

第一章 计算机网络概述

什么是 Internet:

从具体构成的角度来看——

数以亿计的互联计算设备：**主机=端系统** 运行网络应用程序

分组交换设备：转发分组（数据块） 如：路由器，交换机

通信链路：光缆，同轴电缆，无线电，卫星 传输速率：带宽

网络就是计算设备、分组交换设备、链路的集合：由一个组织管理

Internet/互联网：“网络的网络” 互联的 ISP (Internet Service Provider)

协议无处不在：控制信息发送与接收。如，HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4/5G, Ethernet

Internet 标准：RFC: Request for Comments, IETF: Internet Engineering Task Force

从服务的角度来看——

为应用提供通信服务的基础设施：

Web、流媒体视频、电话会议、email、游戏、电子商务、社交网络、互联设备……

为分布式应用提供的编程接口：

提供允许发送/接收型应用程序连接的接口，以便使用互联网传输服务。为应用提供不同服务选项，类似于邮政服务。无连接不可靠传输服务//面向连接的可靠传输服务

什么是协议：

协议定义了网络实体之间发送和接收消息的格式、顺序，以及传输或接收消息时采取的行动。两台计算机通信时，对传送信息内容的理解、信息的表示方式以及各种情况下的应答信号都必须遵循一个共同约定，协议。协议是按功能分成若干层次，**如何分层以及各层中具体采用的协议综合**，称为网络的分层体系结构。

网络边缘：主机、接入网、物理媒体：

主机：客户端和服务器。服务器通常在数据中心

接入网，物理媒体：有线、无线通信链路

网络核心：互相连接的路由器、网络的网络（Internet）

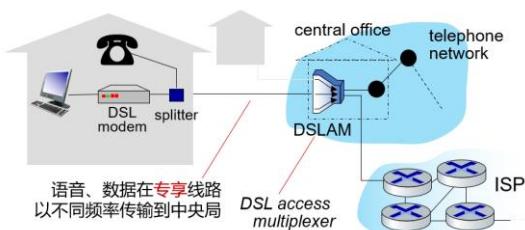
Q：怎样将端系统和边缘路由器连接？

A：住宅接入网络、机构接入网络（学校、公司）、无线接入网络（WiFi, 4G/5G）

接入网的方式：DSL、线缆接入、家庭网络、企业网络、数据中心网络——

接入网：Digital Subscriber Line (DSL)。

使用已有的到电话公司中央局（central office）DSLAM 的电话线：DSL 上的数据被传输到互联网、DSL 上的语音被传输到电话网。24-52 Mbps 专享下行传输速率，3.5-16 Mbps 专享上行传输速率。



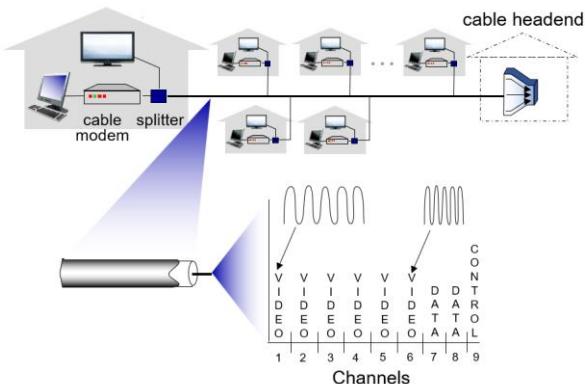
接入网：线缆接入。

频分复用 frequency division multiplexing (FDM): 不同的信号在不同的频段中传输。

数据、电视信号以不同频率传输，从而共享线缆网络

HFC: hybrid fiber coax 混合光纤同轴：非对称：最高 40 Mbps – 1.2 Gbps 下行传输速率，30-100 Mbps 上行传输速率。线缆和光纤网络将家庭用户连接到 ISP 路由器，各用户共享

到线缆头端的接入网络，与 DSL 不同，DSL 每个用户一个专用线路到中央局 (central office)

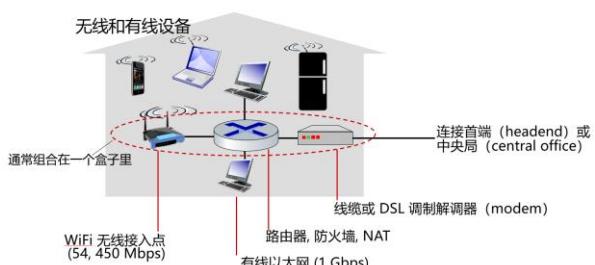


接入网：家庭网络。

端系统通过共享的无线接入点连接到路由器，通过基站，又名“接入点”。

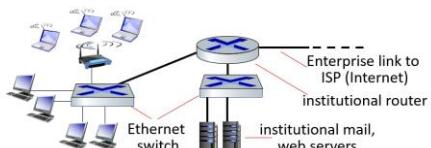
无线局域网 Wireless local area networks (WLANs): 通常在建筑物内部或周围 (约 30 米),
802.11b/g/n (WiFi): 11, 54, 450 Mbps 传输速率

广域蜂窝接入网 Wide-area cellular access networks: 由电信运营商提供 (10's km), 10's Mbps, 4G/5G 蜂窝网络



接入网：企业网络。

用于企业，大学等机构。混合有线和无线技术，混合连接交换机或路由器。以太网：以 100Mbps, 1Gbps, 10Gbps 有线接入。WiFi：以 11, 54, 450 Mbps 无线接入



接入网：数据中心网络。

高带宽链路 (10s 到 100s Gbps) 将数百到数千台服务器连接到一起，并连接到互联网

主机：传输数据

主机发送功能：接收应用的数据信息——将数据切分为长为 L bits 的小块，即分组 packets——以传输速率 R 将分组传输到接入网——链路传输速率，也即链路容量，也即链路带宽。

$$\text{分组传输时延} = \frac{\text{将L bits的包传入}}{\text{链路所需时延}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

链路：物理媒体

bit: 在发送端-接收端之间传播的数据单位。

物理链路：连接每个发送端-接收端之间的物理媒体

导引型媒体：信号沿着固体媒介传播：同轴电缆、光纤、双绞线

非导引型媒体：信号自由传播，例如无线电

双绞线 Twisted pair (TP): 两根绝缘铜导线拧合。5类: 100 Mbps, 1 Gbps 以太网。6类: 10Gbps 以太网。4对双绞线构成网线，网线接口水晶头。

同轴电缆: 两根同轴的铜导线。双向。宽波段: 电缆上有多个频道，每个频道 100's Mbps 光纤和光缆: 玻璃纤维携带光脉冲，每个脉冲 1 bit。高速工作: 高速点对点传输 (10's-100's Gbps)。低错误率: 中继器相距很远，不受电磁噪声影响。

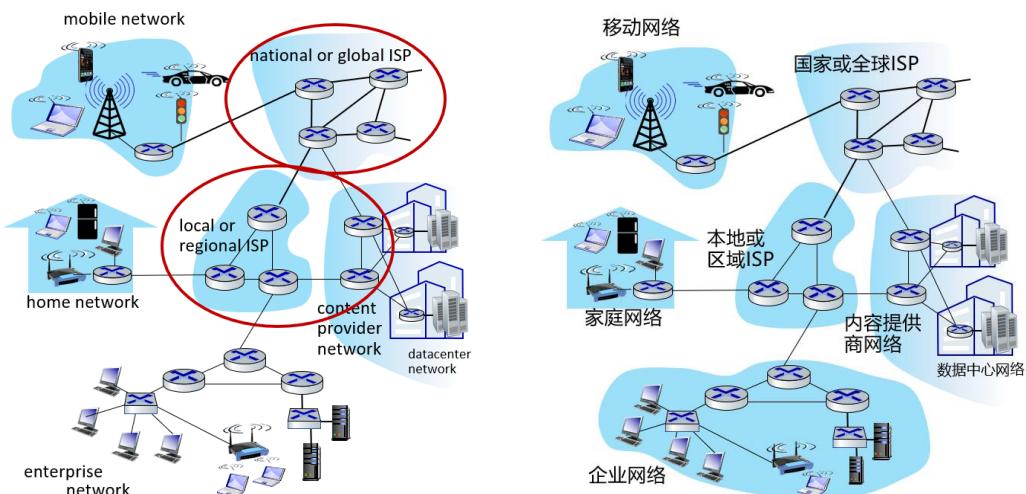
无线电波: 在电磁波谱中以不同“频带”携带信号，无需物理“电线”。广播，“半双工”(发送方到接收方)。传播环境效应: 反射、物体阻挡、干扰/噪声

无线电链路类型: Wireless LAN (WiFi), 10-100's Mbps; 10's of meters。wide-area (e.g., 4G/5G 蜂窝), 10's Mbps (4G) over ~10 Km。蓝牙: 电缆替代，短距离，有限速率。地面微波: 点对点，45 Mbps 信道。卫星: 不超过 100 Mbps (星链) 下载链路, 270 msec 端到端延迟(地球同步卫星)。

网络核心: 分组交换, 电路交换, 互联网架构:

网络核心: 互联网路由器的网状结构

分组交换 (Packet Switching): 主机将应用层信息分成一个个单位——分组。网络将分组从一个路由器转发到下一个相邻路由器，跨越从源头到目的地路径上的链路。



网络核心的两个关键功能: 转发、路由

转发: Forwarding aka switching。本地操作，将到达的数据包，从路由器的输入链路移至相应的路由器输出链路 (通过本地转发表)

路由: 全局操作: 确定数据包的源-目的路径，通过路由算法实现

分组交换: 存储和转发

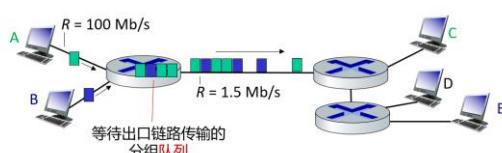
分组传输延迟: 需要 L/R seconds 将 L -bit 的分组传输 (推送) 到 R bps 的链路中

存储和转发: 必须要在整个分组都到达路由器后，才能将分组传输到下一段链路

分组交换: 排队和丢包

排队发生在工作到达的速度快于服务速度时。分组排队和丢包: 如果一段时间内，链路到达的速率超过了链路传输的速率 (单位为 bps)。分组会排队，等待被出口链路传输

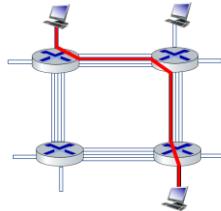
如果路由器的缓存 (buffer) 被填满，分组会被丢弃。



分组交换的替代方案：电路交换 (Circuit Switching) FDM 和 TDM

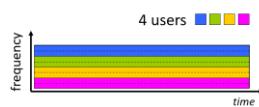
端到端的资源（如频率、时间片）被分配给从源端到目的端的呼叫“call”

- 图中，每段链路有4条线路
 - 该呼叫获取了顶端链路的第2条线路和右侧链路的第1条线路
- 独享资源：不与他人共享
 - 每个呼叫一旦建立性能就有保证
- 如果呼叫未发送数据，被分配的资源就会被浪费（no sharing）
- 通常被用于传统电话网络



频分复用 Frequency Division Multiplexing (FDM)

- 光电频率被划分为窄频段
- 每个呼叫获得自己的频段，可按该窄频段的最大速率进行传输



时分复用 Time Division Multiplexing (TDM)

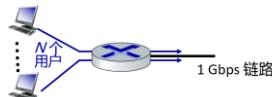
- 时间被划分为时隙
- 每个呼叫获得周期性时隙，仅在其时隙内，可按全频段的最大速率进行传输



分组交换 vs 电路交换

示例：

- 1 Gb/s 链路
- 每个用户：
 - 活跃时 100 Mb/s
 - 在 10% 的时间活跃



Q：在电路交换和分组交换中，多少用户可以使用这个网络？

- 电路交换：10个用户
- 分组交换：有35个用户时，同时有超过10个用户活跃的概率低于0.0004*

(*概率论经典问题) $1 - \sum_{k=0}^{10} \binom{35}{k} p^k (1-p)^{35-k}$

Q：分组交换允许某个用户在其他用户不使用网络时以更快的速度发送数据？

分组交换适用于突发式（bursty）数据——部分时间有数据要发送，其他时候没有资源共享，更简单，不需要建立呼叫。过度使用可能会导致网络拥塞：分组延迟和缓存溢出导致的丢包，从而需要协议保证可靠数据传输，拥塞控制。

Q：如何使用分组交换提供类似电路交换的行为？

用于保证音频/视频应用所需的带宽

尚未完全解决，有一些技术试图尽可能让分组交换接近电路交换。

Q：生活中有哪些预约服务（电路交换）和按需服务（分组交换）的例子？

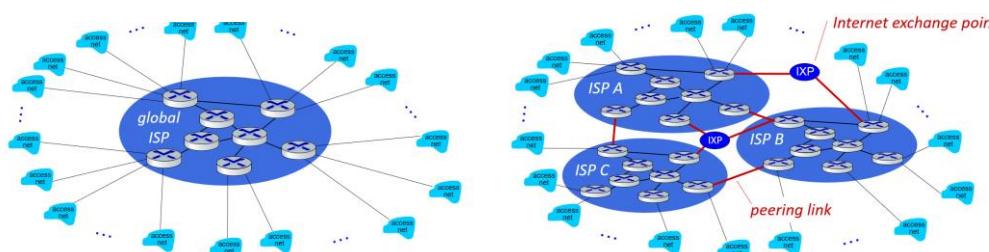
互联网结构是“网络的网络”

主机通过接入 ISPs 连接到互联网，接入 ISPs 必须是互联的，因此任何地方的两个主机都可以发送分组给对方。这导致“网络的网络”非常复杂，经济和国家政策驱动互联网演化

Q：数以百万的接入 ISP，如何将他们互联？

两两相连？ n^2 ，完全图，复杂且不可扩展，不可行。

将每个 ISP 连接到一个全球转运的 ISP？如果全球 ISP 是可行的业务，有利可图，一定会有竞争……但竞争者之间也需要互联，以保证全球网络联通。业务会继续细分，区域 ISP 出现，用于将接入网连接到 ISP。内容提供商网络（例如 Google, Microsoft, Akamai）可能会构建他们自己的网络，用于将服务和内容部署到离端用户更近的地方。



在网络的最中心是少数的充分连接的大型网络（分布广、节点有限、但是之间有多重连接）

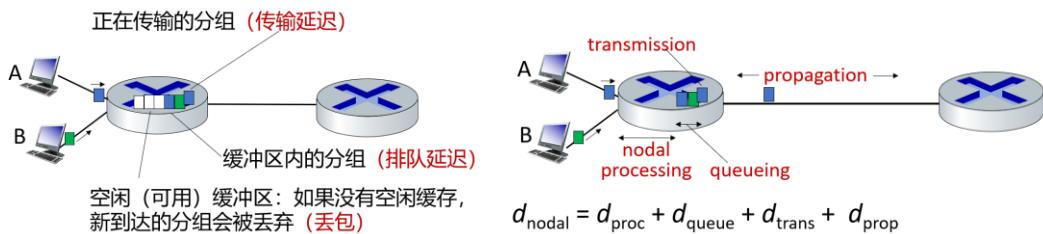
“tier-1” commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT)，提供国家或者国际范围的覆盖
content provider network (e.g., Google, Facebook)：使用专用网络将它们的数据中心接入互联网，方便周边用户的访问；通常会绕过 Tier-1 ISP 和区域 ISP。

- POP：高层 ISP 面向客户网络的接入点，涉及费用结算
- 如一个低层 ISP 接入多个高层 ISP，多宿（multi home）
- 对等接入：2 个 ISP 对等互接，不涉及费用结算
- IXP：多个对等 ISP 互联互通之处，通常不涉及费用结算
- ICP 自己部署专用网络，同时和各级 ISP 连接

分组交换网中的时延、丢包和吞吐量：

分组延迟和包丢失是如何发生的？分组在交换机缓冲区排队，等待传输机会，当链路到达速率（短暂）超过出口链路容量时，队列长度增加。

分组丢失发生在容纳排队分组的存储空间已满时。



分组延迟的四个源头：(右上图) (proc 一般小于毫秒级) (trans 和 prop 区分)

d_{proc} : 节点处理延迟：检查比特位差错、检查分组首部和决定将分组导向哪个出口链路。

d_{queue} : 排队延迟：在出口链路等待传输的时间，取决于路由器的拥塞程度。

d_{trans} : 传输延迟： L : 分组长度(bits)、 R : 链路传输速率(bps)—— $d_{\text{trans}} = L/R$

d_{prop} : 传播延迟： d : 物理链路长度、 s : 传播速度 ($\sim 2 \times 10^8$ m/sec)—— $d_{\text{prop}} = d/s$

a: 分组平均到达速率 L : 分组长度 (bits) R : 链路带宽 (比特传输速率)

$$\frac{L \cdot a}{R} : \frac{\text{比特到达速率}}{\text{比特服务速率}} \quad \text{"流量强度"}$$

$L/R \sim 0$: 平均排队延迟很小

$L/R \rightarrow 1$: 平均排队延迟很大

$L/R > 1$: 比特到达速率超出服务能力，延迟趋向于无穷大

真实互联网延迟和路由是怎样的？

traceroute 程序：提供从源端、经过路由器、到目的端的端到端延迟测量。对于每个正整数 i ：沿着目的的路径，向每个路由器发送 3 个 time-to-live 值设为 i 的探测分组，第 i 个路由器会向发送方返回分组，发送方测量发送和返回之间的时间间隔。

traceroute: gaia.cs.umass.edu to www.eurecom.fr

从 gaia.cs.umass.edu 到 cs-gw.cs.umass.edu 的三个延迟测量结果

```

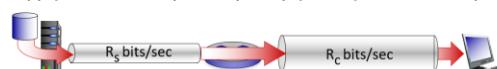
1 cs-gw (128.119.240.254) 1 ms 1 ms 2 ms
2 border-rt-1-rt-fa5-1-0.gw.umass.edu (128.119.3.145) 1 ms 1 ms 2 ms 到 border-rt-1-rt-fa5-1-0.gw.umass.edu 的三个延迟测量结果
3 cht-vprn.qw.umass.edu (128.119.3.146) 5 ms 5 ms 5 ms
4 in1-at1-0-0-19.wor.vbrn.net (204.147.132.129) 16 ms 11 ms 13 ms
5 in1-s07-0-0-wae.vbrn.net (204.147.136.136) 21 ms 18 ms 18 ms
6 abilene-vbrn.abilene.ualberta.ca (198.32.11.9) 22 ms 18 ms 22 ms
7 mycm-wash.abilene.ualberta.ca (198.32.8.46) 22 ms 22 ms 22 ms 跨洋链路
8 60.210.102.129 (60.210.102.129) 109 ms 102 ms 104 ms
9 de2-1.rtr1.de.geant.net (62.40.96.129) 109 ms 102 ms 104 ms
10 de.fr1.fr.geant.net (62.40.96.50) 113 ms 121 ms 114 ms
11 renater-gw.fr1.fr.geant.net (62.40.103.54) 112 ms 114 ms 112 ms 看起来延迟减少了，为什么？
12 nio-n2.cssi.renater.fr (193.51.209.13) 111 ms 114 ms 116 ms
13 r312-nice.cssi.renater.fr (195.220.98.102) 125 ms 125 ms 124 ms
14 r312-nice.cssi.renater.fr (195.220.98.110) 126 ms 126 ms 124 ms
15 eurecom-valbonne.r312.ft.net (193.48.50.54) 135 ms 128 ms 133 ms
16 194.214.211.25 (194.214.211.25) 126 ms 128 ms 126 ms
17 ...
18 * 表示无应答 (探测分组丢失或路由器不回复)
19 fantasia.eurecom.fr (193.55.113.142) 132 ms 128 ms 136 ms

```

分组丢失：链路的队列缓冲区容量有限。当分组到达一个已满的队列时，分组将会丢失。丢失的分组可能会被之前的节点、源端或系统重新传输，或根本不重传。

吞吐量：在发送端和接收端之间传送的速率 (bits/单位时间)

瞬间吞吐量：在一个时间点的速率。平均吞吐量：在一个长时间内的平均值



瓶颈链路：端到端路径上，限制端到端吞吐量的链路

$R_s < R_c$ 平均端到端吞吐量是多少？ R_s , 瓶颈链路是 R_s 那一段

协议层次及其服务类型：

层次：每个层次实现一种服务。通过本层的协议动作，依赖于下层提供的服务。

为什么要分层？概念化：结构清晰，便于标示网络组件，以及描述其相互关系，便于讨论的分层参考模型。模块化：更新、维护系统组件更为容易，改变某一层服务的实现对系统中的其他层次是透明的。

分层思想被认为有害的地方？

分层互联网协议栈：

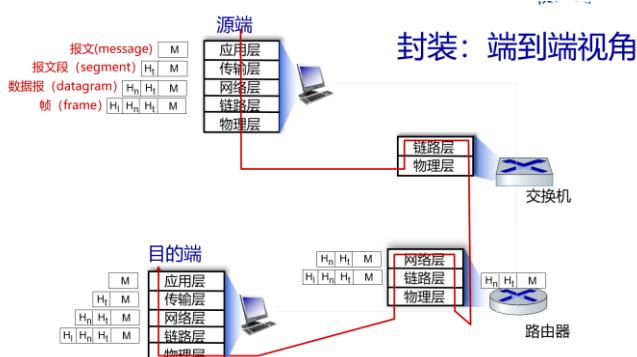
应用层：支持网络应用 HTTP, IMAP, SMTP, DNS。应用层使用传输层的服务交换报文 (message)，来实现一些应用服务。

传输层：进程到进程的数据传输 TCP, UDP。传输层协议使用网络层的服务将 M 从一个进程传输到另一个进程。传输层协议将应用层的报文 M 和传输层的报头 H_t 封装在一起，创造出传输层报文段 (segment) H_t 被传输层协议用来实现其服务。

网络层：对数据报从源到目的进行路由 IP，路由协议。网络层协议使用链路层的服务，将传输层报文段 [H_t | M] 从一个主机传输到另一个。将传输层的报文段 [H_t | M] 和网络层 H_n 封装在一起，创造出网络层数据报(datagram)，H_n 被网络层协议用来实现其服务。

链路层：相邻网络节点间的数据传输 以太网, 802.11 (WiFi), PPP。链路层协议使用物理层的服务，将数据报 [H_n] [H_t | M] 从一个节点传输到相邻节点。链路层协议将网络层数据报 [H_n] [H_t | M] 和链路层报头 H_l 封装在一起，创造出链路层帧 (frame)。

物理层：在线路上传送 bits



TCP/IP 协议栈：沙漏型结构：

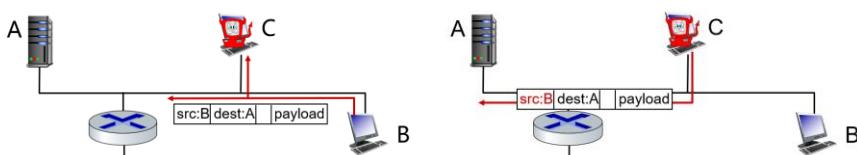
应用层协议多种多样，物理层和链路层技术层出不穷，但都依赖于相对固定且简单的 TCP/IP 网络核心进行通信。网络设计与架构的端对端原则：聪明终端，简单网络。提升可扩展性，适应爆炸性增长，在不可靠部件上建立可靠系统。

网络安全：

Internet 在最初设计时，并没有考虑太多安全问题。最初版本：“一群相互信任的用户连接到一个透明的网络上”，互联网协议设计者正在努力弥补漏洞，安全的考虑需要在所有的分层！

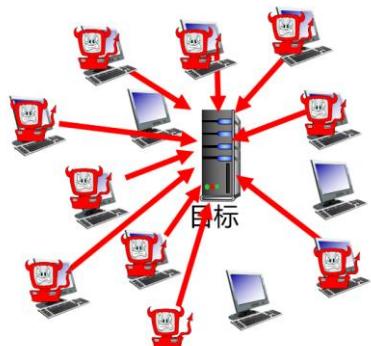
Bad guy 攻击方式：

分组“嗅探”(sniffing)：广播媒体(共享以太网、无线网)，混杂的网络接口读取/记录所有经过的分组(包括密码!) 如 wireshark，一个免费的分组嗅探器。(左图)



IP 哄骗(spoofing): 发送具有虚假源地址的分组。(右图)

拒绝服务攻击(Denial of Service, DoS): 攻击者通过伪造流量来过度利用资源, 使资源(服务器、带宽)无法用于合法流量。



防御方法:

认证: 证明你确实是你说的那个人, 如蜂窝网络通过 SIM 卡提供硬件身份。

保密性: 通过加密。完整性检查: 数字签名防止/检测篡改。访问限制: 受密码保护的 VPN。

防火墙: 接入网和核心网中专用的“中间件”: 默认关闭: 过滤接收到的数据包, 限制发送方、接收方和应用程序, 且能检测/应对 DoS 攻击。

计算机网络和互联网的历史:

1961-1972: 早期分组交换原则: 1961: Kleinrock - 排队理论显示了分组交换的有效性。1964: Baran - 军用网络中的分组交换。1967: 高级研究计划局构想的 ARPAnet。1969: 第一个 ARPAnet 节点投入运行。1972: ARPAnet 公开演示, NCP (Network Control Protocol) 第一个主机-主机协议, 第一个 e-mail 程序, ARPAnet 有了 15 个节点。

1972-1980: 网络互联, 新型和专用网络: 1970: ALOHAnet, 夏威夷上的微波网络。1974: Cerf and Kahn - 网络互联(interconnecting)的体系结构。1976: Xerox PARC 的以太网。70 年代后期: 专有架构: DECnet, SNA, XNA。1979: ARPAnet 有了 200 个节点。Cerf 和 Kahn 的网络互联原则: 极简, 自治 - 不需要内部改变来连接网络, 尽力而为(best effort)服务模型, 无状态的交换机, 去中心化的控制, 定义了今天的互联网体系结构。

1980-1990: 新的协议, 网络的激增: 1983: TCP/IP 部署。1982: smtp 电子邮件协议被定义。1983: DNS 定义: 完成域名到 IP 地址的转换。1985: ftp 协议被定义。1988: TCP 拥塞控制。新的国家级网络: CSnet, BITnet, NSFnet, Minitel, 100,000 主机连接到网络联邦。

1990, 2000s: 商业化, Web, 新的应用: 90 年代初: ARPAnet 退役。1991: NSF 放宽了对 NSFnet 用于商业目的的限制 (1995 退役)。90 年代早期: Web (hypertext [Bush 1945, Nelson 1960's]。HTML, HTTP: Berners-Lee) 1994: Mosaic, 后来的 Netscape。90 年代后期: Web 的商业化。1990 年代后期-2000 年代, 更多杀手级应用 (即时讯息, P2P 文件共享)。网络安全前沿。5000 万主机, 1 亿以上用户, 骨干链路以 Gbps 运行。

2005 至今: 规模化, SDN, 移动性, 云: 积极部署宽带家庭接入 (10-100 兆比特/秒)。2008: 软件定义网络 software-defined networking (SDN)。高速无线接入日益普及: 4G/5G、WiFi 服务提供商 (谷歌、FB、微软) 创建自己的网络。绕过商业互联网, “近距离”连接终端用户, 提供“即时”访问社交媒体、搜索、视频内容...企业在“云”(如 Amazon Web Services, 微软 Azure) 中运行其服务。智能手机的兴起: 互联网上的移动设备多于固定设备 (2017 年) 约 150 亿台设备连接互联网 (2023 年, statista.com)。

第二章 应用层

应用层协议原理：

创建一个网络应用：编写程序：能够在不同端系统运行，通过网络基础设施提供的服务，应用进程彼此通信，如 Web 服务器软件和浏览器软件通信，无需为网络核心设备编写软件，网络核心设备不运行用户应用程序，网络应用只在端系统存在，因此得以快速开发和传播。**客户端-服务器模式 (client-server)**：服务器：一直运行，有固定的 IP 地址，经常部署于数据中心，方便扩展。客户端：与服务器通信，可能间歇性连接，可以是动态的 IP 地址，不直接与其他客户端通信。例如：HTTP, IMAP, FTP。

对等结构 (peer-to-peer)：没有一直在运行的服务器，任意端系统间可以直接通信，端系统向其他端系统请求服务，并向其他端系统提供服务作为回报。自扩展性 – 新的端系统带来新的服务能力，当然也带来新的请求，端系统间歇连接并且可以改变 IP 地址。缺点是难以管理。如 P2P 文件分享 [BitTorrent]

进程通信：

进程：在主机上运行的应用程序。在同一个主机内，使用操作系统定义的进程间通信机制通信。不同主机间，通过交换报文 (Message) 来通信。使用 OS 提供的通信服务，按照应用协议交换报文，借助传输层提供的服务。

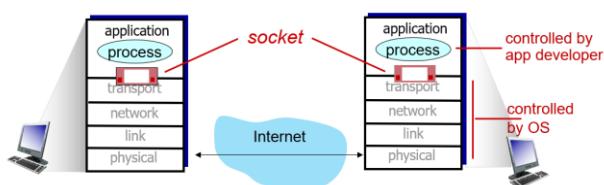
对于 cs 架构：客户端进程：发起通信，服务器进程：等待被连接。

对于 p2p 架构：P2P 架构的应用也有客户端进程和服务器进程之分。

分布式进程通信需要解决的问题：问题 1：进程标示和寻址问题。问题 2：传输层-应用层之间如何提供服务？位置：层间界面的 SAP (TCP/IP : socket) 形式：应用程序接口 API (TCP/IP : socket API)。问题 3：如何使用传输层提供的服务，实现应用进程之间的报文交换，实现应用？定义应用层协议：报文格式，解释，时序等，编制程序，使用 OS 提供的 API，调用网络基础设施提供通信服务传报文，实现应用时序等。

套接字 (socket)：进程向套接字发送报文或从套接字接收报文。套接字 <-> 门户。

发送进程将报文推到门外，发送进程依赖于门另外一侧的传输层设施，将报文交付给接受进程的套接字。涉及到两个套接字：每侧一个。



进程编址 (addressing)：为了接收报文，进程必须有标识符，主机设备有唯一的 32 位 IP 地址。Q：运行进程的主机的 IP 地址是否足以识别进程？ 不足以，一台主机可能多个进程。标识符包括与主机上进程相关的 IP 地址和端口号。端口号示例：HTTP 服务器：80。mail 服务器：25。向 gaia.cs.umass.edu web 服务器发送 HTTP 报文：IP 地址：128.119.245.12。端口号：80。

应用层协议定义了：

交换的报文类型：e.g., 请求, 应答

报文语法：报文中各个字段 (field) 及其划分方式

报文语义：字段中取值的含义，何时、如何发送和响应报文的规则

公开协议：由 RFC 文档定义，任何人都可以访问，允许互操作 e.g., HTTP, SMTP

专用协议：e.g., Skype, Zoom

应用需要什么样的传输服务？

数据完整性：有些应用（如文件传输、网络交易）要求 100% 的可靠数据传输。有些应用（如

音频) 能容忍一定比例以下的数据丢失。

时效性: 有些应用 (如网络电话、交互式游戏) 需要低延迟才能有效运行。

吞吐量: 有些应用 (如多媒体) 需要最小限度的吞吐量才能有效运转。有些应用(弹性应用)能充分利用它们获得的吞吐量。

安全: 加密、数据完整性保证。

| 应用 | 数据损失 | 吞吐量 | 时延敏感? |
|---------|------|-------------------------------------|---------------|
| 文件传输/下载 | 无损 | 弹性 | 不 |
| e-mail | 无损 | 弹性 | 不 |
| Web 文档 | 无损 | 弹性 | 不 |
| 实时音视频 | 容忍损失 | 音频: 5Kbps-1Mbps 视频: 10Kbps-5Mbps | 是, 10' s msec |
| 流式音视频 | 容忍损失 | 同上 | 是, 几秒钟 |
| 交互游戏 | 容忍损失 | Kbps+ | 是, 10' s msec |
| 即时通讯 | 无损 | 弹性 | 看情况 |

互联网传输层提供的服务:

TCP 服务: 发送方和接收方之间的可靠传输。流量控制: 发送方不会淹没接受方。拥塞控制: 当网络出现拥塞时, 能抑制发送方。面向连接: 要求在客户端进程和服务器进程之间建立连接。不能提供的服务: 时间保证、最小吞吐保证和安全。面向字节流。

UDP 服务: 发送方和接收方之间的不可靠传输。不提供的服务: 可靠、流量控制、拥塞控制、时间保证、带宽保证、建立连接。为什么要有 UDP? (作业)

UDP 存在的必要性: 能够区分不同的进程, 而 IP 服务不能。在 IP 提供的主机到主机端到端功能的基础上, 区分了主机的应用进程。无需建立连接, 省去了建立连接时间, 适合事务性的应用。不做可靠性的工作, 例如检错重发, 适合那些对实时性要求比较高而对正确性要求不高的应用, 因为为了实现可靠性 (准确性、保序等), 必须付出时间代价 (检错重发)。没有拥塞控制和流量控制, 应用能够按照设定的速度发送数据, 而在 TCP 上面的应用, 应用发送数据的速度和主机向网络发送的实际速度是不一致的, 因为有流量控制和拥塞控制。

| 应用 | 应用层协议 | 传输层协议 |
|---------|---------------------------------------|-----------|
| 文件传输/下载 | FTP [RFC 959] | TCP |
| e-mail | SMTP [RFC 5321] | TCP |
| Web 文档 | HTTP [RFC 7230, 9110] | TCP |
| 互联网电话 | SIP [RFC 3261], RTP [RFC 3550], 或专用协议 | TCP 或 UDP |
| 流式音频/视频 | HTTP [RFC 7230], DASH | TCP |
| 交互式游戏 | WOW, FPS (专用协议) | UDP 或 TCP |

(TCP 和 UDP 套接字: 没有加密, 发送到套接字的密码以明文穿越互联网! 传输层安全(TLS)。提供加密的 TCP 连接、数据完整性、终端验证。程序使用 TLS 库)

Web 和 HTTP:

Web 页由一些对象组成, 每个都可能存储在不同的 Web 服务器上
对象可以是 HTML 文件、JPEG 图像、Java 小程序、音频文件……

Web 页面包含一个基础 HTML 文件, 引用一些对象, 每个都编址为一个 URL, 例如:
www.someschool.edu/someDept/pic.gif

主机名 路径名

HTTP: hypertext transfer protocol 超文本传输协议, 是 Web 的应用层协议

客户端/服务器模型: 客户端: 请求、(使用 HTTP 协议) 接收和 "显示" 网络对象的浏览器。

服务器: Web 服务器根据请求 (使用 HTTP 协议) 发送对象

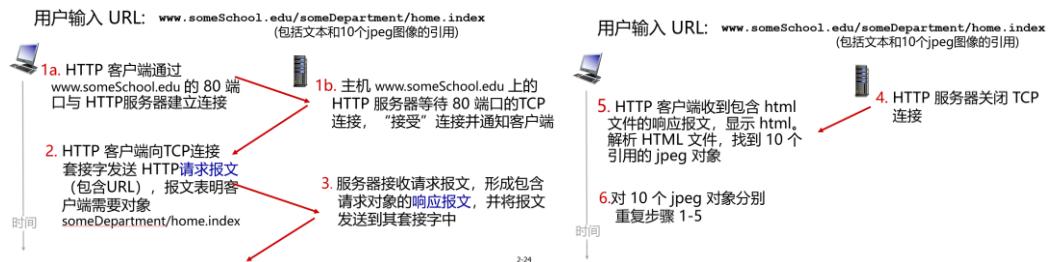
HTTP 使用 TCP: 客户端向服务器 80 端口发起 TCP 连接, 创建套接字, 服务器接受客户端的 TCP 连接, 浏览器 (HTTP 客户端) 和 Web 服务器 (HTTP 服务器) 进行 HTTP 报文 (应用层协议报文) 交换, TCP 连接关闭。HTTP 是无状态的, 服务器不维护客户端的任何信息。(维护"状态"的协议很复杂! 必须维护历史信息 (状态) 如果服务器/客户端崩溃, 它

们对“状态”的视角可能不一致，必须加以协调)



HTTP 连接的两种类型：

非持久 HTTP: TCP 连接开放，通过 TCP 连接发送最多一个对象，TCP 连接关闭（下载多个对象需要多次连接）。示例：



响应时间：RTT（定义）：一个小数据包从客户端到服务器再返回的时间

HTTP 响应时间（每个对象）：1 RTT 用来发起 TCP 连接，1 RTT 用于发送 HTTP 请求和 HTTP 响应返回的前几个字节。对象/文件传输时间

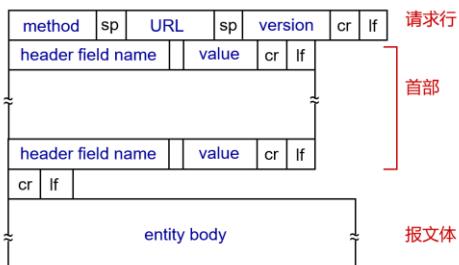
非持久 HTTP 响应时间 = 2RTT+ 文件传输时间

持久 HTTP (HTTP 1.1)：与服务器的 TCP 连接打开，客户端和服务器之间的单个 TCP 连接可以发送多个对象，TCP 连接关闭。

非持久 HTTP 的缺点：每个对象要 2 个 RTT，每个 TCP 连接都需要操作系统开销。浏览器通常打开并行 TCP 连接，以并行获取引用对象。

持久 HTTP (HTTP1.1)：服务器在发送响应后，仍保持 TCP 连接开放，同一客户端/服务器之间通过开放连接发送后续 HTTP 报文，客户端在遇到引用对象时立即发送请求，对所有引用对象的响应时间仅为一个 RTT（响应时间缩短一半）。

HTTP 请求报文：两种类型的 HTTP 报文：请求，应答。编码格式 ASCII（人类可读格式）



HTTP 请求报文的四大方法：POST 方法：网页通常包含表单输入，用户的输入在 HTTP POST 请求报文的报文体中从客户发送给服务器。GET 方法（用于向服务器发送信息）：在 HTTP GET 请求信息的 URL 字段（‘?’之后）中包含用户数据。HEAD 方法：只返回同一 URL 被 GET 方法请求时，响应报文的请求头部分。PUT 方法：向服务器上传新文件（对象），用请求报文体中的内容完全替换指定 URL 上存在的文件。

HTTP 响应报文: 状态行 (协议状态代码、短语) 如 -- HTTP/1.1 200 OK

HTTP 响应状态代码: 状态代码出现在服务器发给用户的响应报文第一行。有:

200 OK 请求成功, 请求的对象在该报文后续部分

301 Moved Permanently 请求对象已被永久转移, 新地址在该报文的 Location 域中指定

400 Bad Request 服务器无法理解请求报文

404 Not Found 服务器上没有找到请求的文档

505 HTTP Version Not Supported

维护客户端/服务器状态: cookies

回顾: HTTP GET 响应和交互是无状态的, 没有多步交换 HTTP 报文以完成网络 "事务" 的概念。客户端/服务器无需跟踪多步交换的 "状态", 所有 HTTP 请求相互独立, 客户端/服务器无需从部分完成但从未完全完成的事务中 "恢复"。

网站和客户端浏览器使用 cookies 在交易之间保持某些状态。四个组成部分:

- 1) HTTP 响应报文中的 cookie 首部行
- 2) HTTP 请求报文中的 cookie 首部行
- 3) 在客户端系统中保存、由用户浏览器管理的 cookie 文件
- 4) Web 站点的后端数据库

示例: 苏珊使用笔记本电脑上的浏览器, 首次访问特定的电子商务网站, 当初始 HTTP 请求到达网站时, 网站会创建唯一 ID (又名 "cookie") 和后台数据库中的 ID 条目。苏珊向本网站发出的后续 HTTP 请求将包含 cookie ID 值, 以便网站 "识别" 苏珊。



2-36

Cookies 记录用户的浏览行为, 可以用于: 授权、购物车、推荐、用户会话状态

挑战: 如何维护状态? 在协议端节点: 在多个交易中维护发送方/接收方的状态。在报文中: HTTP 报文中的 cookies 携带状态。

Cookies 可以被用来跟踪用户在特定网站上的行为 (第一方 cookie), 在多个网站上跟踪用户行为 (第三方 cookie), 而用户无需选择访问跟踪网站 (!)。跟踪可能对用户不可见: 不是显示广告触发 HTTP GET 到跟踪器, 可能是一个隐形链接。通过 cookie 进行第三方跟踪: 在 Firefox 和 Safari 浏览器中默认禁用, 于 2023 年在 Chrome 浏览器中逐步禁用。

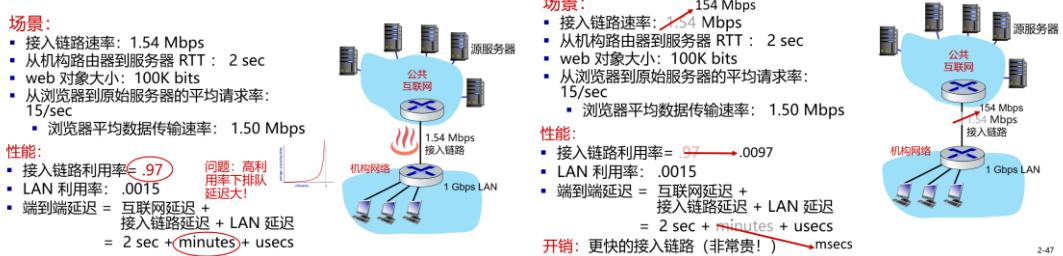
Web 缓存: 目标: 在不涉及源服务器的情况下满足客户请求。

用户将浏览器配置为指向 (本地) Web 缓存, 浏览器将所有 HTTP 请求发送到缓存, 如果对象在缓存中: 缓存将对象返回给客户, 否则, 缓存从源服务器请求对象, 缓存接收到的对象, 然后将对象返回给客户。**Web 缓存又名代理服务器**, 缓存既是客户端又是服务器, 对于浏览器请求来说是服务器, 对于源服务器来说是客户端。服务器会在响应头中告知缓存对象允许的缓存。

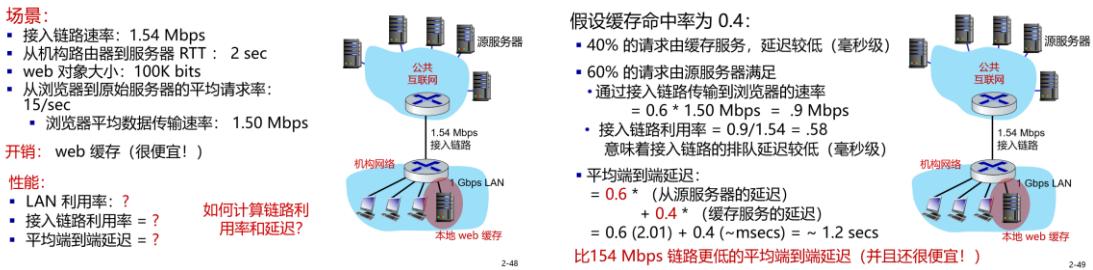
为什么要有 Web 缓存? 降低客户请求获得响应的时间, 缓存离客户更近, 减少机构接入链路的流量, 互联网上有大量缓存, 使"贫乏"的内容提供商能够更有效地提供内容。

缓存示例:

P2: 购买更快的接入链路?



P3 P4: web 缓存

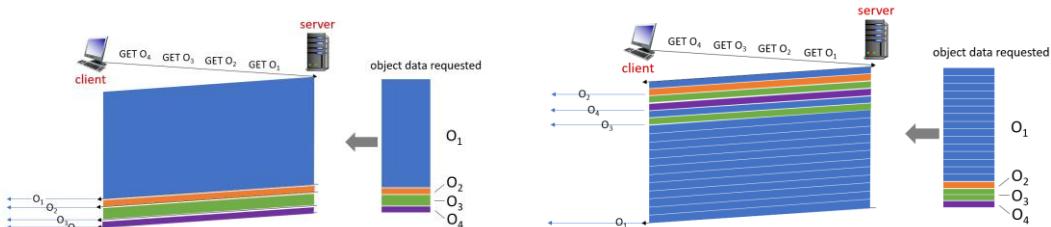


HTTP/2: 核心目标: 减少多对象 HTTP 请求的延迟

HTTP1.1: 在单个 TCP 连接上引入多个流水线 GET: 服务器按顺序 (FCFS: 先到先服务调度) 响应 GET 请求, 使用 FCFS 时, 小对象可能需要在大对象后面等待传输 (队头阻塞 head-of-line (HOL) blocking), 丢失恢复 (重新传输丢失的 TCP 报文段) 使对象传输停滞不前

HTTP/2: [RFC 7540, 2015] 提高服务器向客户端发送对象的灵活性: 方法、状态代码和大多数报头字段与 HTTP 1.1 相同, 根据客户指定的对象优先级 (不一定是 FCFS) 确定请求对象的传输顺序, 将未请求的对象推送给客户端, 将对象划分为帧, 调度帧以减少 HOL 阻塞

HTTP 1.1 (左图): 客户请求 1 个大对象 (如视频文件) 和 3 个小对象: 对象按要求的顺序交付: O2、O3、O4 在 O1 后面等待。而 HTTP/2 (右图): 将对象划分为帧, 帧间交错传输: O2、O3、O4 快速交付, O1 稍有延迟。



HTTP2 到 HTTP3: 基于单个 TCP 连接的 HTTP/2 意味着: 数据包丢失后的恢复仍会使所有对象的传输停滞, 如同 HTTP 1.1, 浏览器倾向于打开多个并行 TCP 连接, 以减少停滞, 提高总体吞吐量。在 TCP 连接上没有安全保障。HTTP/3: 通过 UDP 增加安全性、每个对象的错误恢复和拥塞控制 (更多流水线) 功能。

Email, SMTP, IMAP:

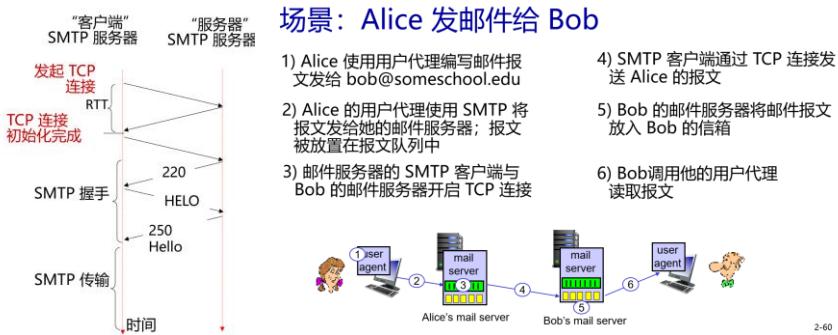
Email 三个主要部分构成: 用户代理、邮件服务器、简单邮件传输协议 SMTP

用户代理, 又名“邮件阅读者”: 撰写、编辑、阅读邮件报文, e.g., Outlook, iPhone 邮件客户端, 发件、收件报文储存在服务器上

邮件服务器: 信箱包含用户接收到的报文, 报文队列包含发送和待发邮件报文, 邮件服务器之间发送电子邮件的 SMTP 协议。客户端: 发送邮件的邮件服务器。“服务器”: 接收邮件的邮件服务器。

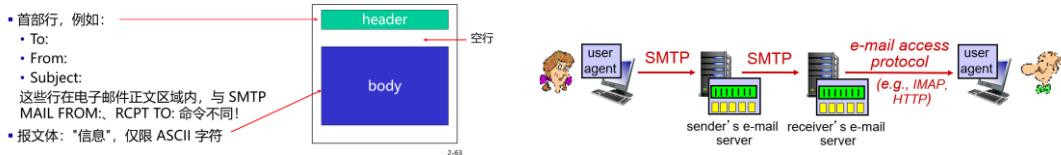
SMTP RFC [5321]: 使用 TCP 将电子邮件从客户端 (发起连接的邮件服务器) 可靠地传输到服务器, 端口为 25。直接传输: 发送方服务器 (行为类似客户端) 到接收方服务器。传输

的三个阶段：SMTP 握手 (greeting) --SMTP 传输报文--SMTP 关闭。命令/响应交互（类似 HTTP）：命令：ASCII 文本，响应：状态代码和短语。



SMTP 与 HTTP 对比：HTTP：客户拉 (pull)、SMTP：客户推 (push)。都有 ASCII 命令/响应交互，状态代码。HTTP：每个对象封装在自己的响应报文中、SMTP：多个对象包含在一个报文中。SMTP 使用持久连接，SMTP 要求报文（首部和主体）为 7-bit ASCII 编码，SMTP 服务器使用 CRLF/CRLFCRLF 来决定报文的结束。

SMTP 邮件报文格式：



SMTP：向接收方服务器发送/存储电子邮件报文。 邮件访问协议：从服务器检索

IMAP: Internet Mail Access Protocol [RFC 3501]: 互联网邮件访问协议，邮件存储在服务器上，IMAP 提供检索、删除功能，并提供服务器上存储邮件的文件夹

HTTP: gmail, Hotmail, Yahoo!Mail, etc. 在 SMTP (发送)、IMAP (或 POP) 检索电子邮件的基础上提供基于 Web 的界面。

域名系统 The Domain Name System (DNS):

互联网主机，路由器：IP 地址 (32 bit) – 用于数据报寻址，“域名”，例如 cs.umass.edu – 人类使用。 Q：如何从 IP 地址映射到域名，或者相反？

Domain Name System (DNS): 由多个域名服务器分层实现的分布式数据库

应用层协议：主机、DNS 服务器通信以解析域名（地址/域名转换）。作为应用层协议实现的互联网核心功能，体现网络“边缘”的复杂性。

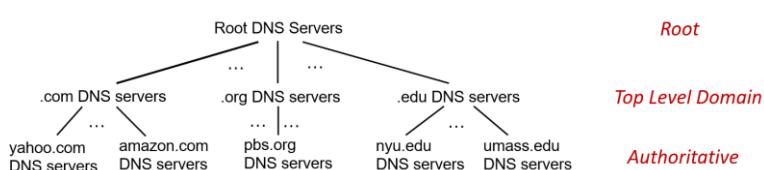
DNS 服务：主机名到 IP 地址的转换、主机别名、正名，别名、邮件服务器、负载分配、复制 Web 服务器：一个域名对应多个 IP 地址。

Q：为什么不集中管理 DNS？单点故障、流量、遥远的中央数据库、维护。

A：不可扩展！仅 Akamai DNS 服务器：每天 2.2T DNS 查询量。

关于 DNS 的思考：庞大的分布式数据库：上亿条记录，但每条都很简单。每天处理数万亿次查询：读取次数远远多于写入次数。性能很重要：几乎所有互联网事务都与 DNS 有关 - 毫秒数很重要！在组织上、物理上分散：数以百万计的不同组织负责其记录。可靠性安全性。

DNS 分布式分层数据库：



客户希望获得 www.amazon.com 的 IP 地址；第 1 次过程类似于：

客户端查询根服务器，找到 .com DNS 服务器

客户端查询 .com DNS 服务器以获得 amazon.com DNS 服务器

客户端查询 amazon.com DNS 服务器，获取 www.amazon.com 的 IP 地址

DNS：根域名服务器：

官方的、由无法解析名称的域名服务器提供的最后联系方式

极其重要的互联网功能：没有它，互联网就无法运行！DNSSEC - 提供安全性（验证、信息完整性），互联网名称与数字地址分配机构（ICANN）负责管理 DNS 根域名。全球 13 个逻辑根域名“服务器”，每个“服务器”复制多次（美国约有 200 个服务器）

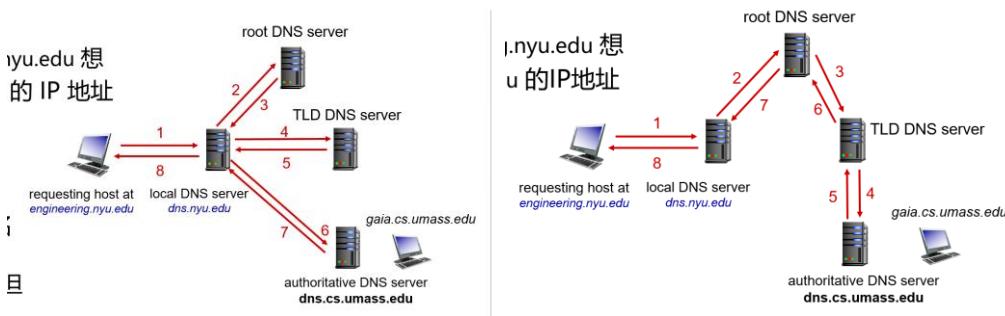
顶级域和权威服务器：Top-Level Domain (TLD) 顶级域名服务器：负责 .com、.org、.net、.edu、.aero、.jobs、.museums 和所有顶级国家域，如：.cn、.uk、.fr、.ca、.jp。Network Solutions: .com、.net TLD 的权威注册机构。Educause: .edu TLD。权威 DNS 服务器：组织自己的 DNS 服务器，为组织命名的主机提供权威的主机名到 IP 映射。可由机构或服务提供商维护。

本地 DNS 服务器：当主机进行 DNS 查询时，查询会发送到本地 DNS 服务器，本地 DNS 服务器返回应答：从其本地缓存中的最新名称到地址转换对（可能已过时），将请求转发到 DNS 层次结构中进行解析。每个 ISP 都有本地 DNS 名称服务器。本地 DNS 服务器并不严格属于层次结构。

DNS 域名解析：例如主机 engineering.nyu.edu 想要获得 gaia.cs.umass.edu 的 IP 地址。

迭代查询：被联系的服务器回复，提供要联系的服务器名称或“我不知道这名字，但可以询问这个服务器”。

递归查询：将域名解析的负担放在所联系的名称服务器上，层次结构的上层负担沉重？



缓存 DNS 信息：一旦域名服务器学习到映射，它就会缓存映射，并在响应查询时立即返回缓存的映射。缓存可缩短响应时间，缓存条目在一段时间 (TTL) 后超时（消失）。TLD 服务器通常缓存在本地域名服务器中，缓存条目可能会过时，如果被命名的主机更改了 IP 地址，在所有 TTL 到期之前，可能不是整个互联网都知道！尽力而为进行名称到地址的转换！

DNS 记录：DNS：存储资源记录 (RR) 的分布式数据库。RR 格式 (name, value, type, ttl)

type=A: name 是主机名，value 是 IP 地址

type=NS: name 是域（例如 `foo.com`），value 是该域的权威域名服务器的主机名

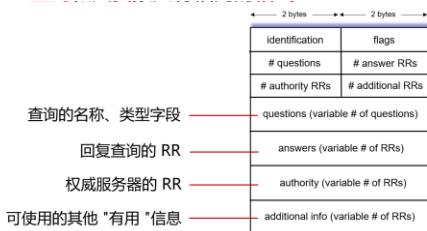
type=CNAME: name 是“正式”（真实）名称的别名，`www.ibm.com` 实际上是 `servereast.backup2.ibm.com`，value 是正式名称。

type=MX: value 是与 name 关联的 SMTP 邮件服务器名称。

DNS 协议报文：DNS 查询和回复报文有相同的格式：

报文首部: identification: 查询有 16 bit 独特的识别码，回复使用与对应查询相同的识别码。

Flags: 查询还是回复，是否希望递归，递归是否可用，回复是否具有权威性。



将你的信息输入 DNS: 例如: 新创公司“网络乌托邦”在 **DNS 注册商** (如 Network Solutions) 处注册名称 networkutopia.com。提供权威名称服务器 (主服务器和辅助服务器) 的域名和 IP 地址。注册商将 NS、A 记录插入 .com TLD 服务器:

(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)

在本地创建权威服务器, IP 地址为 212.212.212.1, 为 www.networkutopia.com 创建 A 记录, 为 networkutopia.com 创建 MX 记录。

DNS 安全:

DDoS 攻击: 用流量轰炸根服务器: 至今未成功、流量过滤、本地 DNS 服务器缓存 TLD 服务器的 IP, 允许绕过根服务器。轰炸 TLD 服务器: 潜在危险更大。

哄骗攻击: 拦截 DNS 查询, 返回虚假回复: DNS 缓存中毒。RFC 4033: DNSSEC 验证服务。

P2P (peer-to-peer) 应用:

没有一直在运行的服务器, 任意端系统间可以直接通信, 端系统向其他端系统请求服务, 并向其他端系统提供服务作为回报。自扩展性 – 新的端系统带来新的服务能力, 当然也带来新的请求, 端系统间歇连接并且可以改变 IP 地址, 难以管理。如 P2P 文件分享[BitTorrent] 文件分发: 客户端-服务器 vs P2P:

Q: 将文件 (大小为 F) 从一台服务器分发到 N 个对等服务器需要多少时间? 对等方上传/下载能力是有限的资源

文件分发时间:

客户端-服务器——

服务器传输: 必须依次发送 (上传) N 份文件副本:

发送一份的时间: F/u_s 发送 N 份文件的时间: NF/u_s

客户端: 每个客户端必须下载文件副本

d_{min} = 客户端最小下载速率 最小客户端下载时间: F/d_{min}

使用客户端-服务器方式向 N 个客户分发 F 的时间: $D_{c-s} > \max \{NF/u_s, F/d_{min}\}$

P2P——

服务器传输: 必须上传至少一份副本: 发送一份副本的时间: F/u_s

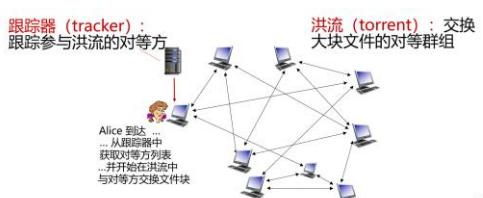
客户端: 每个客户端必须下载文件副本, 最小客户端下载时间: F/d_{min}

客户端: 总计必须下载 NF 位, 最大上传速率 (限制最大下载速率) 为 $u_s + \sum u_i$

使用 P2P 方法向 N 个客户分发 F 所需的时间: $D_{P2P} > \max \{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$

P2P 文件分发: bittorrent:

文件被分成 256Kb 的文件块, 洪流中的对等方发送/接收文件块



对等方加入洪流: 初始没有文件块, 但会逐渐从其他对等方处积累文件块。向跟踪器注册以

获取对等方列表，与对等方子集（“邻居”）连接。下载时，对等方向其他对等方上传文件块，对等方可能会更换与之交换数据块的对等方。流失：对等方可能来来去去。一旦对等方拥有整个文件，它可能（自私地）离开或（利他主义地）留在洪流中。

请求、发送文件块：请求文件块：在任何时候，不同的对等点都拥有不同的文件块子集、Alice 定期向每个对等方索取它们拥有的文件块列表、Alice 向对等方请求缺失的文件块，最稀有的优先。发送文件块：投桃报李：Alice 以最高速率向当前向其发送文件块的四个对等方发送文件块，其他同行被 Alice 阻塞（没有接收到来自 Alice 的文件块），每 10 秒重新评估前 4 名，每 30 秒：随机选择另一个对等方，开始发送文件块，“乐观地取消阻塞”该对等方，新选择的对等方可能会加入前 4 名。

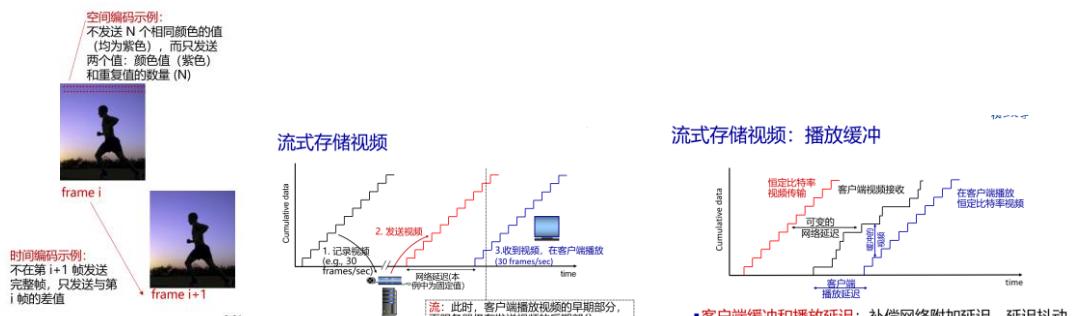
- (1) Alice “乐观地取消阻塞” Bob
- (2) Alice 成为 Bob 的前四名提供者之一；Bob 答谢 Alice
- (3) Bob 成为 Alice 的前四名提供者之一

流媒体 (Video Streaming) 和内容分发网络 (Content Distribution Networks):

流媒体和 CDN 背景：流视频流量：互联网带宽的主要消费者。Netflix, YouTube, Amazon Prime：占住宅 ISP 流量的 80% (2020)。挑战：规模 - 如何覆盖约 10 亿用户？挑战：异质性：不同用户具有不同的能力（例如，有线和移动；带宽富裕和带宽匮乏）。

解决方案：分布式应用级基础设施

多媒体视频：视频：以恒定速率显示的图像序列，例如：24 images/sec。数字图像：像素阵列，每个像素用比特表示。编码：利用图像内部和图像之间的冗余来减少图像编码所用的比特数，空间（图像内部），时间（从一幅图像到下一幅图像）。CBR: (constant bit rate)：视频编码率固定不变。VBR: (variable bit rate)：视频编码率随空间、时间编码量的变化而变化示例：MPEG 1 (CD-ROM) 1.5 Mbps, MPEG4 (常用于互联网, 64Kbps – 12 Mbps)



流式存储视频主要挑战：服务器到客户端的带宽会随着时间的推移和网络拥塞程度的变化而变化（内部、接入网、网络核心、视频服务器）。数据包丢失、拥堵造成的延迟会导致播放延迟或视频质量下降。连续播放约束：在客户端视频播放期间，播放时序必须与原始时序一致……但网络延迟是变化的（抖动），因此需要客户端缓冲区来满足连续播放约束条件。其他挑战：客户端交互性：暂停、快进、后退、跳过视频，视频数据包可能丢失、重传。

流媒体：DASH (Dynamic Adaptive Streaming over HTTP) :

服务器：将视频文件分成多个块，每个块以多种不同的速率编码，不同的速率编码存储在不同的文件中，在不同的 CDN 节点复制文件，清单文件：提供不同块的 URL。

客户端：定期估算服务器到客户端的带宽，咨询清单，每次请求一个数据块，根据当前带宽选择可持续的最大编码率，可在不同时间点从不同服务器（根据当时可用带宽）选择不同的编码率。

客户端的“智能”：客户端决定何时请求分块（以免出现缓冲区饥饿或溢出现象），请求何种编码率（当带宽更宽时，质量更高），向何处请求分块（可向离客户端“近”或可用带宽高的 URL 服务器请求分块）

流视频 = 编码 + DASH + 播放缓冲

Content distribution networks (CDNs):

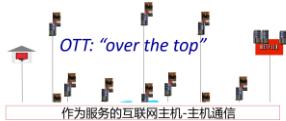
挑战：如何将内容（从数百万部视频中挑选出来），同时流式传输给数十万用户？

方案 1：单一大型“巨型服务器”：单一故障点，网络拥塞点，通往远端客户的路径较长（可能会拥塞）。简单，但该解决方案无法扩展。方案 2：在多个地理上分布式的站点存储/提供多份视频副本（CDN）。enter deep：将 CDN 服务器深入到许多接入网络中：接近用户。

Akamai：部署了 24 万台服务器，遍布 120 多个国家（2015 年）。bring home：将数量更少（10 余个）的更大集群放置在靠近接入网的 POP 中，被 Limelight 采用。

Netflix 如何工作？ Netflix：在其（全球）OpenConnect CDN 节点上存储内容副本（如《MADMEN》）。用户请求内容，服务提供商返回清单：使用清单，客户端以最高支持速率检索内容，如果网络路径拥塞，可选择不同的速率或副本。

OTT 的挑战：从“边缘”应对拥塞的互联网：将哪些内容放到哪个 CDN 节点？从哪个 CDN 节点检索内容？以何种速率检索？



第三章 传输层

传输层服务：

传输层服务和协议：在运行于不同主机的应用程序进程间提供逻辑通信。传输协议在端系统中的作用：发送方：将应用报文分割成报文段，传递给网络层。接收方：将报文段重新组合成报文，并传递给应用层。互联网应用程序可用的两种传输协议：TCP, UDP。

家庭的类比——Alice 家的 12 个孩子给 Bob 家的 12 个孩子寄信：主机 = 房子，进程 = 孩子，应用程序报文 = 装在信封里的信，传输协议 = Alice 和 Bob，负责向家里的兄弟姐妹分发信件，网络层协议 = 邮政服务。传输层：进程间的通信，依赖网络层的服务（延迟、带宽），增强网络层服务（可靠传输、数据加密）。网络层：主机间通信

传输层行为：发送方：被传递一个应用层报文，确定报文段首部字段值，创建报文段，将报文段传递给 IP。接收方：从 IP 接收报文段，检查首部值，提取应用层报文，通过套接字将报文解复用到应用程序。**两个主要的互联网传输协议：**TCP: Transmission Control Protocol 传输控制协议：可靠的、保序的传输，拥塞控制，流量控制，建立连接。UDP: User Datagram Protocol 用户数据报协议：不可靠、不保序的传输，不加修饰地扩展“尽力而为”的 IP。无法提供的服务：延迟保证，带宽保证。

多路复用和解复用：

发送方复用：处理来自多个套接字的数据，添加传输层首部（随后用于解复用）

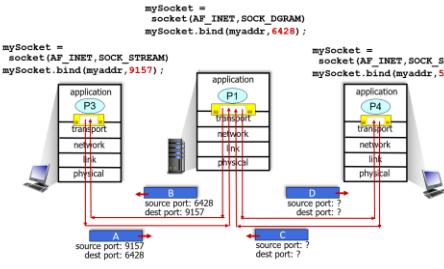
接收方解复用：使用首部信息将接收到的报文段传送到正确的套接字（高速选择出口离开）

解复用工作原理：主机接收 IP 数据报，每个数据报都有源 IP 地址和目的 IP 地址，每个数据报携带一个传输层报文段，每个报文段都有源端口号和目的端口号，主机使用 IP 地址和端口号将报文段导向合适的套接字。

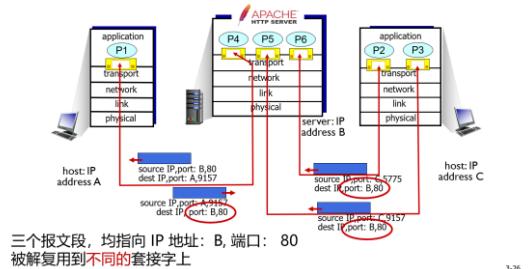
无连接解复用：回顾：创建套接字时，必须指定主机-本地端口号：DatagramSocket mySocket1 = new DatagramSocket(12534)；在创建要发送到 UDP 套接字的数据报时，必须指定：目的 IP 地址和目的端口号。当接收主机收到 UDP 报文段时：检查报文段中的目的端口号，将 UDP 报文段导向对应该端口号的套接字。具有相同目的端口号，但源 IP 地址和/或源端口号不同的 IP/UDP 数据报将被导向接收主机的相同套接字。

面向连接的解复用：TCP 套接字由四元组标识：源 IP 地址，源端口号，目的 IP 地址，目的端口号。解复用：接收器使用全部 4 个值（四元组）将报文段导向合适的套接字。服务器可同时支持多个 TCP 套接字：每个套接字有自己的四元组标识，每个套接字与不同的连接客户端相关联。

无连接解复用：示例



面向连接的解复用：示例



Summary: 复用、解复用：基于报文段、数据报首部字段值。UDP：仅使用目的端口号进行解复用，TCP：使用四元组（源 IP 地址、目的 IP 地址、源端口号、目的端口号）进行解复用。复用/解复用在所有层发生。

无连接传输：UDP：

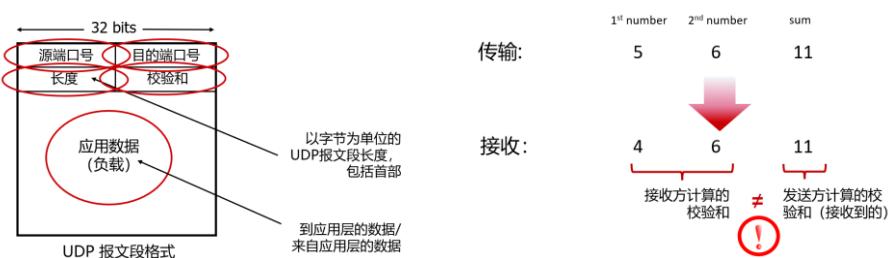
UDP：User Datagram Protocol：“不加修饰”的“基本”互联网传输协议，“尽力而为”的服务，UDP 报文段可能：丢失，无序传输到应用程序。无连接：UDP 发送方和接收方之间不握手，每个 UDP 报文段独立于其他报文段处理。

为什么要有 UDP？ 无需建立连接（连接可能会增加 RTT 时延），简单：发送方和接收方均无连接状态，首部小，无拥塞控制，UDP 可随心所欲地快速传输！面对拥塞也能正常工作。UDP 一般用于：流媒体应用程序（可容忍丢失，对速率敏感），DNS，SNMP，HTTP/3。

如果需要通过 UDP 进行可靠传输（如 HTTP/3）：在应用层增加所需的可靠性，在应用层增加拥塞控制。

UDP 的传输层行为：

发送方：被传递一条应用层消息，确定 UDP 报文段首部字段值，创建 UDP 报文段，向 IP 传递报文段。接收方：从 IP 接收报文段，检查 UDP 首部校验和的值，提取应用层信息，通过套接字将消息解复用至应用程序。



UDP 校验和：目标是检测传输的报文段中的错误（即比特翻转）

发送方：将 UDP 报文段内容（包括 UDP 首部字段和 IP 地址）视为 16 位整数序列。校验和：内容的加法和（1 的补运算），发送方将校验和的值放入 UDP 校验和字段。

接收方：计算接收到的报文段的校验和，检查计算出的校验和是否等于校验和字段的值：不相等 - 检测到错误。相等 - 没有检测到错误。但还是可能有错误：将两个 16 位整数相加：若两个翻转，如 01 变成 10，尽管数字发生了变化（位翻转），但校验和没有变化！

示例：将两个 16 位整数相加

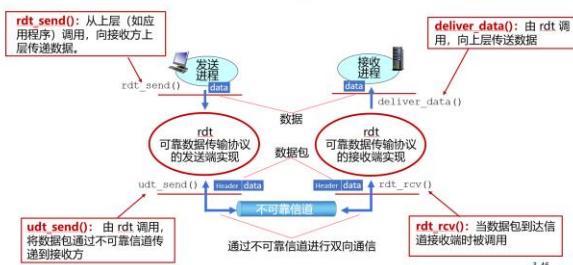
| | |
|-----|---------------------------------|
| 回卷 | 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 |
| 和 | 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 |
| 校验和 | 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 |

注意：进行数字加法运算时，需要将最高位的进位加到运算结果中（回卷）。

目标端：校验内容+校验和=1111111111111111 通过校验，否则没有通过。

Summary: UDP “无装饰”协议：报文段可能丢失、传输可能乱序，尽力而为的服务：“发送并期待最好的结果”。UDP 有其优点：无需设置/握手（不产生 RTT），在网络服务受到影响时仍能发挥作用，有助于提高可靠性（校验和），在 UDP 的基础上在应用层构建附加功能
可靠数据传输的原理：

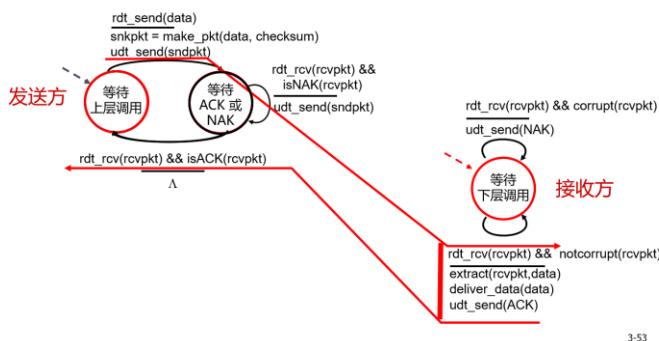
Reliable data transfer protocol(rdt): 接口



3-46

Rdt1.0：在可靠信道上的可靠传输：底层信道完全可靠：没有比特错误，没有分组丢失。发送方和接收方有独立的 FSM：发送方向底层信道发送数据，接收方从底层通道读取数据

Rdt2.0：信道有比特错误：底层信道可能会翻转数据包中的比特，通过校验和（如互联网校验和）来检测比特错误。问题：如何从错误中恢复？确认 acknowledgments(ACK)：接收方明确告知发送方，已收到正确的分组。负确认 negative acknowledgements(NAK)：接收方明确告知发送方，数据包有错误。发送方收到 NAK 后重传分组。停等：发送方发送一个分组，然后等待接收方回应。注意：接收方的“状态”（接收方是否正确接收了我的信息？）并不为发送方所知，除非接收方以某种方式向发送方传递信息。这就是我们需要协议的原因！

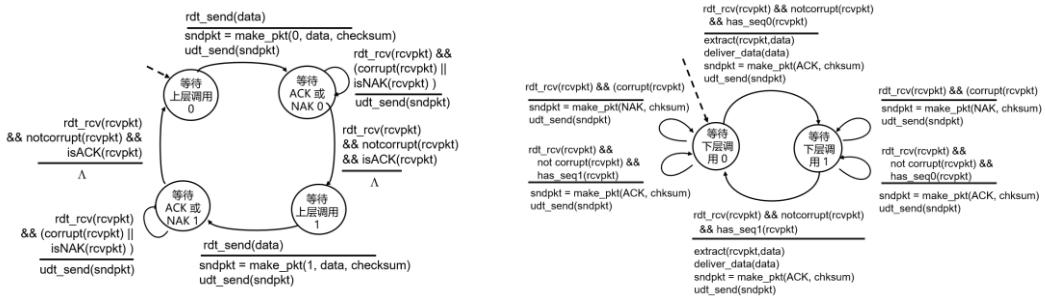


3-53

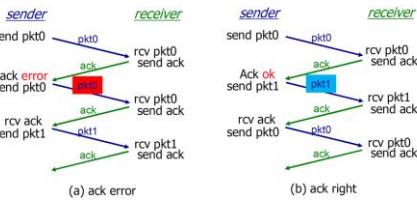
Rdt2.0 的致命缺陷：如果 ACK/NAK 被破坏会发生什么？发送方不知道接收方发生了什么！也不能简单重传：可能重复。处理重复数据：如果 ACK/NAK 出错，发送方重传当前分组，发送方在每个分组上添加序列号，接收方丢弃（不向上层传递）重复的分组。

Rdt2.1：

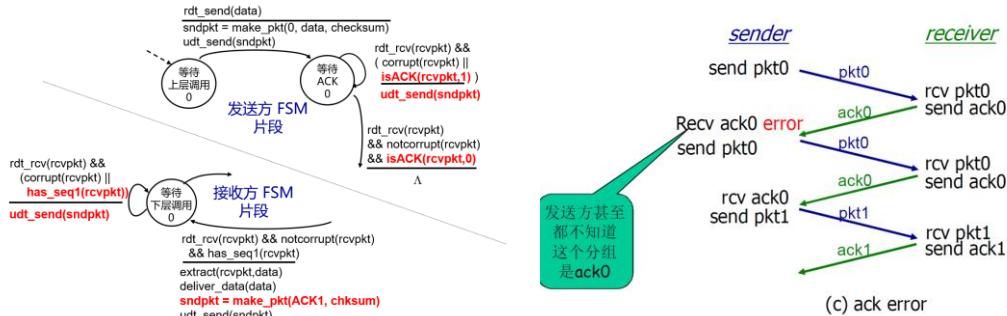
发送方，处理乱码 ACK/NAK（左图）。将序列号添加到分组。两个序列号（0, 1）就够了。必须检查收到的 ACK/NAK 是否损坏。两倍的状态。状态必须“记住”“预期”分组的序列号是 0 还是 1。接收方，处理乱码 ACK/NAK（右图）。必须检查接收到的分组是否重复。状态指示 0 或 1 是预期的分组序列号。注意：接收方无法知道其最后一次 ACK/NAK 是否被发送方正确地接收



运行：发送方不对收到的 ACK/NAK 给确认，没有所谓确认的确认。接收方发送 ACK，如果后面接收方收到的是：老分组 P0？则发送方没有收到正确的 ACK。下一个分组 P1？则发送方收到了正确的 ACK

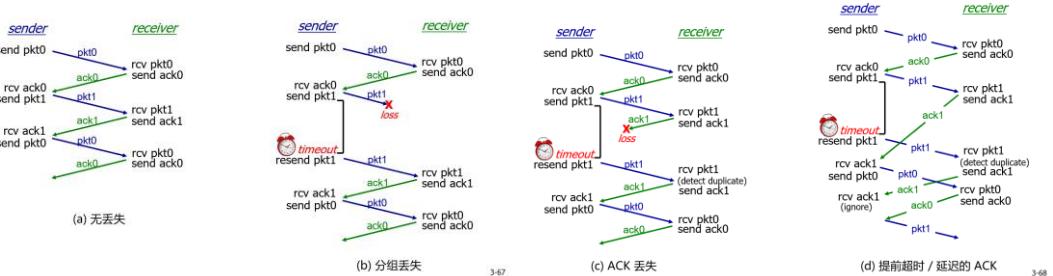


rdt2.2: 无 NAK 协议：功能与 rdt2.1 相同，仅使用 ACK。接收方不发送 NAK，而是对收到的最后一个正确分组发送 ACK，接收方必须明确包含被 ACK 分组的序列号，发送方的重复 ACK 会导致与 NAK 相同的操作：重新发送当前分组。



rdt3.0: 有比特错误和分组丢失的信道:新的信道假设：底层信道也会丢失分组（数据、ACK）校验和、序列号、ACK、重传会有帮助.....但还不够。

方法：超时重传。发送方为 ACK 等待“合理”的时间，如果在这段时间内没有收到 ACK，则重传，如果分组（或 ACK）只是延迟（而不是丢失）：重传将是重复的，但序列号已经处理了这个问题！接收方必须指定被 ACK 的分组的序列号。行为如下：



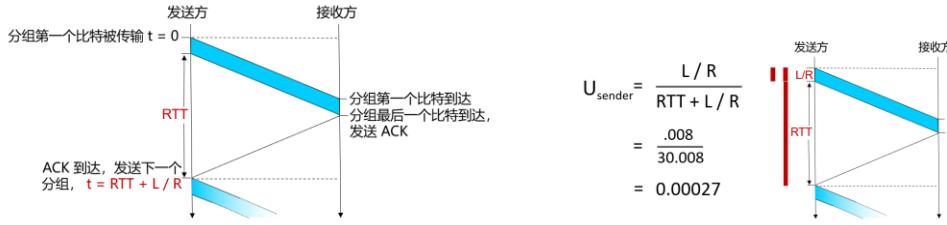
Rdt3.0 性能：U sender: 利用率 – 发送方忙于发送的时间比例。

例如： 1 Gbps 链路, 15 ms 延迟, 8000 bit 分组：

分组传输到信道的时间：

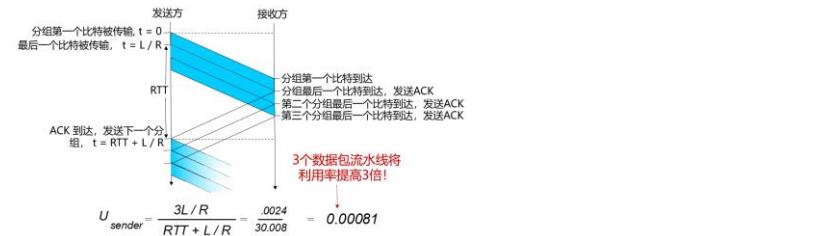
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

Rdt3.0 的停等操作：



rdt 3.0 协议性能太差！协议限制了底层基础设施（通道）的性能

rdt3.0 流水化协议操作：流水线：发送方允许多个“飞行中”、尚未确认的数据包，必须增加序列号的范围，发送方和/或接收方缓冲。流水线提高利用率：



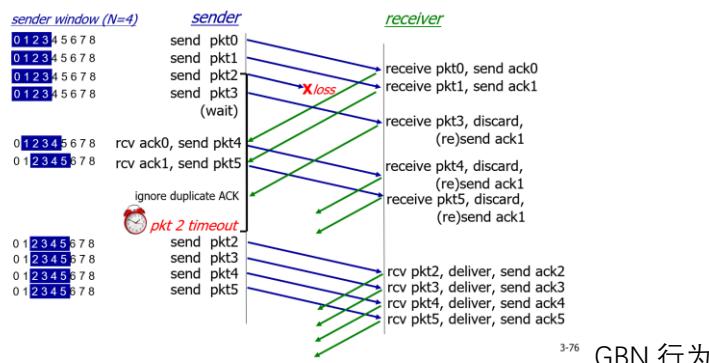
Go-Back-N (GBN) 策略：

发送方：“窗口”最多为 N 个连续发送但未被 ACK 的分组，分组首部中的序列号为 k-bit 累积 ACK: ACK(n): 对序列号为 n 及之前的所有分组 ACK, 收到 ACK(n)时：向前移动窗口，从 n+1 开始，为尚未被 ACK 的分组中最老的计时。

timeout(n): 重传窗口中序列号为 n 及之前的所有分组。



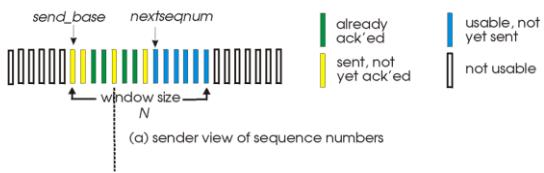
接收方：ACK-only：总是为目前已经收到的、有序且序列号最高的分组发送 ACK，可能会产生重复的 ACK。只需记住 rcv_base，在收到乱序分组时可以丢弃（不缓冲）或缓冲：由实现决定，对有序且序列号最高的分组重新 ACK。



选择重传 (Selective Repeat, SR) 策略：

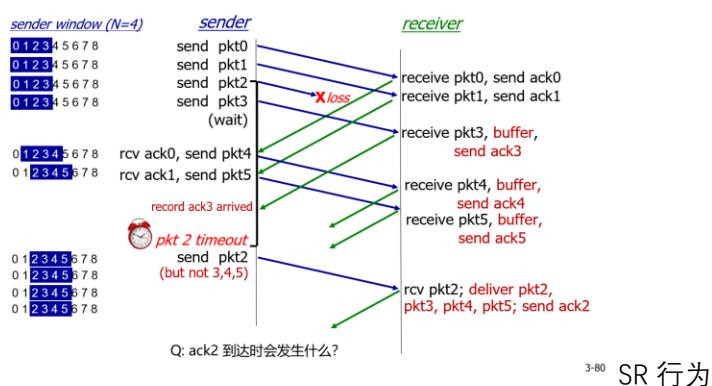
流水化：允许多个分组未确认（“飞行中”），接收方单独 ACK 所有被正确接收的包，根据需要缓冲数据包，以便按顺序传递到上层。发送方：为每个未被 ACK 的分组（概念上）维护一个计时器。超时：重发与超时相关的单个未被 ACK 的分组，（概念上）维护 N 个连续序

列号的“窗口”，限制流水线传输的“飞行中”分组在此窗口内。



发送方：从上层收到数据：如果下一个可用序列号在窗口中，发送分组。timeout(n)：重传分组 n，重启计时器。ACK(n) 在 [sendbase, sendbase+N-1] 中：将分组 n 标记为已收到，如果 n 是最小的未被 ACK 的分组，将窗口 base 向前移动到下一个未被 ACK 的序列号。

接收方：收到在 [rcvbase, rcvbase+N-1] 中的分组 n，发送 ACK(n)，乱序：缓存，有序：（和其他被缓存的有序分组一起）传递到上层，将窗口 base 向前移动到下一个尚未收到的分组。收到在 [rcvbase-N, rcvbase-1] 中的分组 n，发送 ACK(n)，否则忽略。



3-80 SR 行为

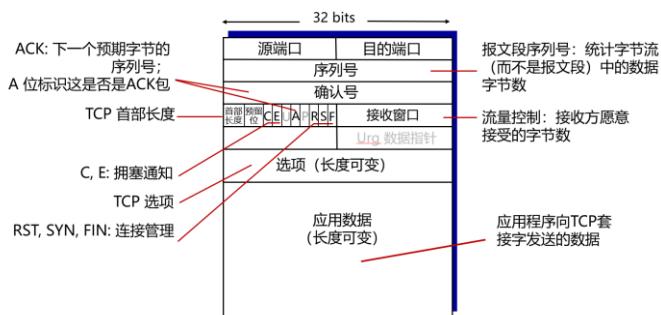
GBN 和 SR 对比：适用范围：出错率低：比较适合 GBN，出错非常罕见，没有必要用复杂的 SR。链路容量大（延迟大、带宽大）：比较适合 SR 而不是 GBN，出错代价太大。

| | GBN | SR |
|----|---------------------|----------------------|
| 优点 | 简单，所需资源少（接收方一个缓存单元） | 出错时，重传一个代价小 |
| 缺点 | 一旦出错，回退N步代价大 | 复杂，所需要资源多（接收方多个缓存单元） |

面向连接的传输：TCP：报文段结构、可靠数据传输、流量控制、连接管理：

TCP 概述：点对点：一个发送方，一个接收方。可靠、有序的字节传输：无“消息边界”，全双工数据：同一连接中的双向数据流，MSS：最大报文段大小。累积 ACK：流水线：TCP 拥塞和流量控制设置窗口大小。面向连接：握手（交换控制信息）在交换数据前初始化发送方和接收方状态。流量受控：发送方不会淹没接收方

TCP 报文段结构：



3-86

序列号：报文段数据中第一个字节的字节流“编号”

确认号：预期对方发出的下一个字节的序列号，累积 ACK

Q: 接收方如何处理乱序报文段 A: TCP 规范没有说明, 由实现者决定

TCP 往返时间, 超时: Q: 如何设置 TCP 超时值? 要比 RTT 长, 但 RTT 会变化! 太短: 过早超时, 不必要的重传, 太长: 对报文段丢失反应慢。Q: 如何估算 RTT? SampleRTT: 测量从报文段传输到收到 ACK 的时间, 忽略重传。SampleRTT 会变化, 希望估算的 RTT 更“平滑”一些, 取最近几次测量的平均值, 而不仅仅是当前的 SampleRTT。 α 经典值 0.125

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- 超时时间: EstimatedRTT 加上“安全边际”
 - EstimatedRTT 变化大: 需要更大的安全边际

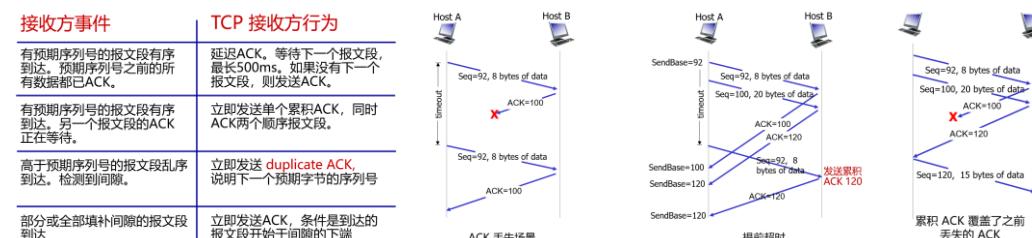
$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

- DevRTT: SampleRTT 与 EstimatedRTT 偏差的 EWMA:

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(通常 $\beta = 0.25$)

事件: 从应用程序接收到数据, 用序列号创建报文段, 序列号是报文中第一个数据字节在字节流中的编号, 如果计时器现在没有运行, 启动计时器, 认为计时器是最老的未被 ACK 的报文段的, 过期时间间隔: TimeOutInterval。**事件: 超时**, 重传导致超时的报文段, 重启计时器。**事件: 收到 ACK**, 如果 ACK 确认了之前未被 ACK 的分段: 更新已知的 ACK, 如果仍有未被 ACK 的报文段, 则启动计时器。 (右图三个重传场景)



TCP 快速重传: 如果发送方收到 3 个针对相同数据的额外 ACK (“三重重复 ACK”), 则重传序列号最小的未 ACK 报文段, 很有可能未被 ACK 的报文段已丢失, 所以不要等到超时。

(收到 3 个重复的 ACK 表明在缺失报文段后收到了 3 个报文段 - 很可能丢失了报文段。因此要重新传输!)

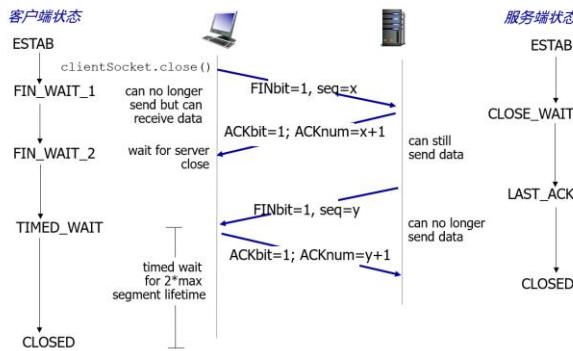
TCP 流量控制: Q: 如果网络层传输数据的速度快于应用层从套接字缓冲区获取数据的速度, 会发生什么情况? 要流量控制: 接收方控制发送方, 确保发送方不会因传输过多、过快而溢出接收方的缓冲区。方法: TCP 接收方在 TCP 首部的 rwnd 字段“公布”可用缓冲空间, 通过套接字选项设置 RcvBuffer (典型默认值为 4096 字节), 许多操作系统自动调整 RcvBuffer, 发送方根据接受的 rwnd 限制未确认 (飞行中) 的数据数量, 保证接收方缓冲区不会溢出。

TCP 连接管理: 在交换数据之前, 发送方和接收方“握手”: 同意建立连接 (双方都知道对方愿意建立连接), 商定连接参数 (如起始序列号)。

Q: 两次握手在网络中总能起作用吗? 可变延迟、因消息丢失而重新传输消息 (例如 req_conn(x))、消息乱序、无法“看到”对方状态。最终, **TCP 三次握手**:



TCP 关闭连接：



拥塞控制原理：

拥塞：非正式定义：发送数据的源太多、太快，数据太多以致网络无法处理。表现：长时间延迟（路由器缓冲区排队）、分组丢失（路由器缓冲区溢出）。不同于流量控制！

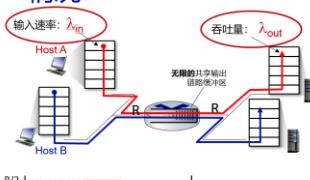
拥塞控制：发送方太多，发送速度太快

流量控制：一个发送方向一个接收方发送速度过快

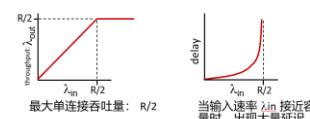
拥塞的原因/代价：情况1

最简单的情况：

- 一个路由器，无限缓冲区
- 输入、输出链路容量：R
- 两条流
- 无需重传



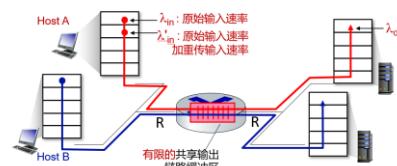
Q: 当输入速率 λ_{in} 达到 $R/2$ 时会发生什么？



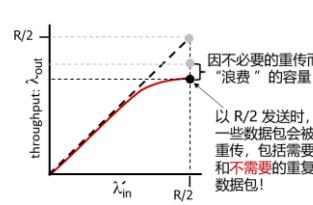
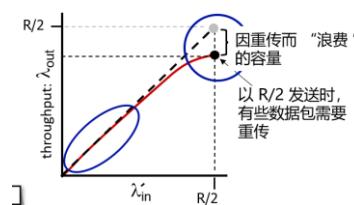
拥塞的原因/代价：情况2

- 一个路由器，有限缓冲区

- 发送方重传丢失、超时的分组
 - 应用层输入 = 应用层输出: $\lambda_{in} = \lambda_{out}$
 - 传输层输入包括重传: $\lambda'_{in} \geq \lambda_{in}$



情况 2：理想化：了解所有信息：发送方仅在路由器缓冲区可用时发送消息。理想化：了解丢失信息，由于缓冲区已满，分组可能会丢失（在路由器上被丢弃），发送方知道数据包何时被丢弃：只有当分组已知丢失时才重新发送。现实情况：不必要的重传：数据包可能丢失，或由于缓冲区已满而在路由器上被丢弃 - 需要重传，但发送方时间可能会提前超时，发送两份副本，两份副本都会送达。拥塞的“成本”：在接收器吞吐量一定的情况下，工作量更大（重传），不需要的重传：链路传递分组的多个副本，降低最大可实现吞吐量。

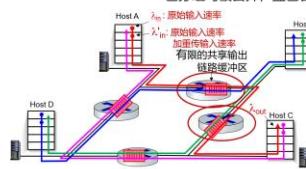


拥塞的原因/代价：情况3

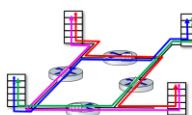
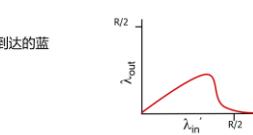
- 四个发送方
- 多跳路径
- 超时/重传

Q: 当 λ_{in} 和 λ'_{in} 增加时会发生什么？

A: 当红色的 λ_{in} 增加时，上级队列中所有到达的蓝色分组均被丢弃，蓝色吞吐量 $\rightarrow 0$



拥塞的原因/代价：情况3

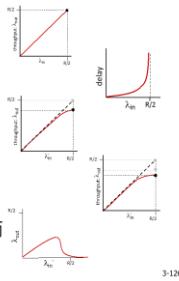


拥塞的另一个“代价”：

- 当分组丢失时，用于该分组的上游传输能力和缓冲都会被浪费！

拥塞的原因/代价：洞察

- 吞吐量永远不会超过容量
- 延迟随着接近容量而增加
- 丢失/重传会降低有效吞吐量
- 不必要的重复会进一步降低有效吞吐量
- 上游传输能力/缓冲因下游数据包丢失而浪费



3-126

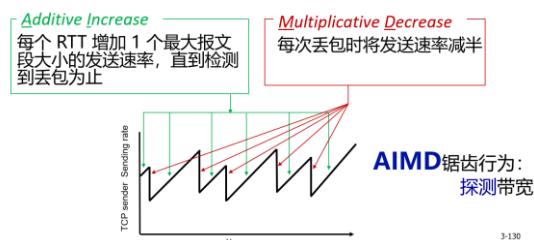
拥塞控制的方法：

端到端拥塞控制：网络无明确反馈、根据观察到的丢包、延迟推断拥塞情况，TCP 所采用。

网络辅助拥塞控制：路由器向通过拥塞路由器的流的发送/接收主机提供直接反馈，可指示拥塞程度或明确设置发送速率，TCP ECN, ATM, DECbit 协议。

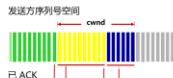
TCP 拥塞控制：

拥塞控制 AIMD：方法：发送方可以提高发送速率，直到发生丢包（拥塞），然后在丢包事件发生时降低发送速率。



3-130

TCP 拥塞控制：细节



TCP 发送行为：
大致：发送 $cwnd$ 字节，在接下来的 RTT 时间内等待 ACK，然后发送更多字节
 $TCP \text{ rate} \approx \frac{cwnd}{RTT} \text{ bytes/sec}$

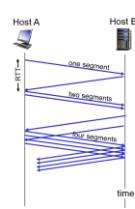
- TCP 发送方传输： $\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$
- 根据观察到的网络拥塞情况动态调整 $cwnd$ (实现 TCP 拥塞控制)

3-132

乘性减细节：当通过三重重复 ACK 检测到丢包时，发送速率减半 (TCP Reno)，当通过超时检测到丢包时，发送速率减至 1 MSS (最大报文段大小) (TCP Tahoe)。为什么选择 AIMD？AIMD 是一种分布式异步算法，已被证明能够：优化全网拥塞流量！具有理想的稳定性

TCP 慢启动

- 连接开始后，速率以指数方式增加，直到出现第一次丢包事件：
- 最初 $cwnd = 1 MSS$
- 每个 RTT 将 $cwnd$ 加倍
- 每收到一个 ACK， $cwnd$ 递增一次



3-133

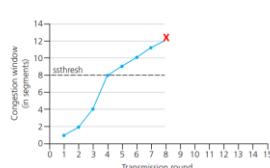
TCP：从慢启动到拥塞避免

Q: 指数增长应该何时切换为线性增长？

A: 当 $cwnd$ 在达到超时前其值的 1/2 时。

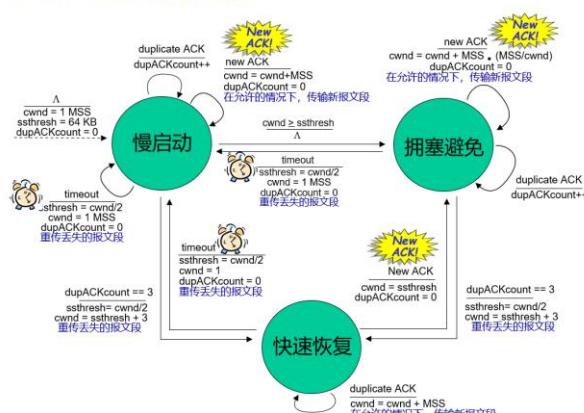
实现：

- 变量 $ssthresh$
- 在丢包事件发生时， $ssthresh$ 被设为 $cwnd$ (丢包发生前值) 的 1/2



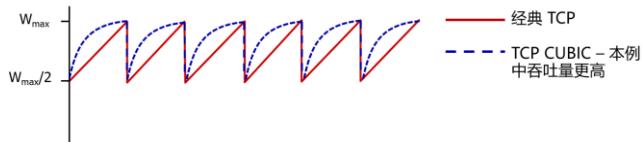
3-132

总结：TCP 拥塞控制



TCP CUBIC: 有没有比 AIMD 更好的“探测”可用带宽的方法? (Linux 默认用 cubic)

直觉: Wmax: 检测到拥塞丢包时的发送速率瓶颈链路的拥塞状态可能(?)没有太大变化



K: TCP 窗口大小将达到 W_{max} 的时间点。K 本身可调

W: 依据当前时间与 K 之间距离的三次函数来增加 W: 离 K 越远, W 的增幅越大, 离 K 越近, W 的增幅越小(越谨慎)。

TCP 和拥塞的“瓶颈链路”: TCP (经典、CUBIC) 提高 TCP 发送速率, 直到某个路由器的输出端(即瓶颈链路)出现丢包为止。了解拥塞情况, 关注瓶颈链路非常有用。



基于延迟的 TCP 拥塞控制: 保持发送方到接收方管道“足够满, 但不能更满”: 让瓶颈链路忙于传输, 但避免高延迟/缓冲。基于延迟的方法: RTTmin: 观察到的最小 RTT (无拥塞路径), 拥塞窗口为 cwnd 的无拥塞吞吐量是 $cwnd/RTT_{min}$ 。如果测得的吞吐量“非常接近”无拥塞吞吐量, 则线性增加 cwnd /* 因为路径不拥塞 */; 否则, 如果测得的吞吐量“远低于”无拥塞吞吐量, 则线性减少 cwnd /* 因为路径拥塞 */



Explicit congestion notification (ECN): TCP 部署通常实现网络辅助拥塞控制: 由网络路由器标记 IP 首部 (ToS 字段) 中的两个比特, 以指示拥塞情况。由网络运营商选择确定标记的策略: 向目的地传送拥塞指示、目的地设置 ACK 报文段上的 ECE 位, 通知发送方拥塞情况、涉及 IP (IP 首部 ECN 位标记) 和 TCP (TCP 首部 C,E 位标记)

TCP 公平性: 公平目标: 如果 K 个 TCP 会话共享同一带宽为 R 的瓶颈链路, 则每个会话的平均速率应为 R/K 。示例: 两个相互竞争的 TCP 会话: 加性增: 斜率为 1, 吞吐量增加。乘性减: 按比例降低吞吐量。

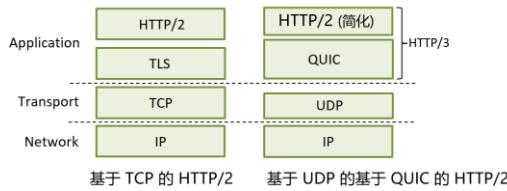


公平性: 所有网络应用都必须公平吗? 公平性和 UDP: 多媒体应用程序通常不使用 TCP。不希望被拥塞控制限制速率。使用 UDP: 以恒定速率发送音频/视频, 容忍数据包丢失, 没有“互联网警察”来监督拥塞控制的使用。公平、并行 TCP 连接: 应用程序可在两台主机之间打开多个并行连接, 网络浏览器可以做到这一点, 例如, 速率为 R 的链接有 9 个现有连接: 新应用程序请求 1 个 TCP, 获得速率 $R/10$, 新应用程序请求 11 个 TCP, 得到 $R/2$ 。

传输层功能演变:

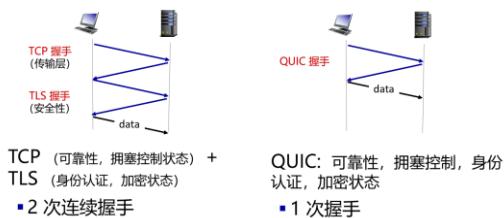
QUIC: Quick UDP Internet Connections:

应用层协议，以 UDP 为基础。提高 HTTP 性能，部署在许多谷歌服务器和应用程序（Chrome、手机 YouTube 应用）

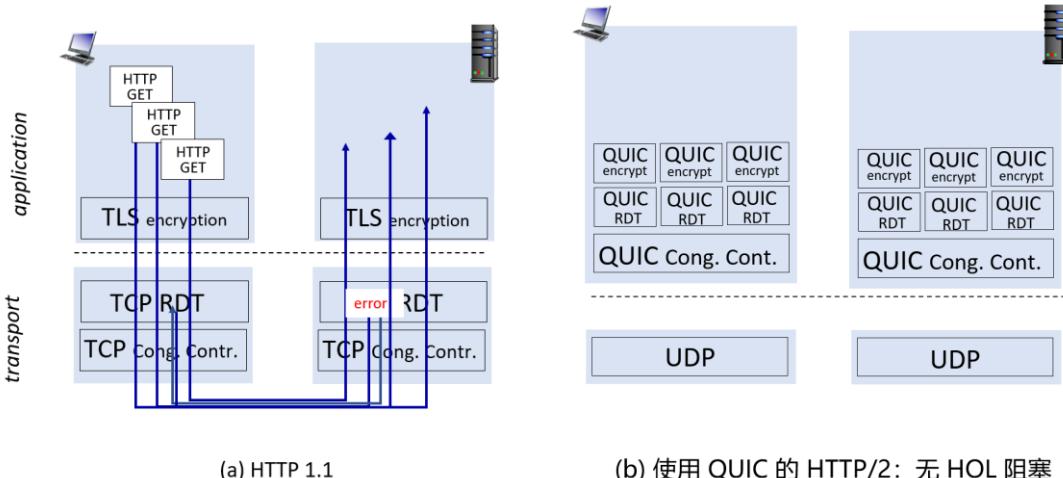


在单个 QUIC 连接上多路复用多个应用级“流”：独立可靠的数据传输、安全性，共同拥塞控制。

QUIC 建立连接：



QUIC 流：并行，无 HOL 阻塞



补充材料：TCP Tahoe---

Slow start: Start with $cwnd = 1$ (slow start) Enter CA when $cwnd \geq ssthresh$

On each successful ACK increment $cwnd$: $cwnd = cwnd + 1$

Exponential growth of $cwnd$, each RTT: $cwnd = 2 \times cwnd$

Congestion Avoidance: Starts when $cwnd > ssthresh$. On each successful ACK: $cwnd = cwnd + 1/cwnd$. Linear growth of $cwnd$: each RTT: $cwnd = cwnd + 1$

Packet loss: Assumption: loss indicates congestion

Packet loss detected by Retransmission TimeOuts (RTO timer), Duplicate ACKs (at least 3)

Fast retransmit: Wait for a timeout is quite long, Immediately retransmits after 3 dupACKs without waiting for timeout. Adjusts $ssthresh$: $flightsize = \min(awnd, cwnd)$, $ssthresh = \max(flightsize/2, 2)$. Enter Slow Start ($cwnd = 1$)

TCP Reno:

Fast retransmission:

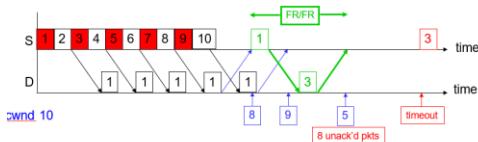
- ① 3 duplicate ACKs: $cwnd = ssthresh = 0.5cwnd$
- ② RTO: $ssthresh = 0.5cwnd$; $cwnd = 1$;

- ③ 3 duplicate ACKs: cwnd=ssthresh=0.5cwnd
- ④ Between ③ and ④: cwnd increases linearly (congestion avoidance).

Fast recovery

- Motivation: prevent 'pipe' from emptying after fast retransmit
- Idea: each dupACK represents a packet having left the pipe (successfully received)
- Enter FR/FR after 3 dupACKs
 - Set $ssthresh \leftarrow \max(\text{flightsize}/2, 2)$
 - Retransmit lost packet
 - Set $cwnd \leftarrow ssthresh + ndup$ (window inflation)
 - Wait till $W=\min(\text{awnd}, \text{cwnd})$ is large enough; transmit new packet(s)
 - On non-dup ACK (1 RTT later), set $cwnd \leftarrow ssthresh$ (window deflation)
- Enter CA

NewReno: Motivation



- On 3 dupACKs, receiver has packets 2, 4, 6, 8, $cwnd=8$, retransmits pkt 1, enter FR/FR
- Next dupACK increment $cwnd$ to 9
- After a RTT, ACK arrives for pkts 1 & 2, exit FR/FR, $cwnd=5$, 8 unack'd pkts
- No more ACK, sender must wait for timeout

图3-29

NewReno: Motivation: multiple losses within a window. Partial ACK acknowledges some but not all packets outstanding at start of FR. Partial ACK takes Reno out of FR, deflates window. Sender may have to wait for timeout before proceeding

Idea: partial ACK indicates lost packets. Stays in FR/FR and retransmits immediately. Retransmits 1 lost packet per RTT until all lost packets from that window are retransmitted. Eliminates timeout.

TCP Vegas:

- Try to achieve equilibrium during congestion avoidance
- Examine TX rate
 - Too fast, cause congestion, RTT increases: reduce window size
 - Too slow, no congestion, RTT does not increase: increase window size
 - Otherwise: no change
- Window size adjustment
 - Max: W/RTT_{min}
 - Real: W/RTT
 - Difference: $diff = W(1/RTT_{min} - 1/RTT)$
 - $diff < a$: $W++$ (this means RTT is close to RTT_{min} , thus no congestion)
 - $diff > b$: $W-$ (this means RTT is increasing, there could be congestion)
 - $a < diff < b$: no change
- $W=1000$, $RTT_{min}=20$, we expect a rate of 50, $[a,b]=[8,12]$:
 - Case 1: If $RTT=30$, the real rate is 33.3, diff is $16.7 > b$, $W-$. This implies congestion because the round trip delay is increasing
 - Case 2: If $RTT=22$, the real rate is 45.5, diff is 4.5, $W++$. This means we can easily reach the expected rate with no congestion. Therefore, it's possible to further increase the rate.
 - Case 3: If $RTT=25$, the real rate is 40, diff is 10, no change to W . This indicates the current rate is moderate.

第四章 网络层-数据平面

网络层：概览：数据平面、控制平面：

网络层服务和协议：在发送方和接收方之间传输报文段，发送方：封装报文段到数据报中，递交给链路层。接收方：递交报文段到传输层。网络层协议存在于每个主机和路由器。路由器：检查每一个经过它的 IP 数据报的头部，将数据报从输入端口移交到输出端口，从而实现将数据包在端到端的路径上传输的目的。

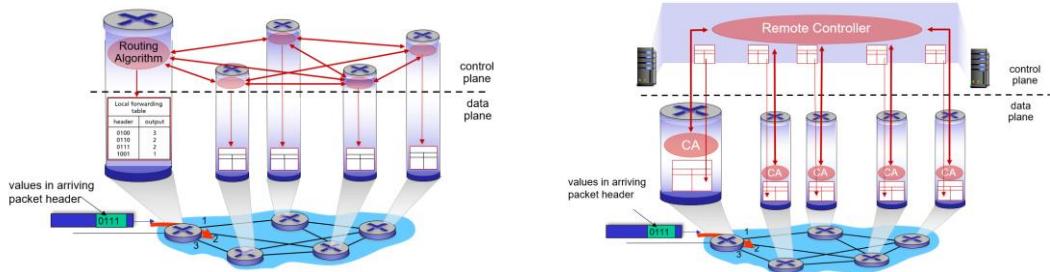
网络层关键的两个功能——转发和路由：

转发：将分组从一个路由器的输入链路移动到路由器合适的输出链路（过单个立交桥往哪）
路由：决定分组从源点到终点的路径（路由算法）（规划起点到终点整个导航路线）

数据平面：本地，每个路由器的功能，决定到达路由器输入端口的数据报如何转发到路由器输出端口。

控制平面：网络范围的逻辑，决定数据报如何在源主机到目的主机的路径上的路由器之间进行路由（两种控制平面的方法：传统路由算法：在路由器中实现，软件定义网络（software-defined networking, SDN）：在远程服务器中实现）。

每-路由器（Per-Router）控制平面：每一个路由器都有单独的路由算法组件，在控制平面进行交互。（左图）



Software-Defined Networking 控制平面：远程的控制器进行计算并与本地控制代理（Controller Agents, CAs）进行交互，将转发表安装到具体的路由器上。（右图）

网络服务模型：Q: 对于从发送方到接收方之间传输数据报的“通道”，网络提供什么样的服务模型？对于单个数据报的示例服务：确保交付，延迟保证，如少于 40ms 的延迟。对于数据流的示例服务：有序数据报交付，保证流的最小带宽，对分组之间间隔变化的限制。

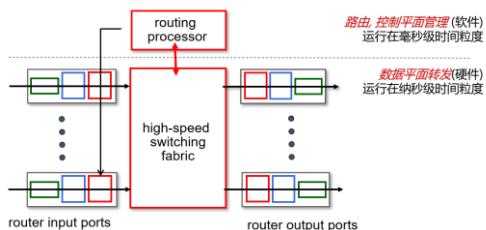
网络层服务模型例子：

| 网络结构 | 服务模型 | Quality of Service (QoS) 保证？ | | | |
|----------|-------------------------------|------------------------------|----------|----------|-----|
| | | 带宽 | 丢包 | 保序 | 时延 |
| Internet | best effort | none | no | no | no |
| ATM | Constant Bit Rate | Constant rate | yes | yes | yes |
| ATM | Available Bit Rate | Guaranteed min | no | yes | no |
| Internet | Intserv Guaranteed (RFC 1633) | yes | yes | yes | yes |
| Internet | Diffserv (RFC 2475) | possible | possibly | possibly | no |

对于尽力而为服务的思考：机制的简单性使互联网得以被广泛部署与应用，充足的带宽供应使得实时应用（如：实时互动语音视频）的性能在大多数时间都足够好，重复的应用层服务（如：数据中心、内容分发网络）连接到靠近用户网络的地方，允许从多个位置提供服务，“弹性”服务的拥塞控制有助于提高网络性能。

路由器组成：输入端口，交换结构，输出端口、缓存管理，调度：

路由器结构概览，通用路由器架构的高层面视图：



输入端口的功能：分布式交换：使用头部字段的值，在输入端口存储中的转发表中查找到相应的输出端口（“匹配+动作”），目标：使输入端口的处理速度达到“线速”，输入端口排队：当数据报的到达速度超过交换结构的转发速度。

基于目的地的转发：仅依赖目的地 IP 地址进行转发（传统方法）

最长前缀匹配 (longest prefix match)：在查找给定目的地地址的转发表项时，使用与目的地地址匹配的最长前缀地址。

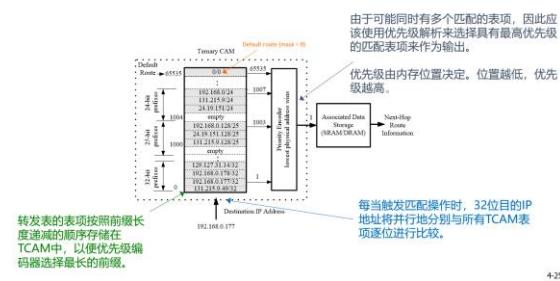
最长前缀匹配：通常使用 ternary content addressable memories (TCAMs) 来执行，内容可寻址：将地址提供给 TCAM，无论表空间有多大，TCAM 可以在一个时钟周期内检索出地址。

Cisco Catalyst 系列路由器：TCAM 中可存储大约 100 万个路由表条目。

使用TCAM实现转发表的查找

| Destination Address Range | Link interface |
|-----------------------------------|----------------|
| 11001000 00010111 00010*** ***** | 0 |
| 11001000 00010111 000110000 ***** | 1 |
| 11001000 00010111 00011*** ***** | 2 |
| otherwise | 3 |

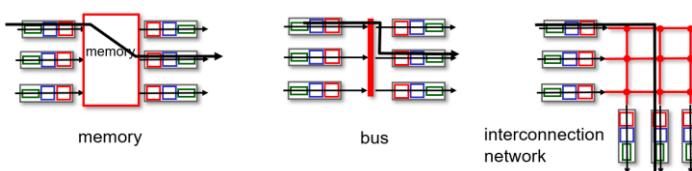
examples: 11001000 00010111 00010110 10100001 which interface? 0
11001000 00010111 000110000 10101010 which interface? 1



(三种主要的) 交换结构: 将分组从输入缓冲区传输到合适的输出端口

交换速率: 分组从输入端口到输出端口传输的速率, 通常以 输入/输出 线速率的倍数作为单位衡量。N个输入端口: 交换速率是线速率的N倍是一个较为理想的情况

▪ 三种主要的交换结构:



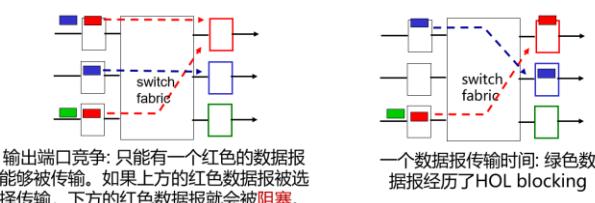
通过内存 (memory) 交换: 第一代路由器: 采用传统的电脑, 在 CPU 直接控制下进行交换, 分组被拷贝到系统内存, 转发速率被内存带宽限制 (每个数据报要通过 bus 两次)

通过总线 (bus) 交换: 数据报通过共享总线从输入端口转发到输出端口。总线竞争: 交换速度受限于总线带宽, 32 Gbps bus, Cisco 5600: 对于接入或者企业级路由器, 速度足够 (但不适合区域或者骨干网络)

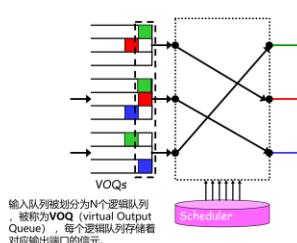
通过互联网络交换: Crossbar 结构, Clos 网络和其它互联网络结构在起初提出来时是为了能够在多处理器结构下进行处理器互联。多级交换机: nxn 交换机由多级小交换机组成。利用并行性: 在入口处将数据报分片为固定长度的信元, 通过互联网络交换信元, 在出口处重新组装信元。并行运用多个交换“平面”进行拓展: 利用并行性来进行加速与拓展。Cisco CRS 路由器: 基础单元: 8 个交换平面, 每个平面: 三级互联网络, 高达 100's Tbps 的交换容量。

输入端口排队:

当交换结构的速率小于所有输入端口速率之和 -> 在输入端口队列处可能可能会发生排队现象, 排队延迟以及由于输入缓存溢出造成了丢包。Head-of-the-Line (HOL) blocking: 排在队头的数据报阻碍了队列中其它数据报的转发。



具有虚拟输出队列 (VOQ) 的输出端口排队



具有虚拟输出队列 (VOQ) 的输出端口排队

输出端口排队: 当数据报从交换结构到达输出端口的速度快于链路传输速率时, 就需要输出端口缓存。丢弃策略:当没有多余的缓存时哪个数据报需要被丢弃? (数据报会因为拥塞、缓存区没有空间等原因而被丢弃) 调度算法选择在队列中排队的数据报中的一个数据报来进行传输 (优先级调度-谁会获得最优的性能, 网络中立原则)。

当数据报通过交换结构的到达速率超过输出端口线速率时, 就需要输出端口缓存
输出端口缓存溢出可能会导致排队延迟和丢包

还有: 输入输出端口结合排队 (Combined Input and Output Queueing)

多少缓存才够用? RFC 3439 经验法则: 缓存大小 B 应该等于平均往返时延 RTT (例如 250ms) 乘以链路容量 C e.g., $C = 10 \text{ Gbps}$ link: 2.5 Gbit buffer

回顾 TCP 拥塞控制: 为了避免缓存干涸 (buffer underflow): Buffer size $B \geq \text{half of the peak window size } W_{\max}$ $(W_{\max}/2)/(RTT) \geq C$ 合并两个不等式: $B \geq RTT \cdot C$

最近一些理论研究:有 N (非常大) 个异步的流时, 所需缓存大小 B 等于: $RTT \cdot C / \sqrt{N}$

缓存管理: 丢弃: 当缓存区已满时要添加或丢弃哪个分组。弃尾: 丢弃到达的分组。优先级: 基于优先级丢弃或移除分组。标记: 决定哪个分组打上拥塞信号的标记 (ECN, RED)。

(四个) 分组调度: 决定下一个要通过链路传输的分组。

FCFS (first come, first served) 调度: 按照分组到达的次序传输到输出端口。也称为 FIFO

Priority: 优先级调度: 到达的流量进行分类, 按照不同的等级进行排队, 任意头部字段都可以用来分类。先传具有缓冲分组的最高优先级队列里的分组, FCFS 结合 priority class round robin (RR) 调度: 到达的流量进行分类, 按照不同的类别进行排队, 任意头部字段都可以用来分类。server 周期性、重复地扫描不同类型队列, 依次从每个类型队列 (如果队列不为空) 里面发送一个完整的分组

Weighted Fair Queueing (WFQ) 调度: 一般化的 Round Robin, 每个类别, i , 有权重, w_i , 在每一次循环中获得相应加权服务量: $w_i / \sum w_i$ 。最小带宽保证(per-traffic-class)

IP: 互联网协议, 数据包格式, 寻址, 网络地址转换, IPv6:

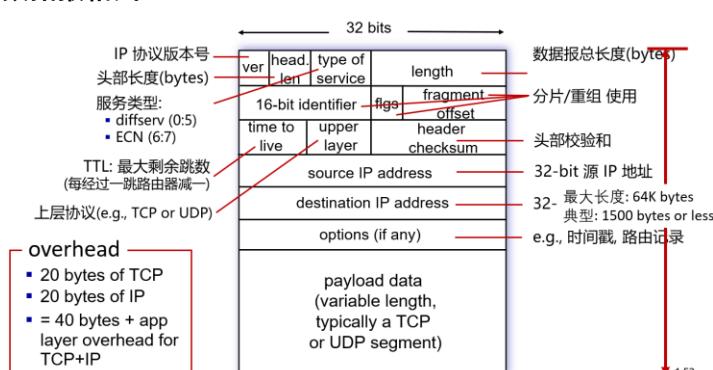
网络层: 互联网: 主机、路由器中的网络层功能:

路径选择算法: 具体实现: 路由协议 (OSPF、BGP), SDN 控制器

IP 协议: 数据报格式、寻址、分组处理约定

ICMP 协议: 错误报告、路由器信号指令

IP 数据报格式:

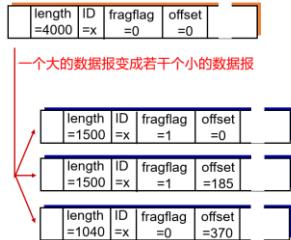


IP 分片和重组: 网络链路有 MTU (最大传输单元) —— 链路层帧所允许携带的最大数据长度, 不同的链路类型有不同的 MTU, 大的 IP 数据报在网络上被分片(fragmented), 一个数据报被分割成若干小的数据报, 其具有: 相同的 ID、不同的偏移量、最后一个分片标记为 0。重组 (reassemble) 只在最终的目标主机进行, IP 头部的信息被用于标识, 排序相关分片。

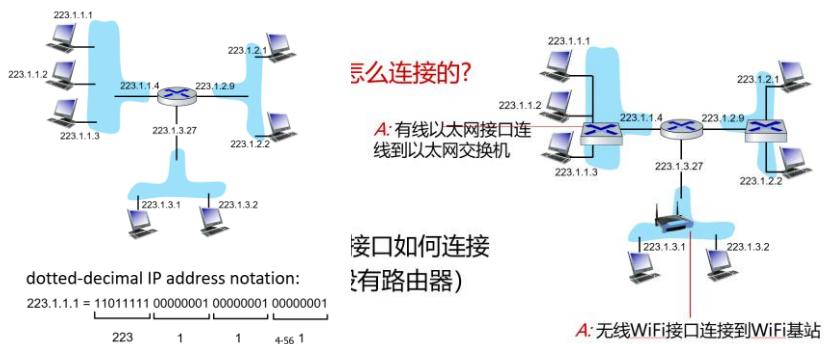
举例:

example:

- 4000 字节数据报
 - 20 字节头部
 - 3980 字节数据
- MTU = 1500 bytes
- 第一片: 20 字节头部 + 1480 字节数据 • 偏移量: 0
- 第二片: 20 字节头部 + 1480 字节数据 • 偏移量: $1480/8=185$
- 第三片: 20 字节头部 + 1020 字节数据 • 偏移量: $2960/8=370$



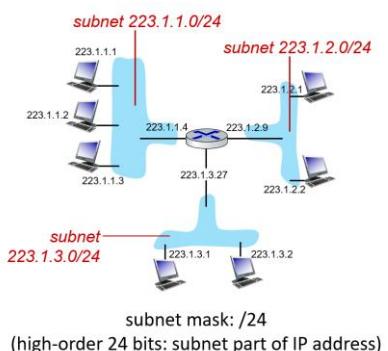
IP 编址: IP 地址: 与每个主机或路由器接口相关联的 32 位标识符。接口: 主机/路由器 和 物理链路的连接处。路由器通常有多个接口, 主机通常有一个或两个接口(e.g., wired Ethernet, wireless 802.11)



子网 (subset): 什么是一个子网? 物理上不需要经过中间路由器就能相互到达的设备接口。

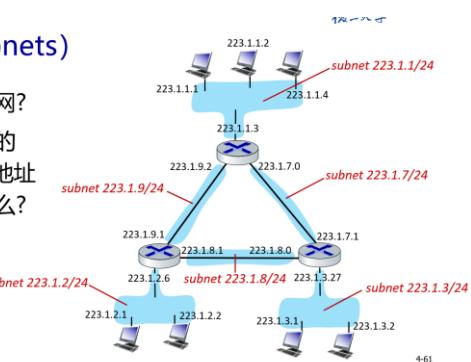
IP 地址的结构: 子网部分: 同一子网中的设备具有共同的高位 bits, 主机部分: 低位 bits。

判断子网的方法: 将每个接口与其主机或路由器分离, 构成一个个网络孤岛, 每一个孤岛网络被称之为一个子网 (subnet)。

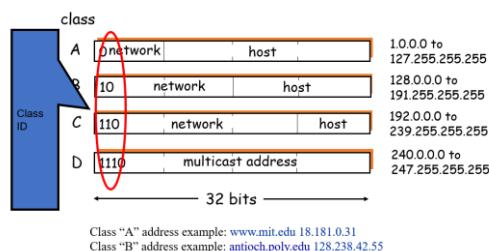


子网 (Subnets)

- 有哪些子网?
- 这些子网的 /24 子网地址 分别是什么?

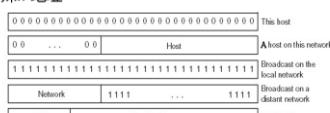


IP 地址分类和特殊 IP 地址:



一些约定:

- 子网部分: 全为 0---本网络
- 主机部分: 全为 0---本主机
- 主机部分: 全为 1---广播地址, 这个网络的所有主机
- 特殊 IP 地址



内网专用地址: 专用地址: 地址空间的一部份供专用地址使用, 永远不会被当做公用地址来分配, 不会与公用地址重复, 只在局部网络中有意义, 区分不同的设备。路由器不对目标地址是专用地址的分组进行转发。专用地址范围:

Class A 10.0.0.0-10.255.255.255 MASK 255.0.0.0

Class B 172.16.0.0-172.31.255.255 MASK 255.255.0.0

Class C 192.168.0.0-192.168.255.255 MASK 255.255.255.0

IP 编址 (CIDR): Classless InterDomain Routing 无类域间路由：地址的子网部分为任意长度，地址格式: a.b.c.d/x, x 是地址中子网部分的长度。

如何获得一个 IP 地址？ 这实际上是两个问题：

1. Q: 主机如何在其网络中获得 IP 地址(地址的主机部分)?
2. Q: 网络如何为自己获取 IP 地址(地址的网络部分)?

系统管理员将地址配置在一个文件中(e.g., /etc/rc.config in UNIX) **DHCP:** Dynamic Host Configuration Protocol: 动态地从服务器获取地址 “plug-and-play” 即插即用。目标: 主机在加入网络时动态地从网络服务器获取 IP 地址，可以在租期快到时更新对主机在用 IP 地址的租期，重新启动时，允许重新使用以前用过的 IP 地址，支持移动用户加入或离开该网络。

DHCP 工作概况：主机广播 DHCP discover 报文 [可选]，DHCP 服务器用 DHCP offer 做出响应 [可选]，主机请求 IP 地址：发送 DHCP request 报文，DHCP 服务器发送地址：DHCP ack 报文。通常情况下，DHCP 服务器将位于路由器中，为路由器所连接的所有子网提供服务。DHCP 可以返回不仅仅是子网中的 IP 地址：第一跳路由器的 IP 地址(默认网关)，DNS 服务器的域名和 IP 地址，子网掩码 (指示地址的网络部分和主机部分)



Q: 如何获得一个网络 IP 地址的子网部分？

A: 从 ISP 获得的地址块中分配一个小地址块

Q: ISP 如何获得地址块？

A: ICANN: Internet Corporation for Assigned Names and Numbers <http://www.icann.org/>

通过 5 个区域注册中心(regional registries, RRs)分配 IP 地址(然后这些注册中心可以分配给本地注册中心)，管理 DNS root zone，包括顶级域名管理(.com, .edu, …)

Q: 是否有足够的 32 位 IP 地址？ ICANN 在 2011 年将最后一批 IPv4 地址分配给了 RRs，NAT 帮助缓解 IPv4 地址空间耗尽的问题，IPv6 有 128 位的地址空间。

NAT: 网络地址转换：

NAT: 对于外界而言，本地网络中的所有设备共享一个 IPv4 地址。



实现: NAT 路由器必须具备的能力：外出数据报: 替换 (source IP address, port #) 为 (NAT IP address, new port #)，远端的 clients/servers 将会用 (NAT IP address, new port #) 作为目的地址。记住每个(source IP address, port #) 和 (NAT IP address, new port #) 转换对 (在 NAT 转换表)。到来数据报: 替换到来数据报 (NAT IP address, new port #) 为 NAT 转换表中对应的(source IP address, port #)

NAT 是有争议的：路由器本应该只处理到第三层，地址短缺可以被 IPv6 解决，违反端到端原则(端口号会被网络层设备修改)，NAT 穿越：如果客户端要连接 NAT 路由器后的服务器要怎么办？但 NAT 仍在被使用：广泛用于家庭和机构网，4G/5G 蜂窝网。

NAT 穿越问题：客户端需要连接地址为 10.0.0.1 的服务器，服务器地址 10.0.0.1 LAN 本地地址 (客户端不能够使用其作为目标地址)，整网只有一个外部可见地址: 138.76.29.7。

方案 1：静态配置 NAT：转发进来的对服务器特定端口连接请求, e.g., (123.76.29.7, port 2500) 总是转发到 10.0.0.1 port 25000

方案 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) 协议。允许 NATted 主机可以: 获知网络的公共 IP 地址 (138.76.29.7), 列举存在的端口映射, 增/删端口映射 (在租用时间内) i.e., 自动化静态 NAT 端口映射配置

方案 3: 中继 (used in Skype): NAT 后面的服务器建立和中继的连接，外部的客户端链接到中继，中继在 2 个连接之间桥接

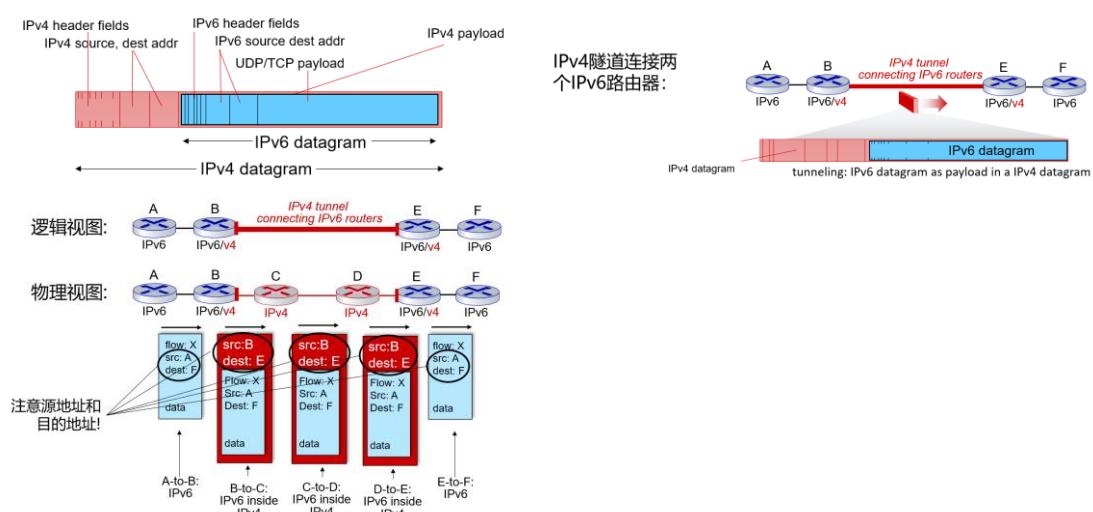


IPv6: 初始动机: 32 位 IPv4 地址空间将被完全分配。额外的动机: 处理/转发速度: 40-byte 固定头部长度, 支持对“流”进行不同的网络层处理。IPv6 报文格式如下:



4-89

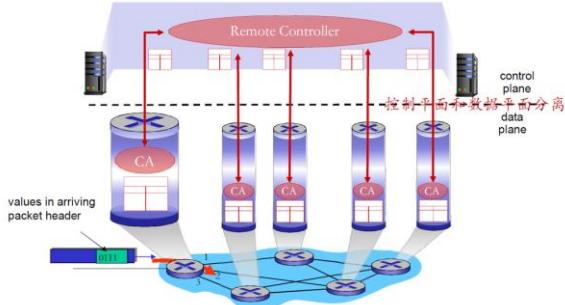
在 IPv4 和 IPv6 路由器混合使用的情况下，网络是如何运转的？**隧道 (Tunneling)** : IPv6 数据报作为 IPv4 数据报的有效载荷在 IPv4 路由器之间传输(“packet within a packet”), 隧道在其他场景中广泛使用(4G/5G)。



泛化转发, SDN, 匹配+动作, OpenFlow: 匹配+动作的实例:

传统方式实现网络功能问题：垂直集成-->昂贵、不便于创新的生态，分布式、固化设备功能==网络设备种类繁多，无法改变路由等工作逻辑，无法实现流量工程等高级特性，配置错误影响全网运行；升级和维护会涉及到全网设备：管理困难。要增加新的网络功能，需要设计、实现以及部署新的特定设备，设备种类繁多。。~2005：开始重新思考网络控制平面的处理方式：集中：远程的控制器集中实现控制逻辑，远程：数据平面和控制平面的分离。

SDN：逻辑上集中的控制平面：一个不同的（通常是远程）控制器和 CA 交互，控制器决定分组转发的逻辑（可编程），CA 所在设备执行逻辑。



SDN 主要思路：网络设备数据平面和控制平面分离

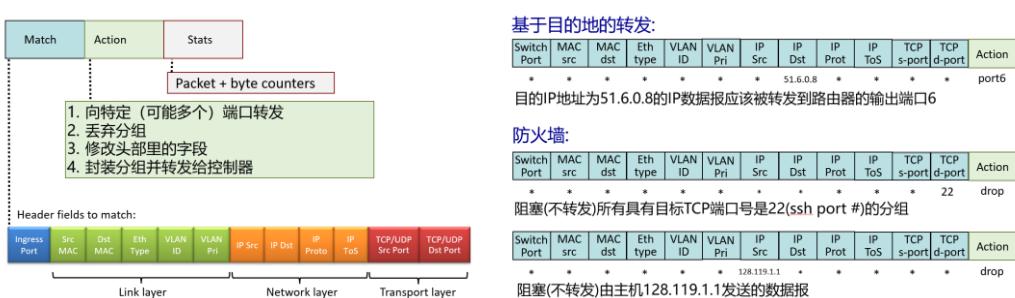
数据平面-分组交换机，将路由器、交换机和目前大多数网络设备的功能进一步抽象成：按照流表（由控制平面设置的控制逻辑）进行 PDU（帧、分组）的动作（包括转发、丢弃、拷贝、泛洪、阻塞），统一化设备功能：SDN 交换机（分组交换机），执行控制逻辑

控制平面 = 控制器+网络应用：分离、集中，计算和下发控制逻辑：流表。

泛化转发：匹配+动作：回顾：每个路由器都有一张转发表，“匹配+动作 (match plus action)”：匹配到达报文中的字段，采取动作。前面讲基于目的地的转发根据目的 IP 进行转发。泛化转发：许多头部字段可以决定动作，并且有许多可选的动作：drop/copy/modify/log packet

流表抽象：流：由头部字段所定义（链路层、网络层、传输层的字段），泛化转发：简单的分组处理逻辑，匹配：分组头部字段的模式 (pattern) 值，动作：对于匹配的分组：丢弃、转发、修改、将匹配的分组发送给控制器，优先级：几个模式匹配了，优先采用哪个，消除歧义，计数器：#bytes 以及 #packets。

Openflow 流表表项结构：



Openflow 抽象：匹配+动作：统一化各种网络设备提供的功能

路由器：match：最长前缀匹配 IP 目的地址，action：通过一条链路转发

交换机：match：目的 MAC 地址，action：转发或者泛洪

防火墙：match：IP 地址和 TCP/UDP 端口号，action：允许或者禁止

NAT：match：IP 地址和端口号，action：重写地址和端口

泛化转发总结：“匹配+动作”抽象：匹配到达分头部中的字段，采取动作，匹配多个域(链路层、网络层、传输层)，动作：丢弃、转发、修改或发送匹配的分组到控制器，可编程的网络范围行为，“网络可编程性”的简单形式，可编程的，逐分组的处理，历史根源：active

networking 主动网络。现在: 更加泛化的编程: P4 (see p4.org).

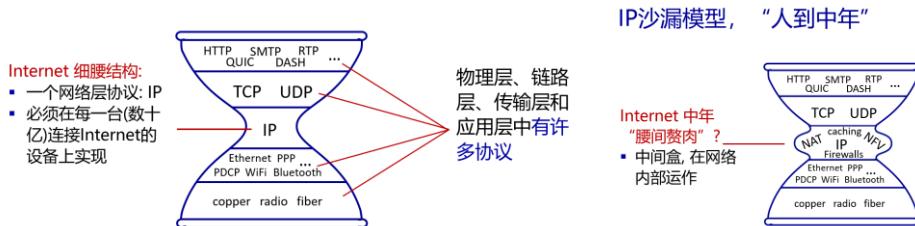
中间盒:

在源主机和目的主机之间的数据路径上, 除 IP 路由器的一般标准功能外, 执行其他功能的任何中间设备。起初:专有(封闭)硬件解决方案, 向实现开放 API 的“白盒”硬件迈进, 逐渐脱离专有硬件解决方案, 可编程的本地动作 (通过 match+action), 在软件领域走向创新/差异化, SDN: (逻辑上)集中控制和配置管理, 通常在私有/公共云中。网络功能虚拟化(network functions virtualization, Nfv): 基于白盒网络、计算、存储的可编程服务。

中间盒无处不在:



IP 沙漏模型:



互联网架构原则三基本信念: 简单的连通性、IP 协议: 细腰、网络边缘的智能性和复杂性

第五章 网络层-控制平面

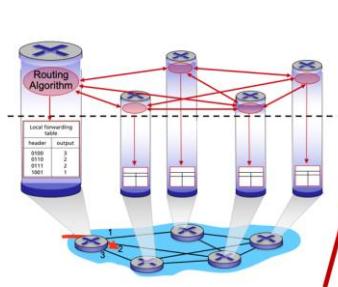
回顾: 网络层功能:

转发: 将分组从路由器的输入端口移动到适当的输出端口 (数据平面)

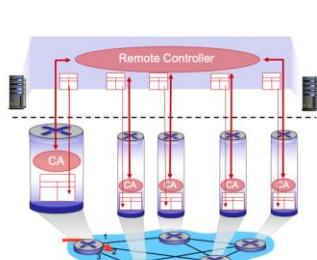
路由: 确定数据包从源到目的的路由路径 (控制平面)

构建网络控制平面的两种方法: 每个路由器的控制功能实现(传统), 逻辑上集中的控制功能实现 (software defined networking)。

每个路由器的控制平面

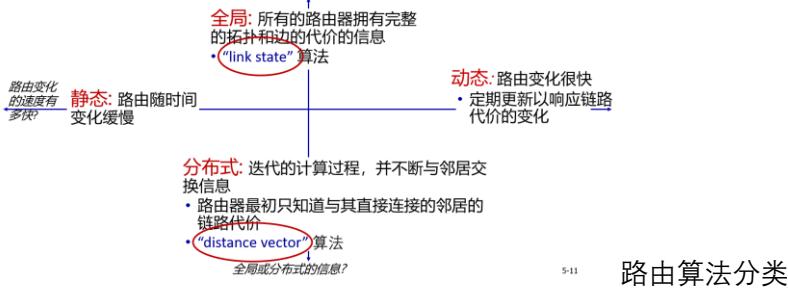


SDN 控制平面



路由协议: 链路状态算法 link state, 距离向量算法 distance vector:

路由协议目标 : 按照某种指标(传输延迟, 所经过的站点数目等)找到一条从源节点到目标节点的较好路径。路径: 分组从给定的初始源主机到最终目的主机的路由器遍历序列, 指标: “成本”最小, “最快”, “最不拥挤”, 路由: 前 10 名的网络挑战! 图数据结构抽象链路代价。

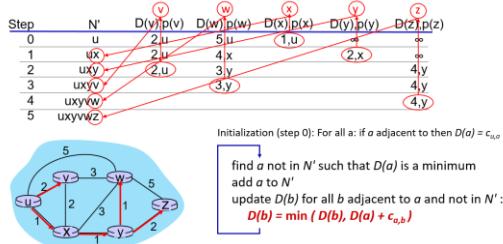


5-11

路由算法分类

Dijkstra link-state 路由算法: 集中的: 网络拓扑结构、链路代价为所有节点所知, 通过“link state broadcast”完成, 所有节点都有相同的信息, 计算从一个节点(“源”)到所有其他节点的最小代价路径, 给出该节点的转发表, 迭代的: 经过 k 次迭代, 知道到 k 个目的地的代价最小的路径。 $c_{x,y}$: 从节点 x 到节点 y 的直连链路代价; $= \infty$ 如果不是直接相邻的邻居。 $D(v)$: 从源到目的地 v 的当前最小成本路径的预估代价。 $p(v)$: 从源到 v 的路径上的前导节点。 N' : 最小代价路径明确已知的节点形成的节点集。

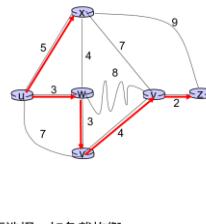
Dijkstra 算法: 实例



| Step | N' | $D(v)$ | $D(w)$ | $D(x)$ | $p(v)$ | $D(y)$ | $p(w)$ | $D(z)$ | $p(y)$ |
|------|------------------|----------|----------|----------|--------|----------|--------|----------|--------|
| 0 | u | ∞ | ∞ | ∞ | | ∞ | | ∞ | |
| 1 | u, w | 7, u | 3, u | 5, u | 7, u | ∞ | | ∞ | |
| 2 | u, w, x | 6, w | 6, w | 5, u | 11, w | ∞ | | ∞ | |
| 3 | u, w, x, v | 6, w | 6, w | 5, u | 11, w | 14, x | 11, w | ∞ | |
| 4 | u, w, x, v, y | 6, w | 6, w | 5, u | 10, v | 14, x | 11, w | 12, y | 12, y |
| 5 | u, w, x, v, y, z | 6, w | 6, w | 5, u | 6, w | 12, y | 10, v | 12, y | 12, y |

notes:

- 通过跟踪前导节点构造最小代价路径树
- 可能存在多条最小代价路径, 可根据目标选择, 如负载均衡



5-28

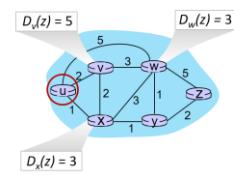
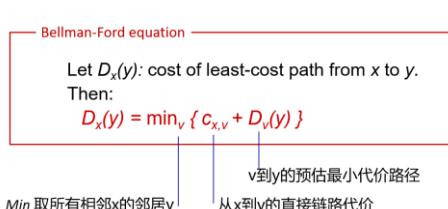
算法复杂度: n 个节点, 每一次迭代: 需要检查所有不在 N' 中节点, $n(n+1)/2$ 次比较: $O(n^2)$ 复杂度。更有效的实现: $O(n \log n)$ (斐波那契堆)。

消息复杂度: 每台路由器必须向其它 n 台路由器广播自己的链路状态信息, 高效的广播算法: 通过 $O(n)$ 个链路来传播来自一个源的广播消息, 每台路由器的消息要经过 $O(n)$ 条链路, 总的消息复杂度为 $O(n^2)$ 。

Distance Vector 算法: 基于 Bellman-Ford (BF) 等式 (动态规划)。

Bellman-Ford 算法 例子

假设 u 的邻居节点, x, v, w, 已知它们到 z 的最小代价路径:

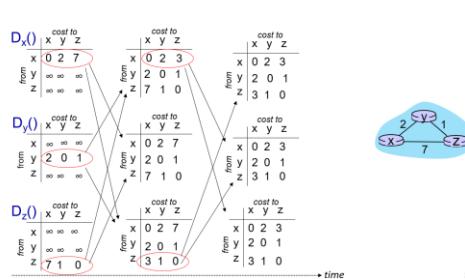


由 Bellman-Ford 等式 可知:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,w} + D_w(z), \\ &\quad c_{u,x} + D_x(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

取最小值的节点 x 是 u 到达目的地 z 的预估最小代价路径上的一跳

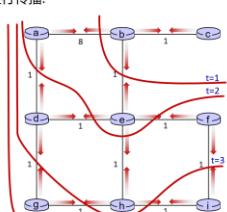
5-33



Distance Vector: 状态信息扩散

迭代通信, 每步计算都会将信息在网络中进行传播:

- t=0 c 在 t=0 时刻的状态只在 c
- t=1 c 在 t=0 时刻的状态已经传播到 b, 并且可能影响到 1 跳距离的 DV 计算, 如 b 处
- t=2 c 在 t=0 时刻的状态现在可能会影响 2 跳距离的 DV 计算, 如在 d, e 处
- t=3 c 在 t=0 时刻的状态现在可能会影响 3 跳距离的 DV 计算, 如在 g, h 处
- t=4 c 在 t=0 时刻的状态现在可能会影响 4 跳距离的 DV 计算, 如在 g, i 处



5-50

Distance Vector: 链路代价变化

链路代价变化:

- 节点检测到本地链路代价变化
- 更新路由信息，重新计算本地DVs
- 如果DV有变化，通知邻居

*t₀: y检测到链路代价的变化，更新它的DV，并通知它的邻居
“好消息传得快” t₁: z接收到y的更新信息，更新了它自己的DV，计算出到x的最小代价，把它的新DV发给它的邻居
t₂: z接收到z的更新信息，更新它自己的DV。y的DV中最小代价不变，所以不向z发送消息。*

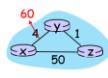


5-52

Distance Vector: 链路代价变化

链路代价变化:

- 节点检测到本地链路代价变化
- **“坏消息传得慢” – 无穷计数问题:**
 - y检测到x的直接链路的新代价是60，但z到x有一条路径的成本是5。y计算出，通过z到x的新代价是6，所以z计算到x的新成本是7，通过y；通知y到x新路径的代价是7。
 - y知道通过z到达x的路径的新代价是7，所以y计算到x的新成本是8，通过z；通知z到x新路径的代价是8。
 - z知道通过y到达x的路径的新代价是8，所以z计算到x的新成本是9，通过y；通知y到x新路径的代价是9。
 - ...
- 解决方案详见课本。分布式算法很 *tricky* !



5-53

LS 算法和 DV 算法比较：消息复杂度：LS: n 台路由器，发出 $O(n^2)$ 消息，DV: 邻居间交换信息；收敛时间变化。收敛速度：LS: $O(n^2)$ 算法， $O(n^2)$ 消息，可能会有振荡。DV: 收敛时间变化，可能存在路由环路，无穷计数问题。健壮性：如果路由器发生故障或被破坏，会发生什么？LS: 路由器会通告不正确的链路代价，每个路由器只计算自己的路由表，错误信息影响较小，局部。DV: DV 路由器可能通告全网所有路由器不正确的路径代价：黑洞。每个路由器的 DV 可以被其他路由器使用：错误可以扩散到整个网络。

ISP 内路由: OSPF:

Internet 实现可扩展性路由的方法：将路由器聚合到称为“autonomous systems” (AS, 自治系统)的区域中 (亦称“domains”)。intra-AS (亦称 “intra-domain”)：同一 AS 内路由器之间的路由，AS 内所有路由器必须运行相同的域内协议，不同自治系统中的路由器可以运行不同的域内路由协议，网关路由器：在 AS 的边缘，通过链路连接到其它 AS 的路由器。inter-AS (亦称 “inter-domain”)：AS 之间的路由，网关执行域间路由(以及域内路由)

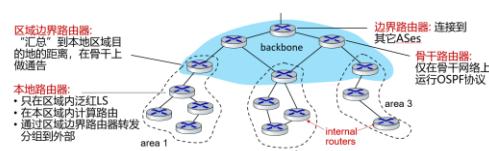
AS 内路由：最常使用的 AS 内路由协议：**RIP**: Routing Information Protocol [RFC 1723]

classic DV: 每 30 秒交换一次 DV(不再广泛使用)。EIGRP: Enhanced Interior Gateway Routing Protocol，基于 DV，几十年来一直是 Cisco 的专利 (在 13 年开源 [RFC 7868])。**OSPF**: Open Shortest Path First [RFC 2328]，链路状态路由，IS-IS protocol (ISO 标准，非 RFC 标准)，本质上与 OSPF 相同。

OSPF (Open Shortest Path First) 路由：“open”: 公开可用的，经典的链路状态 LS，每台路由器将 OSPF 的链路状态通告(直接通过 IP 而不是使用 TCP/UDP)传播给整个 AS 内的所有其他路由器，多个可能的链路代价指标：带宽，延迟，每个路由器都有完整的网络拓扑，使用 **Dijkstra 算法**计算转发表，安全：所有的 OSPF 报文都是经过认证的 (防止恶意的攻击)。

层次化的OSPF

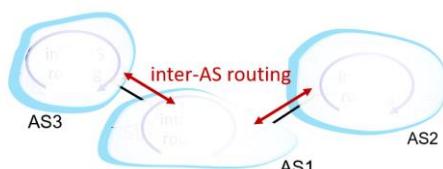
- **2个级别的层次性:** 本地, 骨干。
 - 链路状态通告仅仅在本地区域范围内进行
 - 每个节点都有详细的区域拓扑;只知道到达其他目的地的方向



5-62

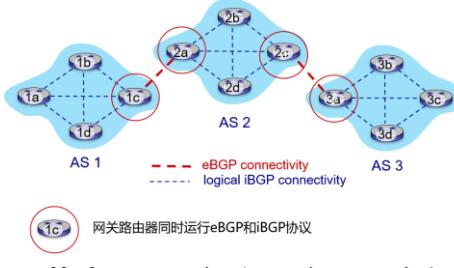
ISP 间路由: BGP:

inter-AS (亦称 “inter-domain”): AS 之间的路由



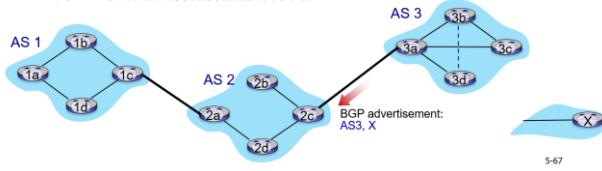
BGP (Border Gateway Protocol): 域间路由协议，“事实上的”标准，“将互联网连接在一起的

“粘合剂”。允许子网向 Internet 的其他部分通告它的存在以及它可以到达的目的地: “I am here, here is who I can reach, and how”。BGP 为每个 AS 提供了: 从邻近的 AS 获取目标网络可达性信息 (eBGP), 根据可达性信息和策略确定到其他网络的路由, 向所有 AS 内部路由器传播可达性信息 (iBGP), 通告(向邻近网络)目的地可达性信息。



BGP 基础: BGP 会话: 两个 BGP 路由器 (“peers”) 通过一个半永久 TCP 连接交换 BGP 报文: 通告到不同目的网络前缀的路径 (BGP 是一种“路径向量”协议)

- 当 AS3 网关路由器 3a 向 AS2 的网关路由器 2c 通告路径: AS3, X
 - AS3 向 AS2 承诺它会将数据报转发给 X



5-67

BGP 协议报文: 使用 TCP 协议交换 BGP 报文, BGP 报文 [RFC 4371]:

OPEN: 打开 TCP 连接, 认证发送方

UPDATE: 通告新路径(或者撤销原路径)

KEEPALIVE: 在没有更新时保持连接, 也用于对 OPEN 请求确认

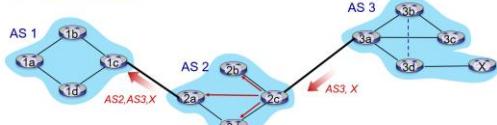
NOTIFICATION: 报告以前报文中的错误; 也用来关闭连接

路径属性和 BGP 路由: BGP 通告路由: prefix + attributes。prefix: 通告的目的地前缀。两个重要的 attributes: AS-PATH: 前缀的通告所经过的 AS 列表, NEXT-HOP: 指定到下一跳 AS 的特定 AS 内部路由器。基于策略的路由: 当一个网关路由器接收到了一个路由通告, 使用输入策略来接受或拒绝 (accept/decline) (如: 不要路由通过 AS Y)。AS 策略还决定是否向其他邻近的 AS 发布路由信息。

路径策略的作用:

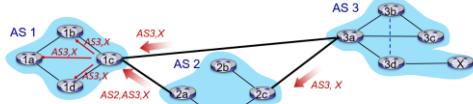


BGP 路径通告



- 路由器 AS2.2c 从路由器 AS3.3a 收到的 AS3, X 路由通告(通过 eBGP)
- 基于 AS2 的策略, 路由器 AS2.2c 决定接收 AS3, X 的通告, 而且通过 iBGP 向 AS2 的所有路由器进行通告
- 基于 AS2 的策略, 路由器 AS2.2a 通过 eBGP 向 AS1.1c 路由器通告 AS2, AS3, X 路由信息

BGP 路径通告: 多路径

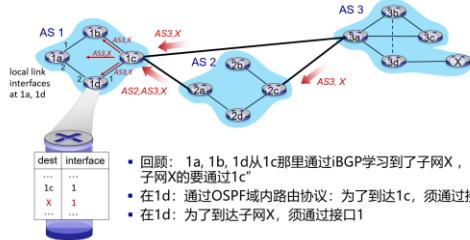


网关路由器可以学习到目的地的多条路径:

- AS1 网关路由器 1c 从 2a 学习到路径: AS2, AS3, X
- AS1 网关路由器 1c 从 3a 学习到路径: AS3, X
- 基于策略, AS1 网关路由器 1c 选择了路径: AS3, X, 而且通过 iBGP 告诉所有 AS1 内部的路由器

5-73

BGP: 转发表



- 回顾：1a, 1b, 1d从1c那里通过iBGP学习到了子网X，“到往子网X的要通过1c”
- 在1d：通过OSPF域内路由协议：为了到达1c，须通过接口1
- 在1d：为了到达子网X，须通过接口1

5-74

BGP: 转发表



- 回顾：1a, 1b, 1d从1c那里通过iBGP学习到了子网X，“到往子网X的要通过1c”
- 在1d：通过OSPF域内路由协议：为了到达1c，须通过接口1
- 在1d：为了到达子网X，须通过接口1
- 在1a：通过OSPF域内路由协议：为了到达1c，须通过接口2
- 在1a：为了到达子网X，须通过接口2

5-75

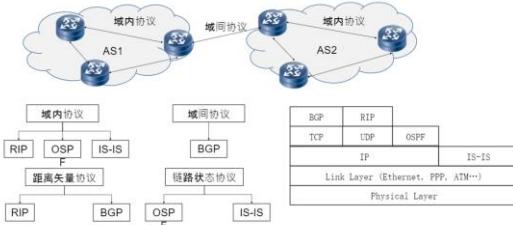
Hot potato routing：热土豆路由：2d 通过 iBGP 获知，它可以通过 2a 或者 2c 到达 X

热土豆路由：选择具备最小内部区域代价的网关 (如：2d 选择 2a，即使往子网 X 可能有比较多的 AS 跳数): 不要担心域间的代价!

BGP 路由选择：路由器可以了解到目的 AS 的多条路由，并根据以下标准来选择路由：本地偏好值属性：偏好策略决定。最短 AS-PATH：AS 的跳数。最近的 NEXT-HOP 路由器：热土豆路由。其它标准。

AS 内路由和 AS 间路由的区别：策略: inter-AS: 管理员想要控制它的流量如何路由，谁在使用它的网络进行数据传输。intra-AS: 单一管理员，所以策略不是问题。规模: 分层路由节省了表大小，减少了更新流量。性能: intra-AS 专注于性能，inter-AS 策略可能比性能更重要

路由协议分类

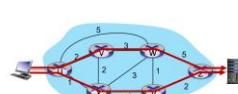


SDN 控制平面：

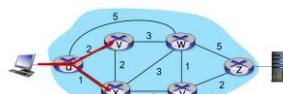
远程控制器计算，并在路由器中安装转发表

为什么是逻辑上集中的控制平面？更容易管理网络：避免路由器错误配置，更灵活的流量管理。基于流表的转发(回顾 OpenFlow API) 允许“可编程的”路由器。集中式“编程”更容易：集中计算流表然后分发，分布式“编程”更困难：在每个单独的路由器上分别运行分布式的算法协议，得到转发表。控制平面的开放(非私有)实现，培育创新.百花齐放。

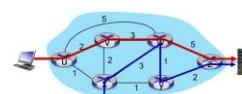
流量工程：传统路由较为困难：



- Q: 如果网络运营商希望u到z流量沿着路径uvw走而不是uxyz，该怎么办？
A: 需要重新定义链路权重，以便流量路由算法根据权重计算路由(或者需要一个新的路由算法！)
链路权重只是控制的“旋钮”：没有太多的控制！



- Q: 如果网络运营商希望沿uvw和uxyz(负载均衡)分割u到z的流量，该怎么办？
A: 做不到(基于目的地的转发，以及LS, DV路由)



- Q: 如果w想把蓝色和红色的流量从w路由到z通过不同的路径呢？
A: 做不到(基于目的地的转发，以及LS, DV路由)
我们在第4章中学习了泛化转发和SDN可以用来实现任何预期的路由

5-81

数据平面交换机：在硬件上实现数据平面泛化转发(章节 4.4)的快速、简单的商用交换机

流表由控制器计算和安装，基于表的交换机控制 API (e.g., OpenFlow)，定义了哪些可以被控制哪些不能，与控制器通信的协议 (e.g., OpenFlow)

SDN 控制器 (网络 OS)：维护网络状态信息，通过北向 API 与“上面”的网络控制应用程序交互，通过南向 API 与“下面”的网络交换机交互，逻辑上集中，但是在实现上通常由于性能、可扩展性、容错性以及鲁棒性采用分布式方法

网络控制应用：控制的“大脑”：采用下层提供的服务（SDN 控制器提供的 API），实现网络控制功能，非捆绑：可以由第三方提供。与交换机厂商可以不同，与控制器厂商也可以不同。

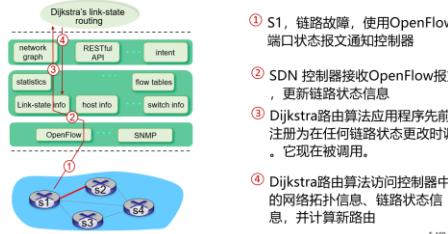
Openflow 协议：在控制器和交换机之间运行，采用 TCP 来交换报文，加密可选。三种 OpenFlow 报文：控制器到交换机、异步（交换机到控制器）、对称（misc.）。不同于 OpenFlow API，用于指定泛化转发动作的 API。

Openflow 控制器到交换机报文：关键的控制器到交换机报文。特性：控制器查询交换机特性，交换机应答。配置：控制器查询/设置交换机的配置参数。修改状态：增加、删除、修改

OpenFlow 流表中的表项。packet-out：控制器可以将分组通过特定的交换机端口发出

Openflow 交换机到控制器报文：关键的交换机到控制器报文。packet-in：将分组（和它的控制）转交给控制器；参见来自控制器的 packet-out 消息。流移除：删除交换机上的流表表项。端口状态：通知控制器端口的变化。

SDN：控制平面与数据平面交互的实例



SDN：控制平面与数据平面交互的实例



5-101

SDN 面临的挑战：强化控制平面：可信、可靠、性能可扩展、安全的分布式系统。失效的鲁棒性：利用控制平面可靠分布式系统的强大理论。可靠，安全：从开始就是“内置”的？

满足特定任务要求的网络、协议，e.g. 实时性，超高可靠性、超高安全性。Internet 扩展性：不仅仅在一个 AS 内，SDN 在 5G 蜂窝网络中面临危急情况。

SDN 与传统网络协议的未来：SDN 计算相比路由器计算转发表：逻辑集中式计算与协议计算的一个例子，可以想象 SDN 计算的拥塞控制：控制器根据路由器向控制器报告的拥塞程度设置发送方速率。

Internet Control Message Protocol (因特网控制报文协议)：

ICMP: Internet Control Message Protocol

- 主机和路由器用于通信网络层信息的协议
- 错误报告：主机不可到达、网络、端口、协议
- echo 请求与回复（ping 所用）
- ICMP 处在网络层，但在 IP 之上：
 - 在 IP 数据报中携带 ICMP 报文
- ICMP 报文：类型，编码，加上 IP 数据报的头 8 字节

| Type | Code | Description |
|------|------|---|
| 0 | 0 | echo reply (ping) |
| 3 | 0 | dest. network unreachable |
| 3 | 1 | dest host unreachable |
| 3 | 2 | dest protocol unreachable |
| 3 | 3 | dest port unreachable |
| 3 | 6 | dest network unknown |
| 3 | 7 | dest host unknown |
| 4 | 0 | source quench (congestion control - not used) |
| 8 | 0 | echo request (ping) |
| 9 | 0 | route advertisement |
| 10 | 0 | router discovery |
| 11 | 0 | TTL expired |
| 12 | 0 | bad IP header |

Traceroute 和 ICMP



- 源主机向目的主机发送一组 UDP 段
- 第 1st set has TTL = 1, 2nd set has TTL = 2, etc.
- 第 nth 组中的数据报到达 nth 路由器：
 - 路由器丢弃数据报，向源主机发送 ICMP 报文 (type 11, code 0)
 - ICMP 报文可能包括路由器的名称和 IP 地址
- 当 ICMP 报文到达源主机时：记录 RTTs

停止判则：

- UDP 段最终到达目的主机
- 目的主机返回 ICMP “端口不可达” 报文 (type 3, code 3)
- 源主机获得这个报文时，停止

网络配置管理：SNMP、NETCONF/YANG：

什么是网络管理：autonomous systems (自治系统，亦称“network”)：由上千个相互作用的软件和硬件部件组成，其他需要监控、配置、控制的复杂系统：喷气式飞机、核电站等等。“网络管理”包括了硬件、软件和人为元素的部署、集成和协同，以便监测、测试、轮询、配置、分析、评价和控制网络和网元资源，用合理的成本满足实时性、性能和服务质量的要求。

网络管理的组件



网络运营商的管理方法

CLI (Command Line Interface)

- 运营商直接发送 (types, scripts) 到单个设备 (e.g., via ssh)

SNMP/MIB

- 运营商使用 Simple Network Management Protocol (SNMP) 查询/设置设备数据 (MIB)

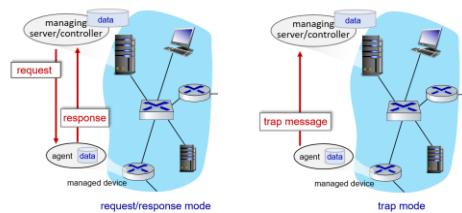
NETCONF/YANG

- 更抽象、网络范围、整体
- 强调多设备配置管理
- YANG 数据建模语言
- NETCONF+YANG 兼容的操作/数据在远程设备之间进行通信

5-113

SNMP协议

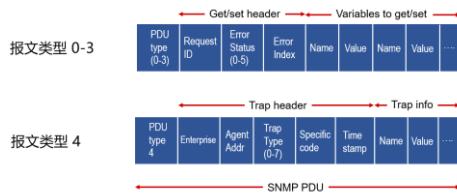
有两种方式传递MIB信息，命令：



SNMP协议：报文类型

| 报文类型 | 功能 |
|--|--|
| GetRequest GetNextRequest GetBulkRequest | 管理实体-代理：“给我数据”(data instance, next data in list, block of data). |
| SetRequest | 管理实体-代理：设置MIB的值 |
| Response | 代理-管理实体：值，对请求的响应 |
| Trap | 代理-管理实体：通知管理实体异常事件 |

SNMP协议：报文格式



SNMP: Management Information Base (MIB)

- 受管理设备的操作和配置数据
- 汇聚到设备 MIB 模块
 - RFC 中定义了 400 个 MIB 模块；更多供应商定义的 MIB 模块
- Structure of Management Information (SMI): 数据定义语言
- 示例：UDP 协议的 MIB 变量：

| Object ID | Name | Type | Comments |
|-----------------|-----------------|----------------|--|
| 1.3.6.1.2.1.7.1 | UDPLnDatagrams | 32-bit counter | total # datagrams delivered |
| 1.3.6.1.2.1.7.2 | UDNNoPorts | 32-bit counter | # undeliverable datagrams (no application at port) |
| 1.3.6.1.2.1.7.3 | UDNErrors | 32-bit counter | # undeliverable datagrams (all other reasons) |
| 1.3.6.1.2.1.7.4 | UDPOutDatagrams | 32-bit counter | total # datagrams sent |
| 1.3.6.1.2.1.7.5 | udpTable | SEQUENCE | one entry for each port currently in use |

5-117

NETCONF 概览：目标：主动管理/配置网络范围内的设备，运行在管理服务器和被管理网络设备之间，操作：检索、设置、修改、激活配置，跨多个设备的原子提交操作，查询运行数据和统计信息，订阅来自设备的通知，Remote Procedure Call (RPC) 范式，用 XML 编码的 NETCONF 协议报文，通过安全可靠的传输协议(如 TLS)交换。

第六章 链路层和局域网

链路层导论：链路层负责通过链路将数据报从一个节点传输到物理上相邻的节点

术语：主机，路由器：节点

沿着通信路径连接相邻节点的通信通道：链路。有线/无线。局域网

第二层包：frame(帧)，封装了数据报

链路层背景：

不同链路层协议在不同链路上传输数据报。e.g., WiFi 在一条链路, Ethernet 在另一条链路

每个链路层协议提供不同的服务 e.g., 能否可以通过链路提供可靠的数据传输



链路层服务：成帧 (Framing), 链路接入 (link access): 将数据报封装成帧，加上帧头、帧尾部。如果采用的是共享介质，信道接入获得信道访问权。在帧头部使用“MAC”地址来标示源和目的(不同于 IP 地址!)。相邻节点间可靠交付(Reliable Delivery)。很少用于低比特差错率的链路。无线链路: 高比特差错率。

Q: 为什么需要同时具有链路级（链路层）和端到端（传输层）可靠性？（作业）

流控：调节相邻发送和接收节点之间的速度

差错检测：由信号衰减、噪声引起的差错，接收端检测错误、发出重传信号或丢帧

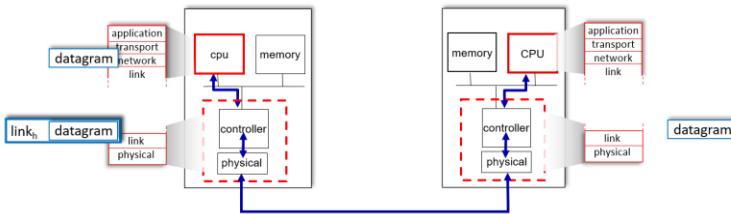
差错纠正：接收端识别并纠正比特错误而不重传

半双工和全双工：对于半双工，链路两端的节点可以传输，但不能同时传输

主机端链路层实现：在每一个主机、路由器、交换机每一个端口上，在片上或网卡(network

interface card, NIC)中实现的链路层实现了链路层和相应物理层的功能，连接到主机的系统总线，硬件、软件、固件的结合

接口通信：



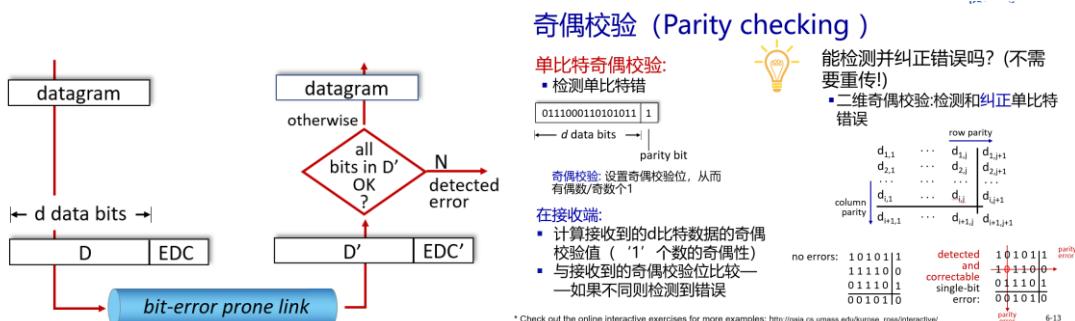
发送端：在帧中封装数据报，增加差错控制编码，可靠的数据传输，流量控制等

接收端：检查有无出错，可靠的数据传输，流量控制等，解封装数据报，将其交给上层

检错、纠错：

差错检测：EDC: 差错检测和纠正比特 Error Detection and Correction bits (e.g., redundancy)

D: 受错误检查保护的数据，包含头部字段。差错检测并非 100% 可靠！协议会漏检一些错误，但很少。EDC 字段越长，检测和校正效果越好



Internet 校验和：目标：检测传输段中的错误(如，翻转位)

发送方：将 UDP 段的内容(包括 UDP 头部字段和 IP 地址)视为 16 位整数序列，校验和：分段内容的相加 (得到的和的反码)，校验和放进 UDP 校验和字段

接收方：计算接收的段的校验和，检查计算的校验和是否等于校验和字段值：不相等——检测到错误；相等——没有检测到错误，但可能还是有错误

循环冗余检测 Cyclic Redundancy Check (CRC)：更强大的差错检测码。

CRC: 例子

- D: 数据比特(假设这些是二进制数)
 - G: 比特模式 (生成多项式), $r+1$ 比特 (在CRC标准中规定)
- | | |
|--|---|
| $\leftarrow d \text{ data bits} \rightarrow$ | $\leftarrow r \text{ CRC bits} \rightarrow$ |
| D | R |
| bits to send | formula for these bits |
- $$< D, R > = D \cdot 2^r \text{ XOR } R$$

- 发送方：**计算 r 位 CRC 比特 R，使 $< D, R >$ 正好被 G 整除 (模2运算)
- 接收方知道 G，将 $< D, R >$ 除以 G，余数非零：检测到错误！
 - 可以检测所有小于 $r+1$ 位突发错误
 - 在实践中广泛应用(Ethernet, 802.11 WiFi)

发送方想要计算 R，使得：

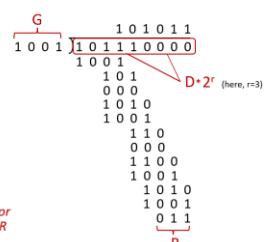
$$D \cdot 2^r \text{ XOR } R = nG$$

等价于(在等式两边XOR R):

$$D \cdot 2^r = nG \text{ XOR } R$$

如果我们用 G 来除 $D \cdot 2^r$ ，
余数值刚好为要求的 R:

$$R = \text{remainder } [D \cdot 2^r] \text{ algorithm for computing R}$$



多路访问协议：

多路访问链路：两种“链路”——

点对点，以太网交换机、主机之间的点对点链路，用于拨号访问的 PPP

广播 (共享线路或媒体): 传统以太网，HFC 上行链路，802.11 WLAN，4G/5G，卫星网络

多路访问协议：确定节点如何共享信道的分布式算法，即：确定节点何时可以传输。关于信道共享的通信必须使用信道本身！没有用于协调的带外(out of band)信道

单个共享广播信道，两个或多个节点同时传输：冲突。

冲突(collision)：如果节点同时接收到两个或两个以上的信号。

一个理想化的多路访问协议：假设：速率 R bps 的多路访问信道 (multiple access channel, MAC) 想要实现的目标：1.当一个节点想要发送时，它可以以速率 R bps 发送。2.当 M 个节点需要发送时，每个节点可以以平均速率 R/M bps 发送。3.完全去中心化：不需要特殊的节点来协调传输，不需要时钟和时隙的同步。4. 简单的。

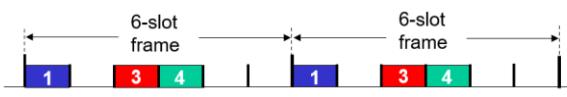
MAC 协议分类：三大类——

信道划分协议：将信道分成更小的“片段”(时隙、频率、编码)，分配“片段”给节点独占使用。
随机接入协议：不划分信道，允许冲突，能从冲突中“恢复”。“轮流”协议：通过轮流访问信道避免冲突，但是有更多数据传输的节点可以获得较长的信道使用权。

信道划分 MAC 协议：

TDMA: time division multiple access 时分复用，轮流访问信道。每个站点在每轮中获得固定长度的时隙(长度=分组传输时间)。未使用的时隙变为空闲。

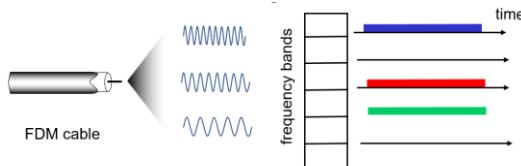
例子：6 站点局域网，1,3,4 有分组要传输，时隙 2,5,6 空闲：



TDMA:不同的人在不同的时刻讲话

FDMA: frequency division multiple access 频分复用，信道的有效频率范围被分成一个个小的频段，每个站点被分配一个固定的频段，频段内未使用的传输时间变为空闲

例子：6 站点局域网，1,3,4 有分组要传输，频段 2,5,6 空闲：



FDMA:不同的组在不同的小房间里通信

CDMA: code division multiple access 码分复用，所有站点在整个频段上同时进行传输，每个节点分配一种不同的编码，完全无冲突，抗干扰性强，广泛应用于蜂窝电话中

CDMA:不同的人使用不同的语言讲话

随机接入协议：当节点有分组要发送，以全信道数据速率 R 传输，节点之间没有先验的协调，两个或多个传输节点：“collision” 冲突。随机接入协议 规定：如何检测冲突，如何从冲突中恢复(e.g.,通过延迟重传)

随机接入 MAC 协议的例子：ALOHA, slotted ALOHA, CSMA, CSMA/CD, CSMA/CA

时隙 ALOHA：假设：所有帧都是等长的，时间分成大小相等的时隙(传输 1 帧的时间)，节点只在时隙开始时传输，节点在时钟上是同步的，如果时隙内有 2 个及以上节点传输，则所有节点检测到冲突。运行：当节点获得新帧时，在下一个时隙发送，如果没有冲突：节点能够在下一时隙发送新帧，如果有冲突：节点在随后每一个时隙以概率 p 重传帧，直到成功。

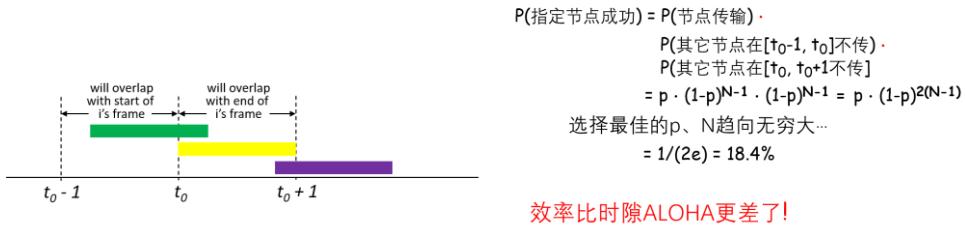
优点：单个活跃节点可以以信道的全速率连续传输，高度去中心化：仅需要节点之间在时隙上的同步，简单

缺点：冲突，浪费时隙，空闲时隙，节点检测冲突的时间可能小于帧传输的时间，时钟同步



纯 ALOHA: 无时隙 ALOHA: 更简单, 无需节点间时间同步, 当帧第一次到达时立即传输。当没有时间同步时, 冲突概率增加: 在 t_0 时刻传输的帧与其它在 $[t_0-1, t_0+1]$ 时间段内传输的帧产生冲突。

纯 ALOHA 效率

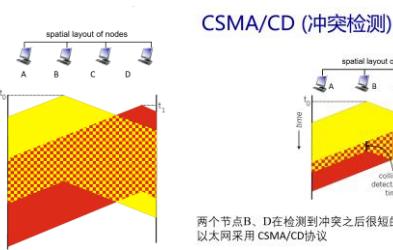


CSMA(载波监听多路访问): CSMA: carrier sense multiple access 在传输前先侦听信道: 如果侦听到信道空闲, 传送整个帧, 如果侦听到信道忙, 推迟传送

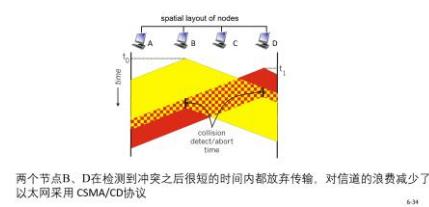
人类活动类比: 不要打断别人正在进行的说话

CSMA冲突

冲突仍然可能发生:
由传播延迟造成: 两个节点可能侦听不到正在进行的传输
冲突:
整个冲突帧的传输时间都被浪费了, 是无效的传输(红黄区域)
注意:
传播延迟(距离)决定了冲突的概率



CSMA/CD (冲突检测)



CSMA/CD (冲突检测): CSMA/CD: collision detection 可以在短时间内检测到冲突。冲突发生时则传输终止, 减少对信道的浪费。冲突检测: 有线局域网中容易实现: 检测信号强度, 比较传输与接收到的信号是否相同; 在无线局域网中实现困难: 自身信号远大于其他节点信号(CSMA/CA collision avoidance, IEEE 802.11 MAC 协议)

人类类比: 礼貌的对话, 说话之前先听, 如果与他人同时开始说话, 停止说话

以太网 CSMA/CD 算法:

1. 适配器从网络层获得数据报, 创建帧。
2. 适配器发送前侦听信道 CS : 1) 闲: 开始传送帧 2) 忙: 一直等到闲再发送。
3. 发送过程中冲突检测 CD: 1) 没有冲突:成功 2) 检测到冲突:放弃,之后尝试重发。
4. 发送方适配器检测到冲突, 除放弃发送外, 还发送一个 Jam 信号使所有站点都知道冲突。
5. 放弃传输后, 适配器进入指数退避状态, 在第 m 次失败后, 适配器从 $\{0, 1, 2, \dots, 2m-1\}$ 中随机选择一个 K 值, 等待 $K \times 512$ 位时 (发送 512bit 进入以太网所需时间量), 然后转到步骤 2。一个帧经历的冲突越多, K 选择的范围越大, binary exponential backoff 二进制指数退避算法。

效率计算: 定义: 当有大量的活动节点, 而每个节点有大量帧要发送时, 帧在信道无碰撞地传输的那部分时间在长期运行时份额。

t_{prop} = 信号在任意两个节点之间传播所需要的最大时延 t_{trans} = 传输一个最大帧的时间

$$\text{efficiency} = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

当 t_{prop} 接近 0 时, 效率接近 1, 当 t_{trans} 变得很大时, 效率也接近于 1

比 ALOHA 更好的性能, 而且简单, 廉价, 去中心化。

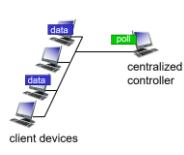
轮流 MAC 协议: 寻找两全其美的方案!

信道划分 MAC 协议: 在高负载下有效和公平地共享信道。低负载时效率低下: 信道访问延迟, 即使只有 1 个活跃节点也只能分配 $1/N$ 带宽!

随机接入 MAC 协议: 低负载高效:单节点充分利用信道, 高负载: 冲突开销。

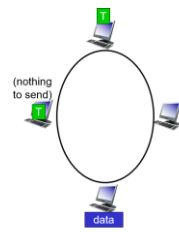
Polling 轮询:

- 中央控制器依次“邀请”其他节点进行传输
- 通常用于“dumb”设备
- 缺点:
 - 轮询开销
 - 延迟
 - 单点故障(主节点)
 - 蓝牙使用轮询



令牌传递 token passing:

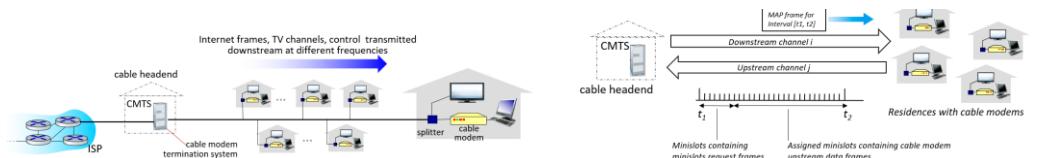
- 控制令牌报文按顺序从一个节点传递到下一个节点
 - 持有令牌时传输
- 缺点:
 - 令牌开销
 - 延迟
 - 单点故障(令牌)



电缆接入网络: FDM, TDM 和 随机接入

多个下行 (广播) FDM 信道: 高达 1.6 Gbps/channel: 单个 CMTS 传输信道。多个上行信道 (高达 1 Gbps/channel): 多路访问: 所有用户竞争 (随机访问) 某些上游信道时隙; TDM

DOCSIS: data over cable service interface specification 有线电缆数据服务接口规范。使用 FDM 将下行和上行网络段划分为多个频率信道。TDM 上行网络段: 一些时隙被分配, 一些有竞争: 下行信道 MAP 帧: 分配上行信道时隙, 对上行信道的时隙(和数据)的请求在选定的时隙中使用随机接入 (二机制退避) 进行传输。



MAC 协议总结: 信道划分: 按照时间、频率或编码, Time Division 时分复用, Frequency Division 频分复用。**随机接入 (动态):** ALOHA, S-ALOHA, CSMA, CSMA/CD, 载波侦听: 在有线介质上容易实现, 在无线介质上较难实现, CSMA/CD 在以太网中使用, CSMA/CA 在 802.11 WLAN 中使用。**轮流:** 由一个中心节点轮询, 令牌传递, 蓝牙, FDDI, 令牌环。

局域网:

MAC (or LAN or physical or Ethernet) 地址:

32 位 IP 地址: 接口的网络层地址, 用于网络层转发 e.g.: 128.119.40.136。

48 位 MAC 地址 (对于多数局域网) 固化在 NIC ROM, 也可以通过软件设置。

功能: 从一个接口获取帧传递到另一个物理连接的接口 (同一子网, 在 IP 寻址意义上)

e.g: 1A-2F-BB-76-09-AD (十六进制表示)。LAN 中的每个接口有唯一的 48 位 MAC 地址, 有一个本地唯一的 32 位 IP 地址。

MAC 地址分配由 IEEE 管理, 制造商购买部分 MAC 地址空间 (以确保唯一性)

类比: MAC 地址: 就像社会保险号, IP 地址: 就像邮政地址

MAC 平面地址: 可移动性, 可以将接口从一个局域网移动到另一个局域网, 回顾 IP 地址不可移动: 依赖于节点所连接的 IP 子网。

ARP: address resolution protocol 地址解析协议:

Q: 已知 IP 地址, 如何确定接口的 MAC 地址?

ARP 表: LAN 中每个 IP 节点(主机, 路由器) 都有 ARP 表

一些局域网节点的 IP/MAC 地址映射: < IP address; MAC address; TTL>

TTL (Time To Live): 地址映射失效的时间 (典型 20 min)

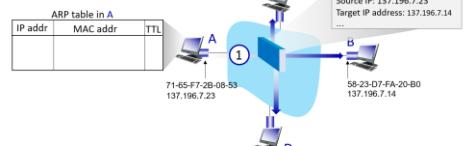
例子: A 想要发送数据报到 B

• B 的 MAC 地址不在 A 的 ARP 表, 所以 A 使用 ARP 来寻找 B 的 MAC 地址

A 广播 ARP 请求, 包含 B 的 IP 地址

① destination MAC address = FF-FF-FF-FF-FF-FF

• LAN 上所有节点都会收到 ARP 请求



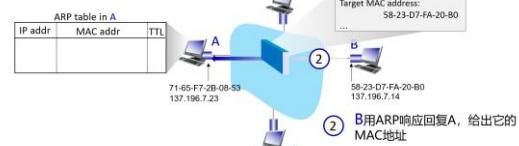
例子: A 想要发送数据报到 B

• B 的 MAC 地址不在 A 的 ARP 表, 所以 A 使用 ARP 来寻找 B 的 MAC 地址

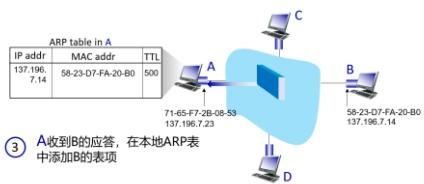
ARP message into Ethernet frame (sent to 71-65-F7-2B-08-53)

Target IP Address: 137.196.7.14

Target MAC Address: 58-23-D7-FA-20-B0



例子: A 想要发送数据报到 B
• B的MAC地址不在A的ARP表。所以A使用ARP来寻找B的MAC地址

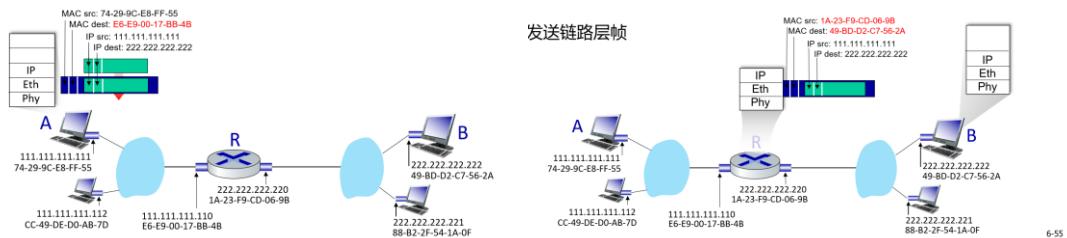


ARP 协议示例 (上面三幅图)

路由到另一个子网: 寻址

预排: 通过 R 将数据报从 A 发送到 B。关注寻址——在 IP 层(数据报)和 MAC 层(帧)级别

假设: A 知道 B 的 IP 地址, A 知道第一跳路由器 R 的 IP 地址(怎么知道?), A 知道 R 的 MAC 地址(怎么知道?)。A 创建 IP 数据报, 其中源 IP 为 A, 目的 IP 为 B。A 创建包含 A 到 B 的 IP 数据报的链路层帧, R 的 MAC 地址是帧的目的地。帧从 A 发送到 R。帧在 R 处被收到, 数据报被提取并传到 IP 层。R 确定出接口, 以 IP 源地址 A、目的地址 B 向链路层传递数据报。R 创建包含 A 到 B 的 IP 数据报的链路层帧。帧的目的地址: B 的 MAC 地址。发送链路层帧。B 接收帧, 提取目的地址为 B 的 IP 数据报。B 通过协议栈将数据报传递到 IP 层。



Ethernet 以太网: “主流”有线局域网技术: 第一个广泛使用的局域网技术, 简单, 廉价, 带宽不断提升: 10 Mbps – 400 Gbps , 单芯片, 多速度(e.g., Broadcom BCM5761)。

以太网的物理拓扑: 总线: 在 90 年代中期流行: 所有节点在同一冲突域中(彼此相互冲突)。
星型: 目前主流: 链路层交换机在中心, 每个节点以及相连的交换机端口使用 (独立的) 以太网协议 (不会和其他节点的发送产生冲突)

以太网的帧结构: 发送接口将 IP 数据报 (或其他网络层协议分组) 封装在以太网帧中。

| type | | | | |
|----------|---------------|----------------|----------------|-----|
| preamble | dest. address | source address | data (payload) | CRC |
| | | | | |

前导码: 用于同步接收方、发送方时钟速率, 7 bytes 的 10101010 后面跟着 1 byte 的 10101011。**地址:** 6 byte 源、目的 MAC 地址: 如果适配器接收到具有匹配的目的地址或广播地址的帧 (例如, ARP 包), 它将帧中的数据传递给网络层协议。否则, 适配器丢弃帧。

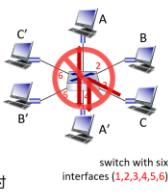
类型: 表明高层协议: 主要是 IP, 也支持其它网络层协议, e.g., Novell IPX, AppleTalk, 用于接收端上解复用。**CRC:** 接收端循环冗余校验, 检测到差错: 帧被丢弃

以太网是不可靠无连接的: 无连接: 发送方网卡和接收方网卡之间没有握手。不可靠: 接收方网卡不向发送方网卡发送 ACK 或 NAK。丢失的帧中的数据只有在初始发送方使用更高层的 rdt (如 TCP) 时才能恢复, 否则丢失的数据将彻底丢失。以太网 MAC 协议: 无时隙的二进制退避 CSMA/CD

以太网交换机: 交换机是链路层设备: 扮演主动角色, 存储、转发以太网 (或其他类型) 帧。检查到来的帧的 MAC 地址, 有选择地将帧转发到一个或多个传出链路, 当帧需要向某个(些) 网段进行转发, 需要使用 CSMA/CD 进行接入控制。透明的: 主机不知道交换机的存在: 即插即用, 自学习。交换机不需要配置

交换机：多路同时传输

- 主机与交换机有专用直接的连接
- 交换机缓存帧
- 在每个帧进入的链路上使用以太网协议。因此：
 - 没有冲突；全双工
 - 每条链路都是一个独立的冲突域
- 交换:** A-to-A' 和 B-to-B' 可以同时传输，没有冲突
但是 A-to-A' 和 C to A' 的传输不能同时发生



交换机：帧过滤/转发

在交换机接收到帧时：

```

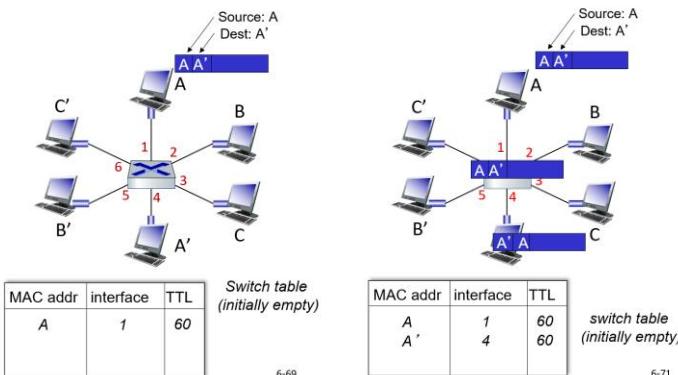
1. 记录输入链路，发送主机的MAC地址
2. 使用目标MAC地址对交换表进行索引
3. if entry found for destination
   then {
     if destination on segment from which frame arrived
     then drop frame
     else forward frame on interface indicated by entry
   }
  else flood /* forward on all interfaces except arriving
  interface */
  
```

Q: 交换机如何知道 A' 可通过接口 4 到达，B' 可通过接口 5 到达？

A: 每个交换机都有交换表，每个表项：(MAC address of host, interface to reach host, time stamp) 就像路由表

Q: 如何在交换表中创建和维护表项？像路由协议的方案？

A: 自学习。交换机通过学习得到哪些主机（mac 地址）可以通过哪些端口到达。当接收到帧时，交换机“学习”发送者的位置：传入局域网网段。记录发送方 MAC 地址/进入端口映射关系在交换表中。 帧的目的地位置未知：泛洪。 已知：选择性地只发送一个链路。



交换机 vs 路由器：

都是存储转发：路由器：网络层设备（检查网络层头部） 交换机：链路层设备（检查链路层头部）
都有转发表：路由器：使用路由算法计算得出路由表，IP 地址。交换机：通过泛洪、自学习来生成转发表，MAC 地址

虚拟局域网：

虚拟局域网(Virtual LANs, VLANs): 动机

Q: 随着局域网规模的扩大，用户改变了连接点，会发生什么？

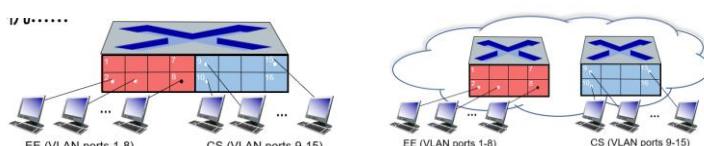
单个广播域：

- 可扩展性：所有第二层广播流量（ARP、DHCP、未知 MAC 地址而进行广播的帧）必须穿越整个局域网
- 效率、安全、隐私问题

行政管理问题：

- CS 用户将办公室移动到 EE -- 物理上连接到 EE 交换机，但希望在逻辑上保持连接到 CS 交换机

基于端口的虚拟局域网：交换机端口分组（由交换机管理软件），使单个物理交换机作为多个虚拟交换机运行。



流量隔离：从/到端口 1-8 之间的帧只会涉及到端口 1-8，还可以根据端点的 MAC 地址定义

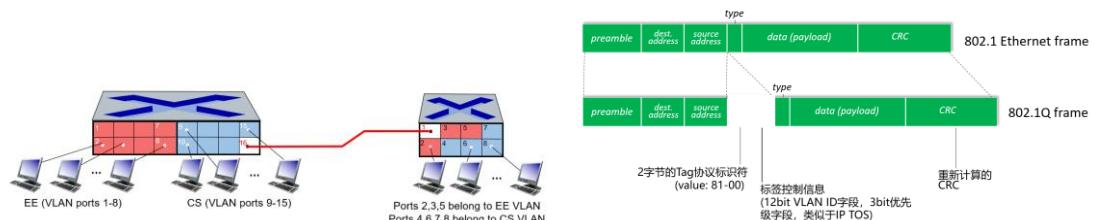
VLAN，而不是交换机端口。

动态成员：端口可以被动态地分配给不同的 VLANs

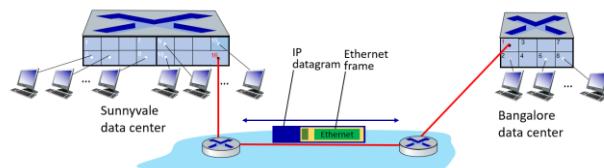
在 VLANs 间转发：通过路由完成（就像单独的交换机一样），实际生产中，设备生产商可以提供集成交换机和路由器功能的设备

跨多台交换机的虚拟以太网：干线端口(trunk port): 在多个物理交换机上定义的 VLANs 之间传输帧。在一个 VLAN 内不同交换机之间转发的帧不能是普通的 802.1 帧（必须携带 VLAN ID 信息），802.1q 协议为干线端口之间转发的帧增加/删除附加的报头字段

802.1Q VLAN 帧格式



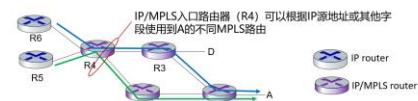
EVPN: Ethernet VPNs (aka VXLANS): 第 2 层以太网交换机逻辑上相互连接（例如，使用 IP 作为 underlay）。在站点之间的 IP 数据报中携带的以太网帧，“隧道方案将第二层网络覆盖在第三层网络之上…在现有的网络基础设施上运行，并提供了一种“扩展”第二层网络的方法”



链路虚拟化: MPLS:

多协议标签交换 (Multiprotocol label switching, MPLS): 目标：使用固定长度标签（而不是使用地址，采用最长前缀匹配）在支持 MPLS 的路由器之间进行高速 IP 转发，使用固定长度标识符来更快地查找，借鉴了虚电路(virtual circuit, VC)的思想，但是 IP 数据报仍然保留 IP 地址！具有 MPLS 能力的路由器亦称为标签交换路由器。仅根据标签值向出接口转发分组（不检查 IP 地址）。MPLS 转发表不同于 IP 转发表，灵活性：MPLS 转发决策可以不同于 IP 转发决策，使用目的地址和源地址将流以不同路径路由到同一目的地（流量工程），链路故障时快速重路由：预先计算好的备份路径

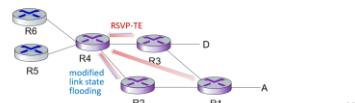
MPLS vs IP path:



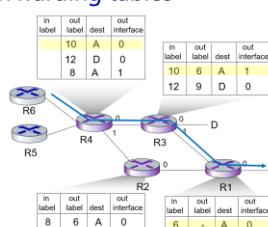
- **IP路由：**到目的地址的路径仅由目的地址决定
- **MPLS路由：**可根据源地址和目的地址选择到达目的地址的路径
 - 广义转发风格 (MPLS 10年前)
 - 快速重新路由：以防链路故障，提前计算好备份路由

MPLS 信号

- 修改OSPF、IS-IS链路状态泛洪协议，使其携带MPLS路由使用的信息：
 - e.g.,链路带宽，预留链路带宽的量
- 入口MPLS路由器使用RSVP-TE信令协议在下行路由器上建立MPLS转发表



MPLS forwarding tables

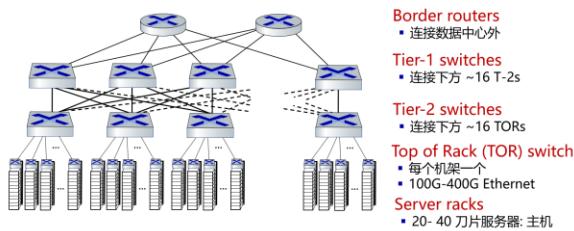


数据中心网络：

数万-数十万台主机构成 DC 网络，密集耦合、距离临近：电子商务 (e.g. Amazon), 内容服务器 (e.g., YouTube, Akamai, Apple, Microsoft), 搜索引擎, 数据挖掘 (e.g., Google)

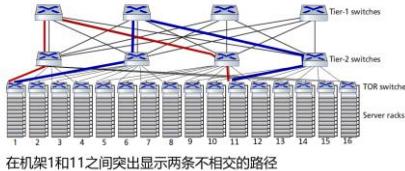
挑战：多个应用程序，每个应用程序服务大量客户端，可靠性，管理/负载均衡，避免处理，联网，数据瓶颈

数据中心网络：网络元素



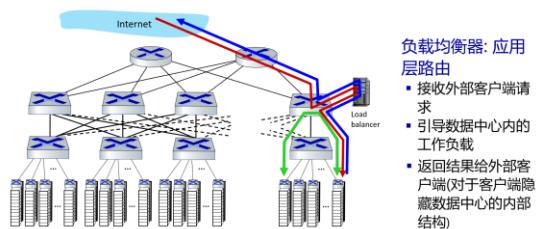
数据中心网络：多路径

- 交换机、机架间互连丰富：
 - 增加机架之间的吞吐量（可能有多条路由路径）
 - 通过冗余提高可靠性



6-96

数据中心网络：应用层路由



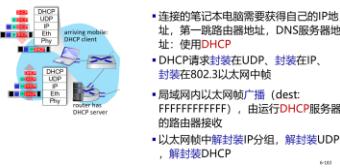
数据中心网络：协议创新

- **链路层:**
 - RoCE: remote DMA (RDMA) over Converged Ethernet
- **传输层:**
 - 显式拥塞通知 (explicit congestion notification, ECN) 用于传输层拥塞控制 (DCTCP, DCQCN)
 - 逐跳 (反压, backpressure) 拥塞控制的相关试验
- **路由, 管理:**
 - SDN在组织的数据中心内广泛使用
 - 将相关的服务和数据尽可能地放在一起 (例如, 在同一个机架或附近的机架中)，以尽量减少tier-2和tier-1的通信

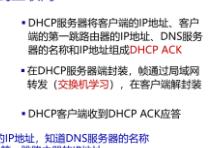
ORION: Google's new SDN control plane for internal datacenter (Jupiter) + wide area (B4) network: 路由 (域内, iBGP), 流量工程: 在 ORION 核心之上的应用程序中实现, edge-edge 基于流的控制 (e.g., CoFlow scheduling) 来满足签约的服务等级协议。管理: pub-sub 分布式微服务位于 ORION 核心, OpenFlow 用于交换机信令/监控。

总和：WEB 界面请求：

日常场景：连接到互联网



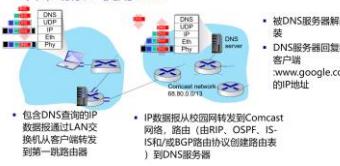
日常场景：连接到互联网



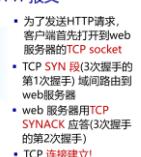
日常场景：ARP (DNS之前, HTTP之前)



日常场景：使用DNS



日常场景：TCP连接携带HTTP报文



日常场景：HTTP请求和应答

