端口号

- 接受主机用所有的四个值去把分组导向恰当的套接字
  - 不同的连接/会话会自动分离到不同的套接字之中

Lecture 9

总结: 基础的套接字编程

- 他们相对简单
  - UDP: 数据报套接字
  - TCP: 服务器套接字, 套接字
- 套接字的主要功能是对应用进程进行多路复用或者多路分用
  - UDP 使用 (目的 IP, 端口)
  - TCP 使用 (源 IP, 源 端口,目的 IP, 目的 端口)
- 要关注编码与解码

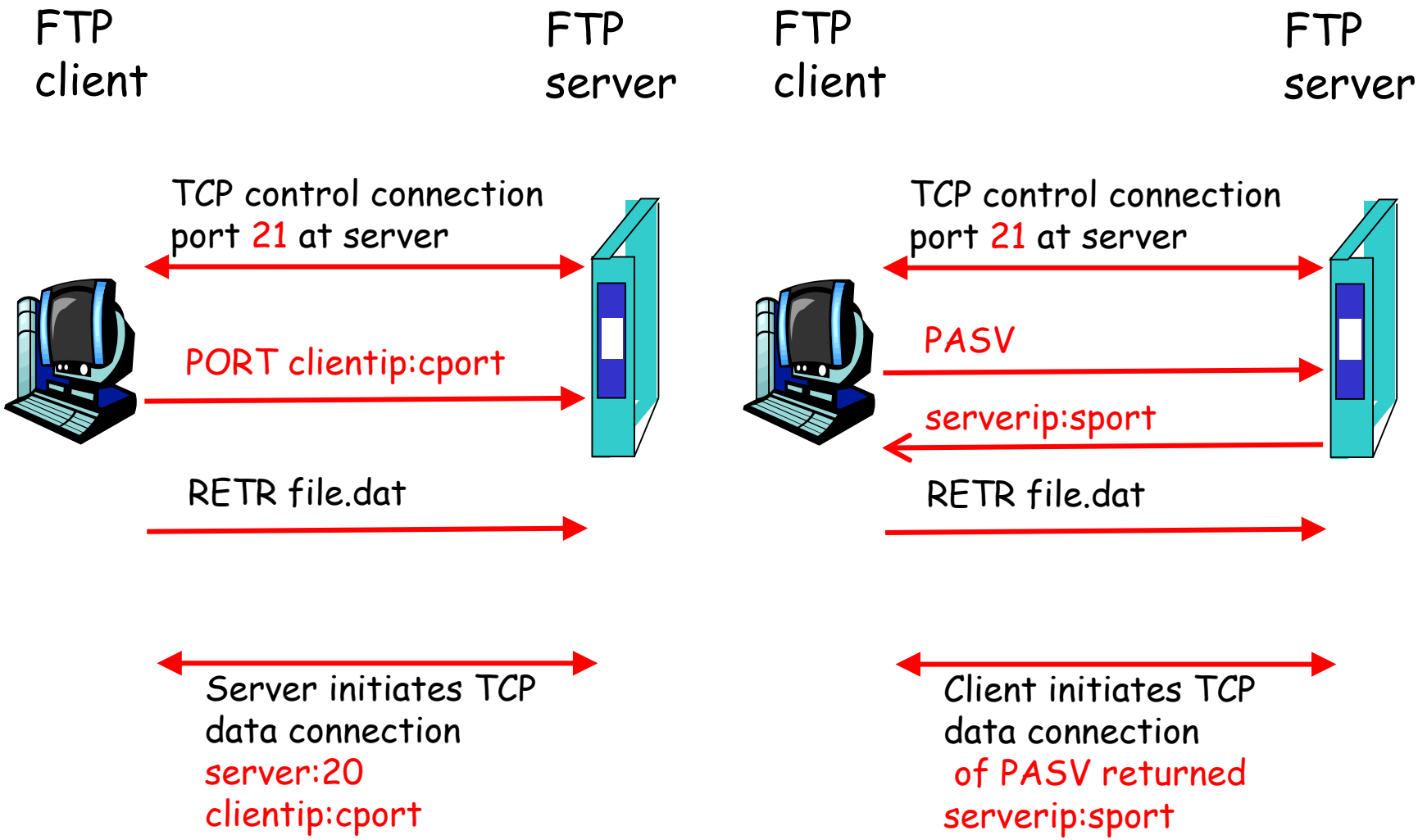
FTP: 一个具有独立的控制和数据连接的客户段-服务器应用程序

- 两种 TCP连接类型:
  - 控制连接: 在客户端，服务器之前交换命令和响应.
    “带外控制”
  - 数据连接: 每个从服务器来或者去的文件数据

讨论: FTP为什么要分离控制/数据连接?

Q: 如何去创建一个新的数据连接?

FTP PASV: 服务器指定数据端口,客户端发起连接

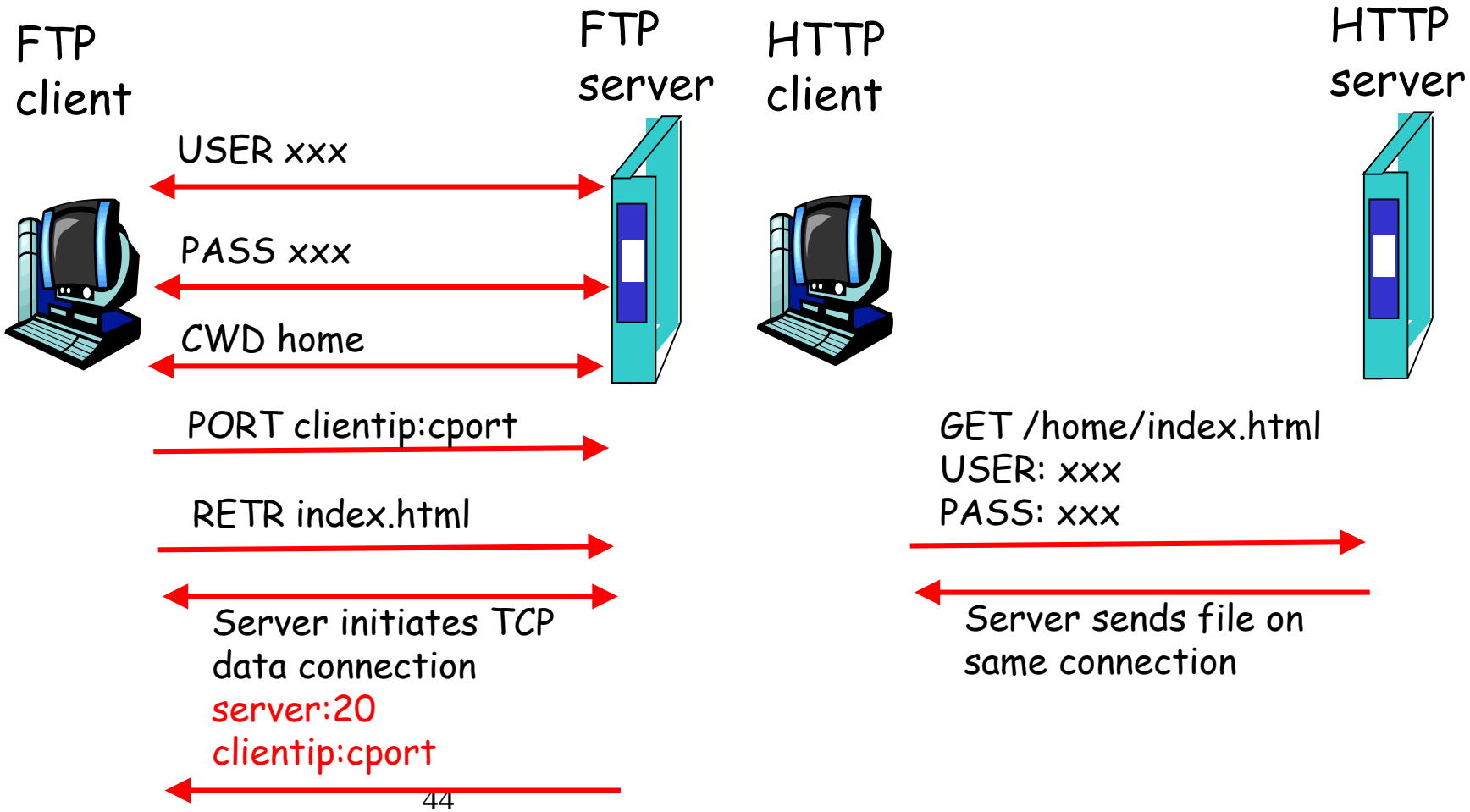


HTTP 1.0 消息流

- 服务器等待客户机的请求
- 客户机创建与服务器的TCP连接(创建套接字)，端口号为 80
- 客户端发送文档请求
- 网页服务器发回文档
- TCP 连接关闭
- 客户端解析文档来找到嵌入的对象(图片)
  - 重复上面的操作来获取每一个图片

HTTP1.0 消息流

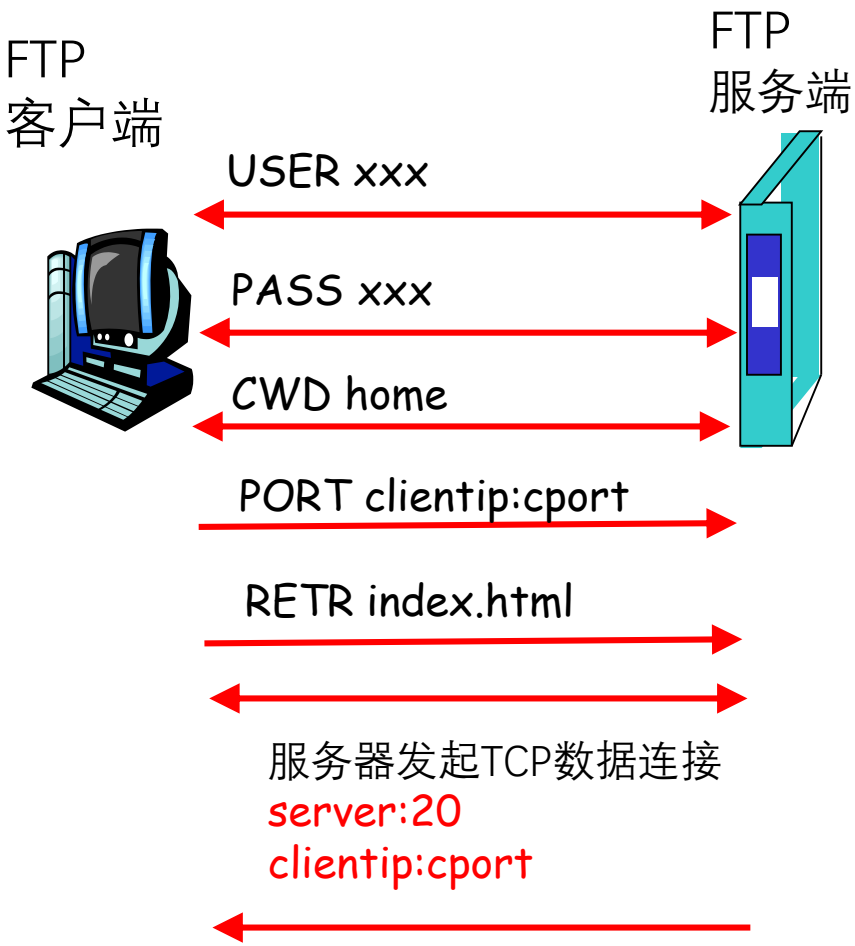
- HTTP1.0 服务器是无状态服务器: 每一个请求都是独立的



Lecture 10

重点总结：FTP

- 是一个有状态的协议
  - 通过类似的命令来建立状态：
    - USER/PASS, CWD, TYPE
- 存在多个TCP连接
  - 一个控制连接
  - 多个数据连接
    - 两种模式： PORT vs PASV
    - GridFTP：并行数据连接；
      分块数据传输；



有状态的用户-服务器交互：cookie

目标:

没有显式的应用级别对话

- 在回复信息中服务端发送“cookie”给客户端
  - **Set-cookie: 1678453**
- 客户端在之后的请求中携带cookie
  - **Cookie: 1678453**
- 服务端将呈现的cookie和已经存储的信息进行匹配
  - 身份验证
  - 记住用户偏好, 过去的选择

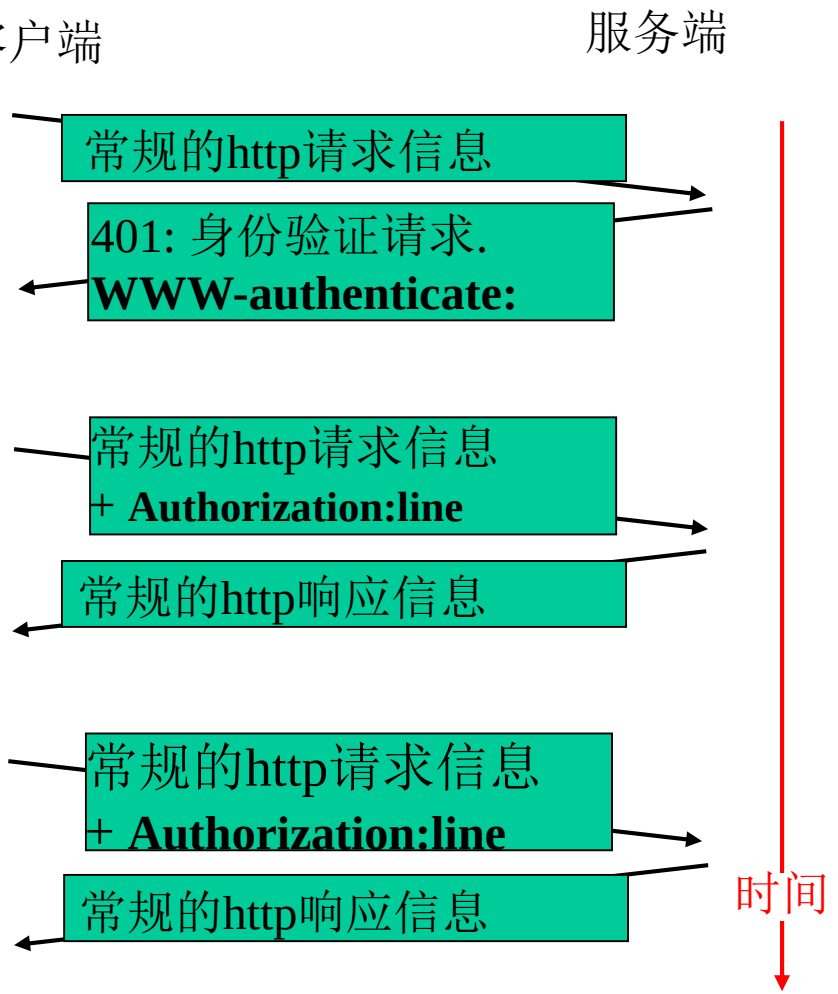
客户端请求认证

身份验证目标：

控制对服务文档的访问

- 无状态：客户端必须在每个请求中提供身份验证身份验证: 通常是姓名，密码
  - **Authorization:**
    请求中的第一行
  - 如果没有身份验证信息，服务端拒绝这次访问，并在响应信息的第一行发送
    WWW-authenticate

浏览器缓存名称和密码
以便于用户不需要重复输入



加速基本的HTTP/1.0的一些努力

- 减少获取的对象数量 [浏览器缓存]

- 减少数据量 [数据压缩]
- 头部压缩 [HTTP/2]

- 减少服务器获取内容的延迟 [代理缓存]
- 移除为了获取对象造成的额外的RTTs [持久 HTTP, 即HTTP/1.1]

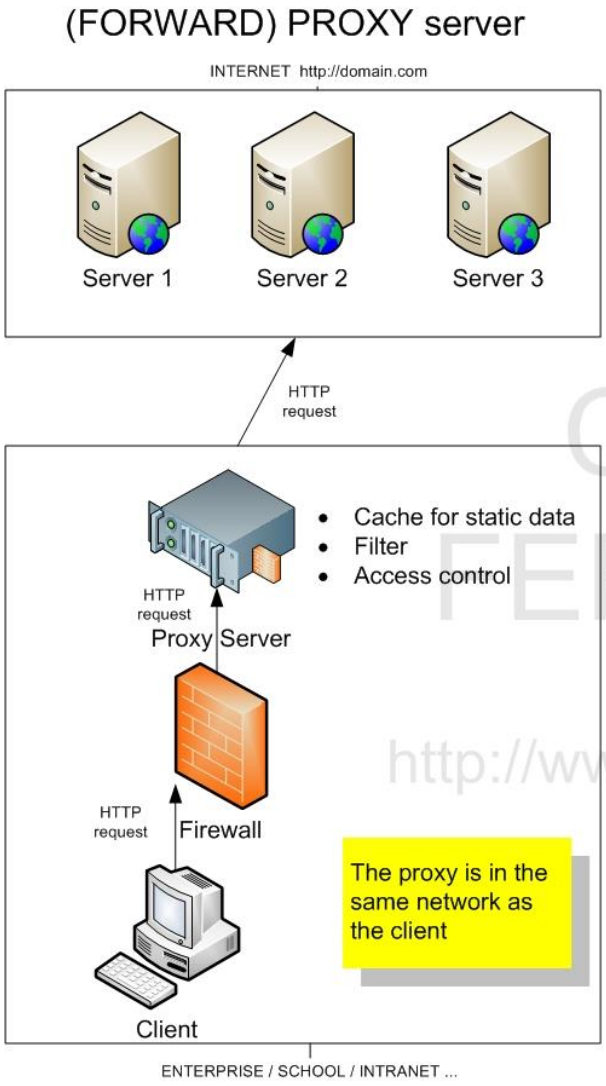
- 增加并发度 [多个TCP 连接]
- 异步获取 (多个流) 使用单个TCP [HTTP/2]

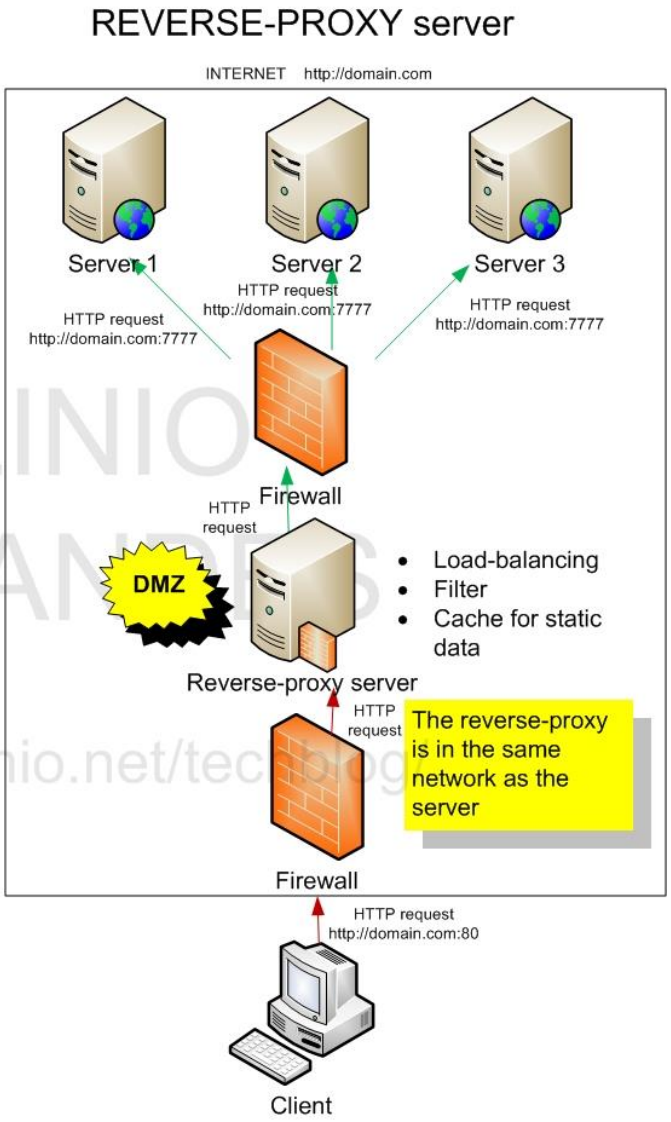
- 服务器推送[HTTP/2]

http://www.celinio.net/techblog/?p=1027

两种代理

通常在与客户端相同的网络中





通常在与服务端相同的网络中

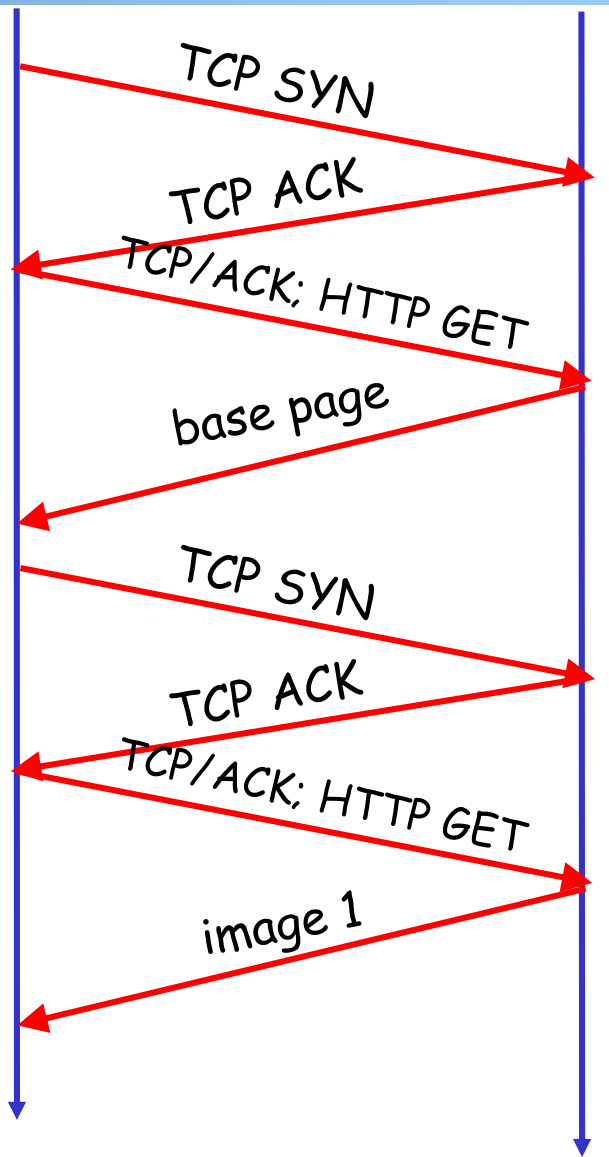
CELINIO FERNANDES
http://www.celinio.net/techblog/

Lecture 11

回顾: 基础 HTTP/1.0 的延迟

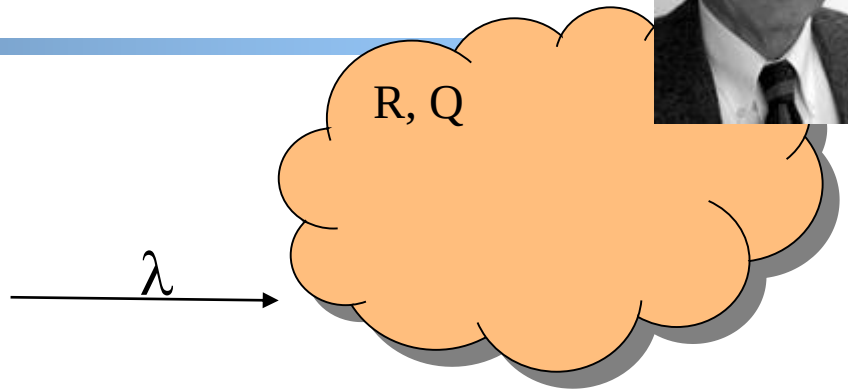
- 每个对象 >= 2 RTTs :
  - TCP 握手 --- 1 RTT
  - 客户端请求和服务器响应

    --- 至少 1 RTT (如果对象可以包含在一个数据包中)



背景: 利特尔法则 (1961)

- 适用于任何没有或低损耗的系统。
- 假设
  - 系统平均到达率 λ, 平均时间 R 和系统平均请求数 Q
- 那么，Q，λ,，和 R的关系是:

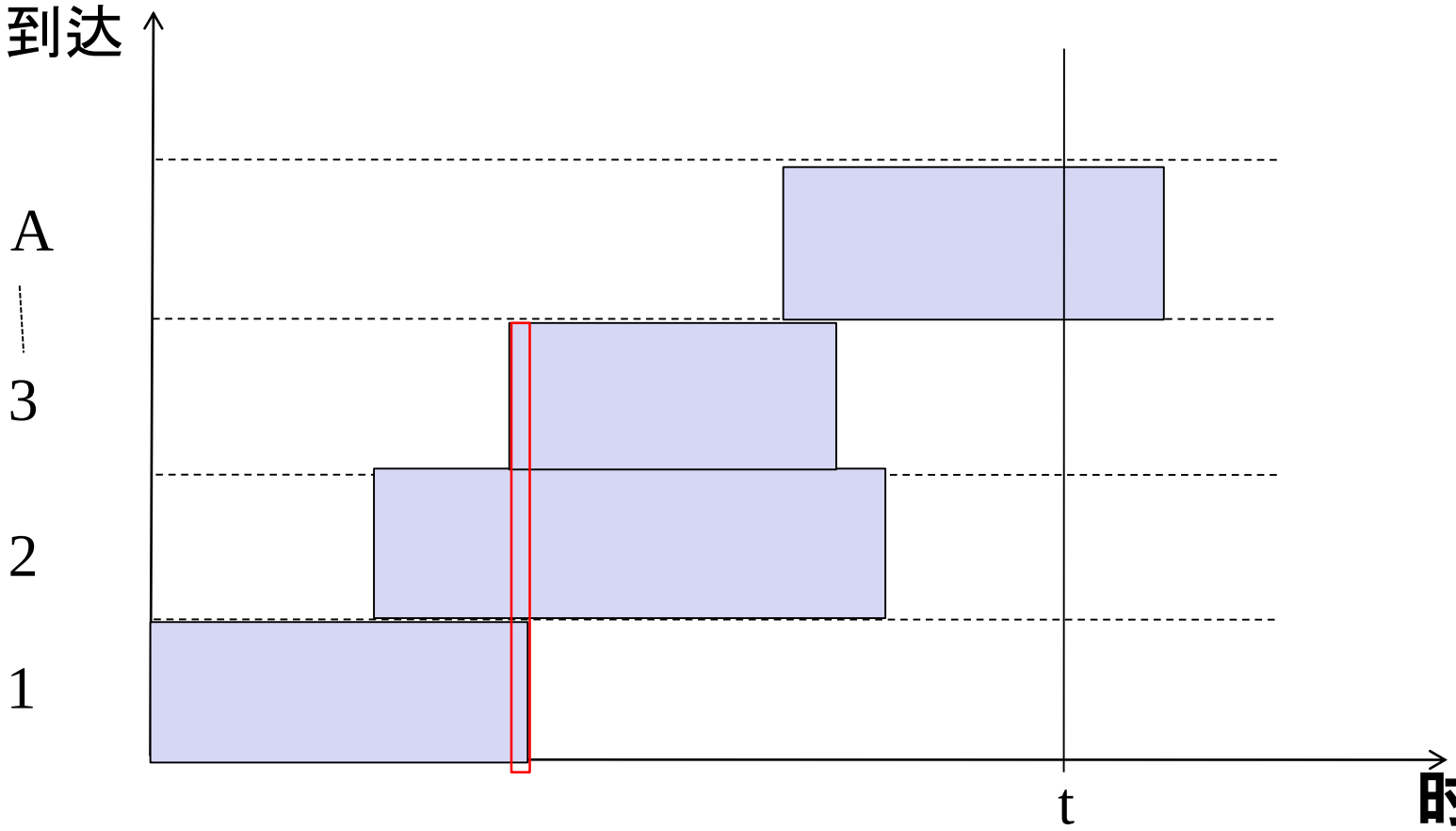


$$Q = \lambda R$$

示例:厦门大学每年招收3000名学生，平均每名学生停留4年，有多少学生入学?

利特尔法则 ：证明

$$Q = \lambda R$$



$$\lambda = \frac{A}{t} \quad R = \frac{\text{面积}}{A} \quad Q = \frac{\text{面积}}{t}$$

Lecture 12

总结：程序正确性分析

- 安全性
  - 禁止同时读写；解决写/写冲突
    - 在读取或者修改共享数据队列Q前获取锁（Q.wait_list）
  - Q.remove() 必须保证队列不空
- 活跃性 (进展)
  - 调度线程总是能给队列Q添加任务
  - 队列Q中的每个连接任务将会在有限时间内被处理
- 公平性
  - 例如，在一些场景下，设计者想要线程间负载均衡

Lecture 14

总结：高性能网络服务器

- 避免阻塞（以达到最大吞吐量）
  - 引入线程
- 限制线程的开销
  - 线程池，异步IO
- 共享变量
  - 同步，包括锁，同步原语
- 避免忙等待
  - 等待/唤醒机制：FSM；异步通道/Future对象/Handler
- 扩展性/鲁棒性
  - 对接口的设计和编程语言支持
- 系统建模和度量
  - 队列分析，操作性分析

为什么使用多台服务器？

- 扩展性
  - 实现更大的吞吐量（克服单台服务器的局限）
    - 单台服务器的处理能力被硬件所限制
      - 处理能力（CPU/带宽/硬盘吞吐量）
      - 存储能力（硬盘存储量/内存大小）
  - 实现跨域的低延迟
    - 在以光纤为主要传输媒介的网络中，光速是有限的
    - 次优的传输路径和传输延迟进一步增加了总延迟

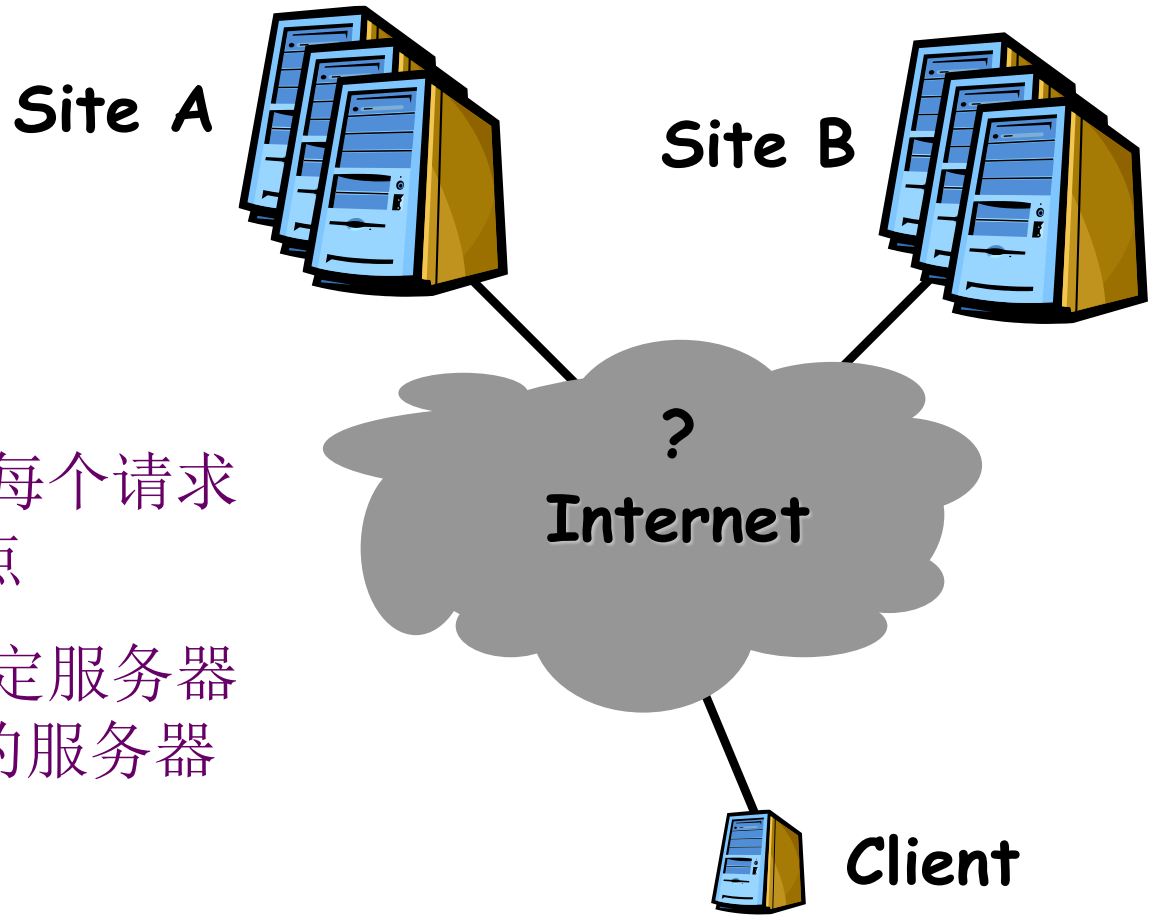
为什么使用多台服务器？

- 冗余性和灾备能力
  - 管理/维护（例如，增量式更新）
  - 冗余性（例如，克服单机宕机）

为什么使用多台服务器？

- 从系统架构和软件架构角度看
  - 资源分布在不同机器上（例如，实现数据库服务器的备份服务器；从第三方访问资源，彼此的松耦合增加了灵活性）
  - 安全性（例如，前端、业务逻辑和数据库分离，增加了系统的安全性）
- 如今，我们主要关注利用多台同质服务器提高扩展性。

请求路由概览

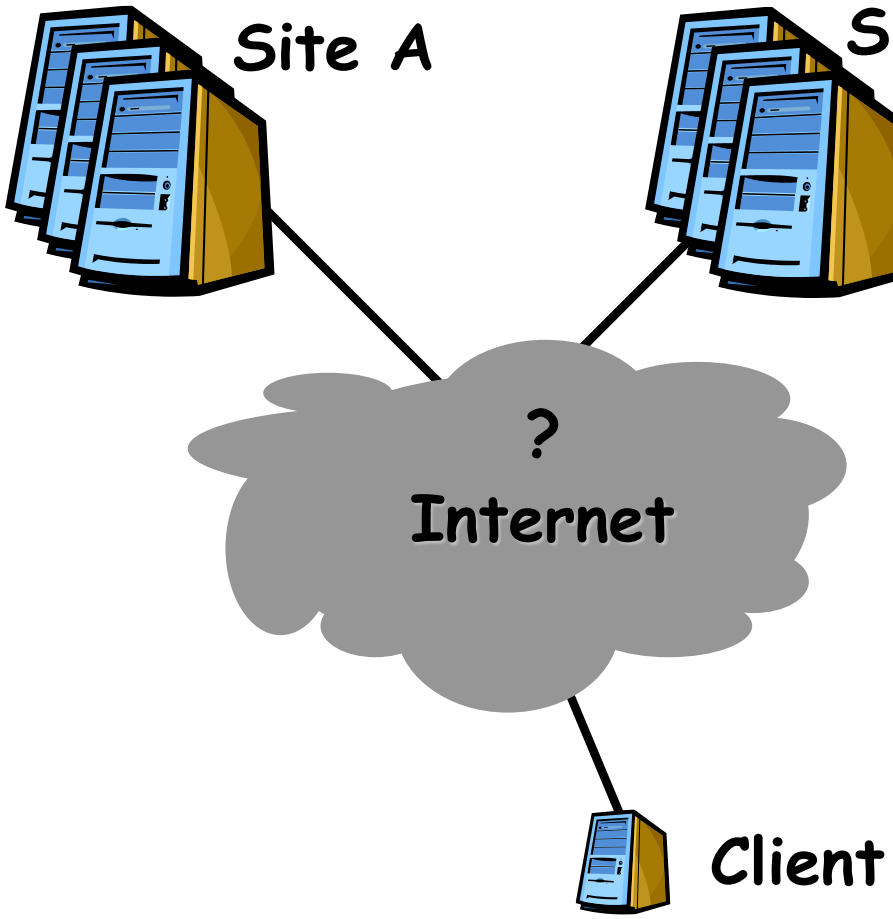


-全局请求路由：为每个请求选择一个服务器站点

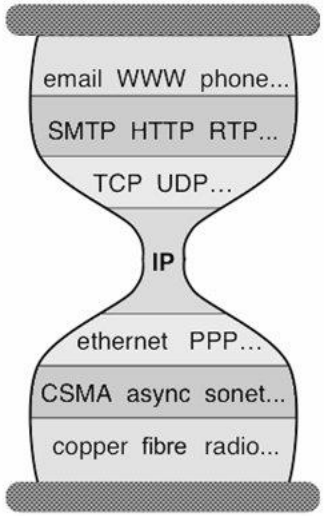
-局部请求路由：确定服务器站点后，选择具体的服务器

请求路由：基础架构

- 主要组件
  - 服务器状态监视
    - 负载 (incl. failed or not);记录它能处理什么类型的请求
  - 网络路径属性估计
    - 例如，客户端和服务器间的带宽，延迟，丢包和网络花费
  - 服务器指派算法
  - 请求方向的机制

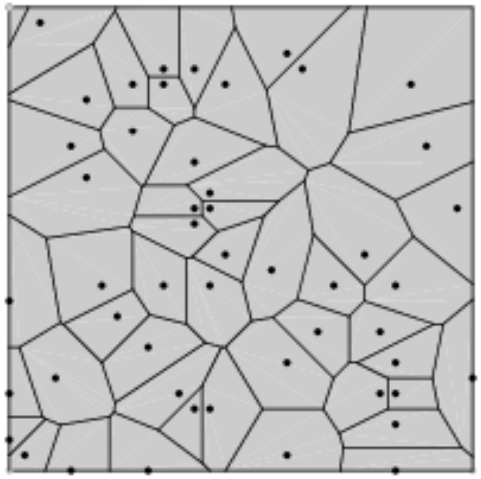


客户端定向机制



- 关键挑战
  - 需要处理大量客户端请求
- 机制的类型
  - 应用层, 例如,
    - 应用或用户被赋值多个候选服务器名
    - HTTP重定向
  - DNS层:域名解析服务返回多个服务器地址
  - IP层:相同的IP地址表示多个物理机
    - IP任播:多个服务器共享相同IP，在因特网的不同部分声明。不同地域的客户端向不同服务器请求
    - 间接智能交换:服务器IP地址是虚拟地址，由一簇物理服务器共享

两层定向



- 高层DNS：检索到渐近服务器，并定向到低层DNS缩小范围;

输入: dscj.akamaiedge.net 和客户端IP,
输出:域 (低层) DNS

- 低层DNS :负责管理不同IP地址的服务器集群

输入：e12596.dscj.akamaiedge.net 和客户端IP
输出:具体的服务器

讨论

- 多台服务器使用DNS的优点
  - 利用现有的DNS的特性（例如，别名，层次化命名）
  - 利用现有的DNS发布和优化
- 使用DNS的缺点
  - 分布式缓存降低了相应速度
  - 只能以IP地址为单位