

计算机网络

计算机网络

一、计算机网络和Internet

计网定义

具体构成

服务描述

protocol

Internet 标准

网络边缘

接入网络，物理介质

网络核心

电路交换

分组交换

分组传输时延 (Transmission Delay)：

统计复用

处理拥塞

分组交换网络中的延迟、丢失和吞吐量

流量强度

网络吞吐量bps

对比

带宽表示网络中单位时间内能够传输的数据量

TCP&UDP

协议层 服务模型

逻辑通信

Internet 协议栈

实体、协议、服务和服务访问点

1.7 攻击威胁下的网络

1、植入恶意软件

病毒&蠕虫

2. 攻击服务器和网络基础设施

3. 嗅探分组

4 .伪装

5.修改或删除报文

二、应用层

网络应用程序体系结构

进程与计算机网络的接口-套接字

因特网运输协议提供的服务：TCP&UDP

因特网应用：应用层协议，传输协议

Web应用和HTTP协议

HTTP: 超文本传输协议

HTTP连接

非持久HTTP连接

往返时间RTT

非持久HTTP响应时间：

TCP连接的“三次握手”过程

持久HTTP连接

非流水线vs流水线

非持久连接VS持久连接

HTTP请求页面时间分析

HTTP报文结构

常用方法

响应状态码

HTTP/2

HTTP3的改进

HTTP1、2、3

Cookie

包括四部分

Web缓存

条件GET方法

原理

过程

因特网中的电子邮件

电子邮件

发送报文过程

SMTP

邮件访问协议

SMTP vs HTTP

POP3

辨析

DNS:英特网的目录服务

主要功能

分层架构

查询方式

查询方式辨析

域名解析过程

缓存

DNS攻击

1、DNS劫持

2、缓存投毒

3、DDOS攻击 (Distributed Denial of Service)

4、DNS欺骗

内容分发网络CDN

为什么需要

流程

两种部署方式

提升速度、降低压力、增强稳定性

三、运输层

运输层服务

两个基本的传输层协议

多路复用和多路分解

端口

套接字

无连接多路分解

原理

例子

为什么说是无连接?

面向连接的多路分解

四元组标识

多路复用

多路复用和多路分解小结

无连接传输: UDP

特点

在 UDP 上实现可靠传输

UDP是面向报文的

封装过程

UDP：传输层动作

UDP 首部格式

UDP 校验和

伪首部

不一定检测出

UDP优点

可靠数据传输原理

实现机制

停止-等待协议SW

信道利用率

回退N帧协议GBN

选择重传协议SR

流水线协议：总结

面向连接的传输TCP

TCP报文段

处理失序报文段?

设置TCP超时值

TCP可靠数据传输机制

快速重传

TCP流量控制

持续计时器

TCP连接管理

两次握手建立连接

三次握手

四次挥手

拥塞控制原理

原因和代价

拥塞控制方法

端到端的拥塞控制

网络辅助拥塞控制

TCP拥塞控制

AIMD 探测带宽

发送逻辑

拥塞控制算法

TCP从慢启动、拥塞避免切换到快速恢复

瓶颈链路

基于延迟的 TCP 拥塞控制

显式拥塞通知

TCP 平均吞吐量

TCP公平性

四、网络层

虚电路和数据报网络

数据报网络

特点

路由器的工作原理

两个功能

结构

输入端口功能

转发表

最长前缀匹配

数据报分组的传输与转发表维护机制

交换结构

基本原理

三种交换结构

早期交换结构--经内存

 内存结构的速率瓶颈

经总线的交换结构

经交换矩阵的交换结构

排队（输入、输出端口）

 输入

 输出

缓冲

 缓冲区管理

路由器中哪些部分由硬件实现，哪些由软件实现？

基于目的地转发 vs 通用转发

网际协议：IPv4、寻址、IPv6及其他

因特网中的网络层协议

IP 数据报格式（IPv4）

 为什么TCP/IP在运输层/网络层都执行差错检测？

 校验和计算过程

IP地址

 编址方法

 1. 分类 IP 地址

 特殊IP地址

 重要特点

 互联网中IP地址

 2. 子网划分

 子网掩码

 默认子网掩码

$(IP\ 地址) \text{ AND } (\text{子网掩码}) = \text{网络地址}$

 子网

 子网划分

 基于子网的路由

 3. 无分类编址 CIDR

 主要特点

 路由聚合/构成超网

 定长&变长的子网掩码

获取IP地址

 DHCP协议（动态主机配置协议）

 工作过程

 组织机构如何获取IP 地址？

NAT: 网络地址转换

 NAT 的动机和基本原理

 实现

 工作机制

 优点

 限制

ICMP 因特网控制报文协议

 ICMP 报文类型及对应常见代码

 Traceroute 和 ICMP

 工作流程详解

IPv6

 头部

- 与IPv4比较
- 扩展首部
- 地址表示
- 从IPv4 过渡到 IPv6
 - 隧道
- 通用转发和SDN
 - 流表抽象
- 选路算法
 - 分类
 - LS链路状态路由算法
 - Dijkstra
 - 路由震荡?
 - 距离向量路由算法DV算法
 - 基于BF动态规划
 - LSvsDV
- 层次选路
 - OSPF
 - 优点
 - 层次 OSPF
 - Internet 域间选路: BGP
 - 路径属性 和 BGP 路由
 - BGP 路由选择
 - BGP 报文
 - 为什么AS内选路和AS间选路采用不同的协议 ?
- SDN
 - SDN体系结构的4个关键特征
 - SDN: 通用转发
 - SDN—数据平面交换机
 - SDN—控制器
 - SDN—网络控制应用程序
- 网络层: 总结
- 五、数据链路层
 - 概述
 - 链路层提供的服务
 - 链路层实现的位置
 - 接口通信
 - 差错检测和纠错
 - 三种主要差错检测技术
 - 比特奇偶校验
 - Internet校验和方法
 - 循环冗余检测
 - 模2运算
 - 计算R (CRC比特)
 - 循环冗余码CRC的特点
 - 差错检测方法比较
 - 多路访问链路和协议
 - 多路访问协议类型 (三类)
 - 1.1时分多路访问
 - 特点
 - 1.2频分多路访问FDMA
 - 1.3码分多路访问CDMA
 - 2.随机访问协议
 - ALOHA

- CSMA (载波侦听多路访问)
- 带冲突检测的CSMA(CSMA/CD)
- CSMA/CD协议工作流程
- 以太网CSMA/CD的运行机制
- CSMA/CD效率

3.轮流协议

- 轮询协议 (Polling)
- 令牌传递协议 (Token Passing)

总结

交换局域网

- 常见的网络拓扑结构
- 计算机与局域网的连接
- 局域网体系结构
- 链路层寻址和ARP
 - MAC地址分配
 - MAC地址识别
 - 节点的3种不同地址表示

ARP(Address Resolution Protocol): 地址解析协议

以太网(Ethernet)

- 以太网链路层控制技术
- 以太网的帧结构
- 以太网: 不可靠的无连接服务

链路层交换机

- 支持多节点同时传输
- 转发表
 - 自学习
 - 数据帧的过滤/转发
- 交换机互连
- 交换机的交换特点
 - 交换方式1.存储转发
 - 交换方式2.快速分组

三层交换机

- 交换机制
- 三层交换机的工作原理
- 交换机 vs. 路由器
- 交换机的优缺点
- 路由器的优缺点

VLAN

- 跨越多个交换机的 VLAN
- 链路虚拟化：网络作为链路层
- 多协议标签交换
 - 特性

一、计算机网络和Internet

计网定义

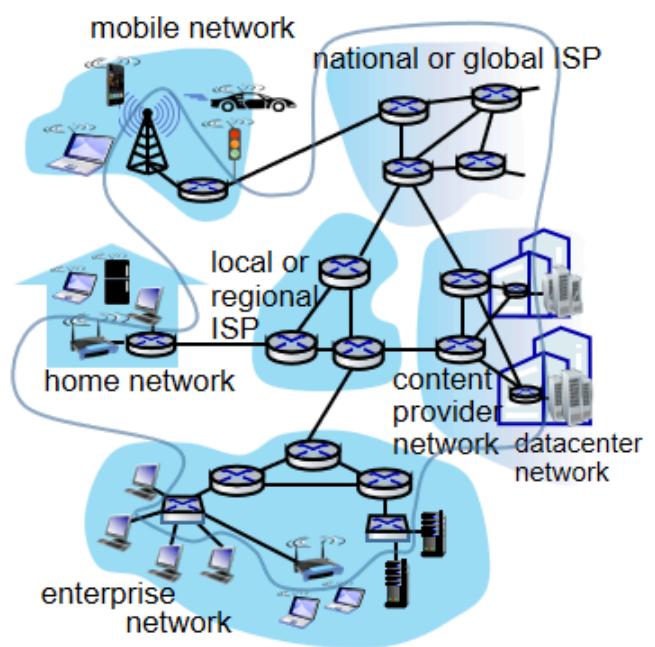
两台以上具有独立操作系统的计算机通过某些介质连接而成的相互共享软硬件资源的集合体

向用户提供的most important功能：连通 共享

具体构成

什么是Internet：具体构成描述

- 数以亿计的计算互连设备：
 - 主机 (host) = 端系统 (end system)
 - 运行网络应用程序
- 分组(packet)交换：
 - routers, switches
- 通信链路(link)：
 - fiber, copper, radio, satellite
 - 传输速率 = 带宽(bandwidth)
- 网络 (Networks)：
 - 设备、路由器、交换机和链路的集合，由机构组织管理



服务描述

什么是Internet：服务描述

- 提供**网络应用程序**的基础设施

允许终端系统上运行分布式应用程序，并彼此交换数据：

- Web, email, games, e-commerce, database, VOIP, P2P file sharing

- 为分布式应用程序提供**通信服务接口（套接字接口）**

- 无连接服务connectionless
- 面向连接服务connection-oriented
- 不提供数据传递时间保证（发送端到接收端）的服务

protocol

什么是协议？

协议：定义了两个或多个通信实体间所交换报文的**格式 (format)**和**次序 (order)**，以及在报文发送和/或接收或者其他事件方面所采取的**行动 (响应) (actions)**。

协议的基本要素：**语法、语义和同步**

人类的协议

询问“几点了？”
上课提问“我有个问题”

关键点：
... 发送出具体的消息
... 收到消息后具体的行动，
或者其他事件

网络协议

是计算机，而非人参与
Internet所有的通信活动都由协议掌控

Protocols define the **format, order of messages sent and received** among network entities, and **actions taken** on message transmission, receipt

语法：用来规定用户数据与控制信息的格式

语义：对具体事件应发出何种控制信息，完成何种动作以及作出何种应答

时序：对事件实现顺序的详细说明

- 分布式实体，交换消息(由协议管理)
- 随时间流逝
- 浏览协议的定义(显示格式、发送和接收消息的顺序以及所采取的操作)

Internet 标准

IETF

RFC

网络边缘

- 主机：客户机和服务器
- 服务器一般位于数据中心内

接入网络，物理介质

- 有线通信链路
- 无线通信链路

网络核心

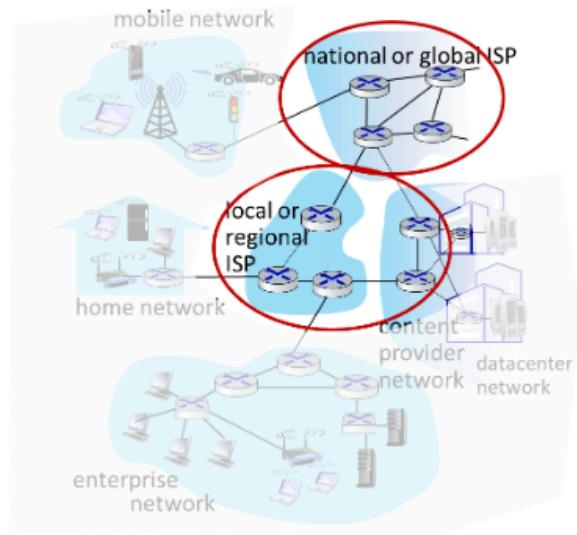
- 互联的路由器
- 网络组成的网络

网络核心部分

网络核心部分：相互连接的路由器构成的网络

在这个网络中传送数据基本原理：

- **电路交换**：每次会话预留沿其路径（线路）所需的独占资源——电话网
- **分组交换**：数据以离散的数据块通过网络来发送



电路交换

时分复用TDMA

电路交换举例

- 从主机A到主机B经一个电路交换网络需要多长时间发送一个640Kb的文件？

假设：

- 所有链路都是 1.536 Mbps
- 每个链路使用TDM划分成24个时隙
- 创建一条端到端的链路需要500msec

发送一个文件时间 = 创建电路时间 + 文件传输时间

文件传输时间：文件长/电路的传输速率

每条电路的传输速率：链路传输速率/时隙数

$$(1.536M \text{ b/s}) / 24 = 64\text{kb/s}$$

文件传输时间： $640\text{kb} / (64\text{kb/s}) = 10\text{s}$

A 到 B 的总发送时间： $0.5\text{s} + 10\text{s} = 10.5\text{s}$

频分复用FDMA

2. 分组交换

每个端到端的数据流被划分成分组

- 所有分组**共享**网络资源
- 每个分组**使用全部链路带宽**
- 资源**按需使用**



❖ 每个端到端的数据流被划分成分组

- 数据不会整体传送，而是被拆成一小块一小块的“**分组 (packet)**”
- 每个分组独立地走网络，不一定走相同的路径

❖ 所有分组**共享**网络资源

- 不像电路交换那样“预留”一条通路，而是**谁先来谁用**
- 所有用户共用同一条带宽，只要网络空闲就可以发数据

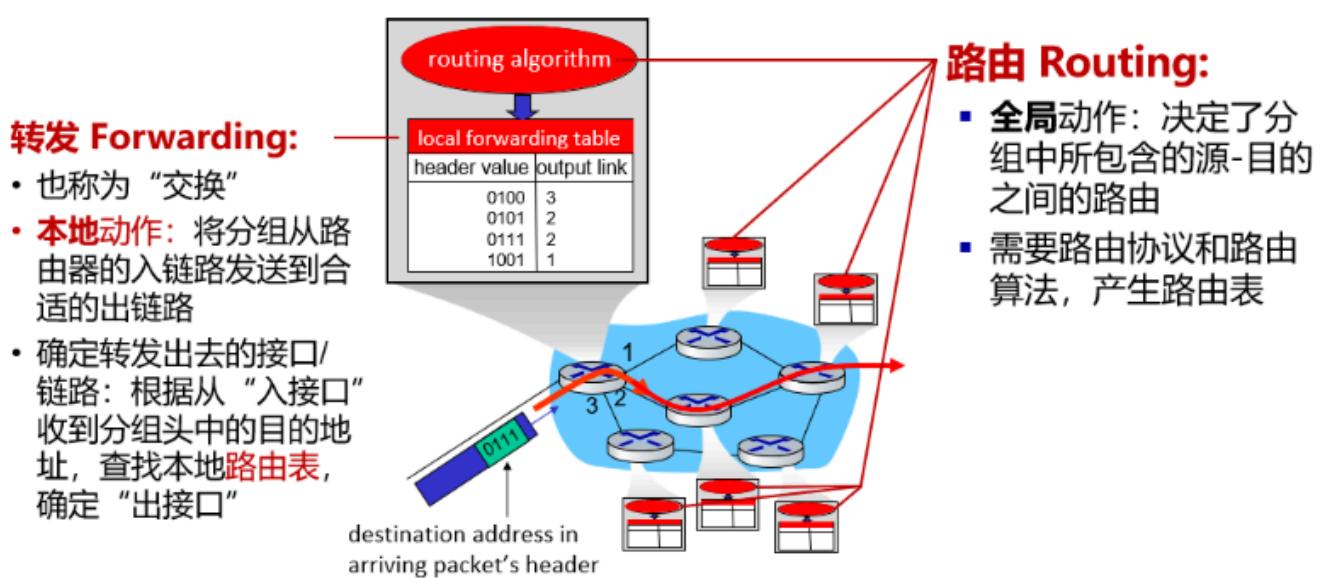
❖ 每个分组**使用全部链路带宽**

- 一旦轮到这个分组发送，它就可以用**整个链路的带宽**
- 没有“划分时隙”“频率通道”的限制 → 利用率高

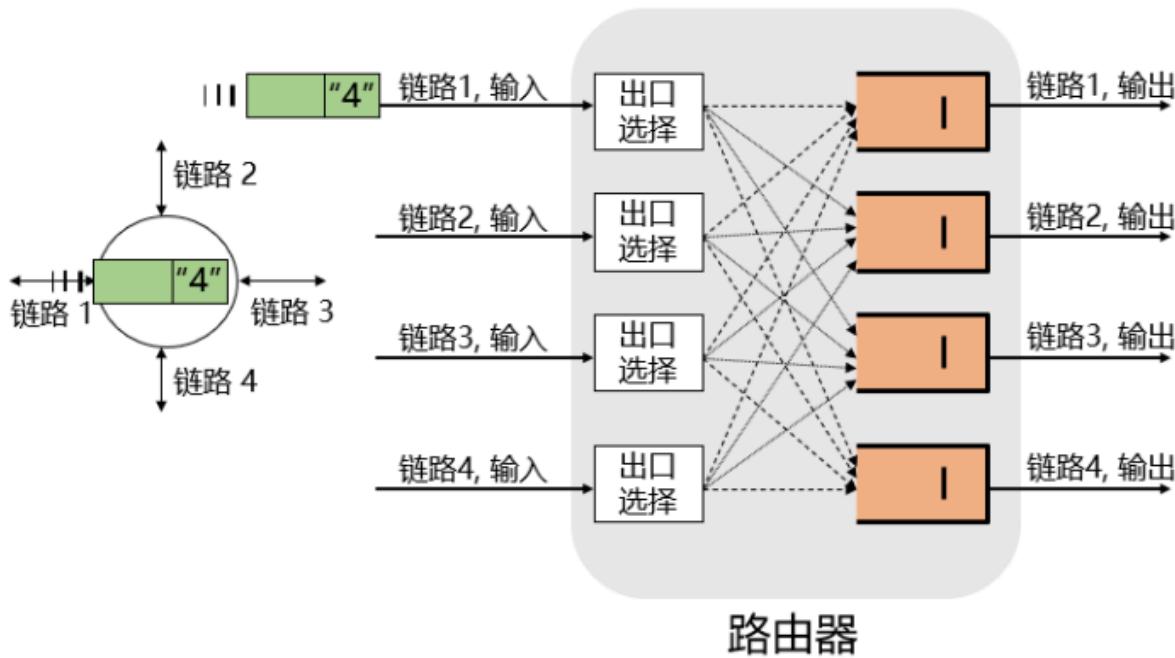
资源按需使用

- 网络资源不会“提前分配”
- **什么时候有数据，什么时候用资源**
- 空闲时资源可腾出让给其他用户，效率高

分组交换两个关键网络核心功能



路由器转发模型



总结一句话

分组交换的“路由”决定**怎么走**，“转发”负责**实际搬运**。

主机的发送流程：

1. 接收应用层消息

比如用户在浏览器中请求网页，主机先接收这个请求。

2. 划分为较小的数据块：分组（packets）

为了便于传输，数据被拆成多个小块，每块叫做一个“分组”或“包（packet）”，大小为 **L bits**。

3. 用链路的传输速率 **R** 发送到网络中

- 单位是 **R bits/sec**，表示每秒最多能发送多少位。
- 注意这个 **R** 就是我们常说的**带宽**，或叫**链路容量**。

分组传输时延（Transmission Delay）：

$$\text{传输时延} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

L 是要发送的分组长度（bit）

R 是链路的传输速率（bit/s） 带宽

单位最终是“秒（s）”

① 分组传输时延 (Transmission Delay)

- 即：将 L 比特的数据推出链路所需时间

$$\text{传输时延} = \frac{L}{R}$$

② 存储转发机制影响下的总时延

- 因为分组到达路由器后，必须等整个分组接收完毕后才开始转发
- 所以每跳的转发包含两个阶段的传输：
 - 发送端 \rightarrow 路由器: $\frac{L}{R}$
 - 路由器 \rightarrow 接收端: $\frac{L}{R}$

$$\text{端到端传输时延} = 2 \cdot \frac{L}{R}$$

(假设只有一个路由器，且传播时延为 0)

✓ 总结一句话：

统计复用 = 分组交换的核心机制，实现了：

- “不浪费带宽”
- “动态排队调度”
- “用户间公平共享链路资源”

✓ 拥塞现象如何处理？

□ 采用分组排队机制：

- 分组太多时，暂时排队等待
- 类似交通堵车，车只能在路口等红灯

📦 存储转发过程：

- 每经过一个路由器（“1站”），都要**完整接收**整个分组，才能转发
- 每跳都会**存储一次 → 再转发一次**

命题追踪 ► 分组交换网中各种时延的计算（2010、2013、2023）

- 发送时延，也称传输时延。节点将分组的所有比特推向链路所花的时间，即从发送

分组的第一个比特算起，到该分组的最后一个比特发送完毕所花的时间。

$$\text{发送时延} = \text{分组长度} / \text{发送速率}$$

- 传播时延。电磁波在信道（传输介质）中传播一定的距离所花的时间，即一个比特从链路的一端传播到另一端所需的时间。

$$\text{传播时延} = \text{信道长度} / \text{电磁波在信道上的传播速率}$$

分传输时延和传播时延。**传输时延是节点将分组推向网络所需的时间**，它取决于分组长度和发送速率。**传播时延是一个比特从一个节点传播至另一节点所需的时间**，它取决于两个节点之间距离和信道的传输介质，而与分组长度或发送速率无关。

- 处理时延。分组在交换节点为存储转发而进行的一些必要处理所花的时间。例如，分析分组的首部、差错检验或查找合适的路由等。
- 排队时延。分组在**路由器的输入队列或输出队列中排队等待**所花的时间。

因此，数据在网络中经历的总时延就是以上4部分时延之和：

$$\text{总时延} = \text{发送时延} + \text{传播时延} + \text{处理时延} + \text{排队时延}$$

在考试中，通常不用考虑处理时延和排队时延（除非另有说明）。

流量强度

- R = 链路带宽 (bps)
- L = 分组长度 (bits)
- a = 平均分组到达率
average packet arrival rate

流量强度：

$$\frac{\text{Traffic intensity}}{\text{intensity}} : \frac{\text{arrival rate of bits}}{\text{service rate of bits}} = \frac{La}{R}$$

越大，排队延迟越大

网络吞吐量bps

data packets per second

对比

对比项	分组交换	电路交换
传输方式	拆分成“分组”独立发送	先建立一条“电路”再传输
带宽资源使用	按需、共享	独占、预留
灵活性与效率	高：资源空闲时可供他人使用	低：即使空闲也被占用
是否必须先连接	否，直接发	是，需先建立连接
应用场景	现代互联网（IP、Web、微信等）	传统电话通信

比较分组交换与电路交换

➤ 电路交换的主要特点

- 电路交换通常采用**面向连接方式**
- **先呼叫建立连接**，实现端到端的**资源预留**
- **预留的资源包括**：链路带宽资源、交换机的交换能力
- 电路交换连接建立后，物理通路被通信双方**独占**，**资源专用**，即使空闲也不与其他连接共享
- 由于建立连接并预留资源，因此传输性能好；但如果传输中发生设备故障，则传输被中断

无法应对互联网中广泛存在的
“突发”（Burst）流量

➤ 分组交换 (packet switching)

- 将**大报文拆分成多个小分组**
- 通信双方以**分组**为单位、使用**存储-转发机制**，实现数据交互的通信方式
- **以分组作为数据传输单元**
- 每个分组的**首部都含有地址**（目的地址和源地址）等控制信息
- 每个分组在互联网中**独立地选择传输路径**
- 支持灵活的**统计多路复用**

带宽表示网络中单位时间内能够传输的数据量

✿ 带宽是“共享”还是“独占”？

✓ 两种情况都存在：

类型	说明
共享带宽	多个用户/设备共用一段带宽资源，互相影响
独占带宽	某用户/连接被分配固定带宽，其他人不影响

TCP&UDP

✓ 总结对比图：

对比项	TCP（面向连接）	UDP（无连接）
是否连接	三次握手建立连接	无连接，直接发送
是否可靠	是，丢包会重传	否，丢了就丢了
数据顺序	有顺序	没顺序
适合的应用	文件传输、网页、登录等	视频、语音、DNS等
控制机制	有流量控制 + 拥塞控制	无控制机制
效率与速度	慢但可靠	快但可能丢数据

1.2.3 面向连接的服务

目的: 在端系统间传送数据。

- 握手: 客户和服务器事先进入戒备状态, 为接下来的分组交换做好准备
 - 如同Hello, hello back 这样的人类电话通信
 - 建立“连接”, 在两个彼此通信的端系统之间
- TCP - Transmission Control Protocol
 - Internet 的面向连接的服务

TCP 服务 [RFC 793]

- 可靠、顺序、字节流传输:
 - 丢失: 确认和重传
- 流量控制:
 - 发送者不至于淹没接收者
- 拥塞控制:
 - 当网络拥塞时发送者降低发送速率

目的: 在端系统间传送数据。

- **UDP - User Datagram Protocol** [RFC 768]: Internet 无连接服务
 - 不可靠的数据传输
 - 无流量控制
 - 无拥塞控制

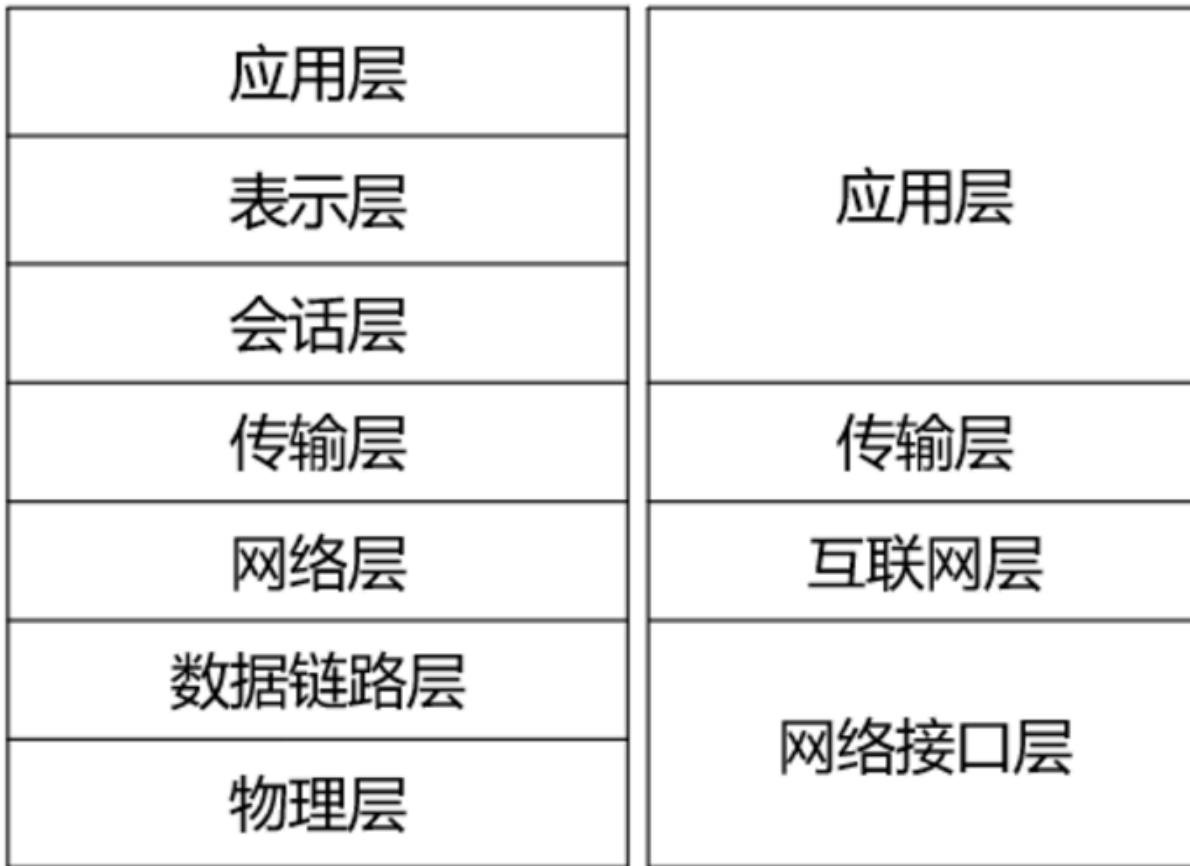
使用TCP的网络应用:

- **HTTP** (Web), **FTP** (file transfer), **Telnet** (remote login), **SMTP** (email)

使用UDP的网络应用:

- 流媒体, 视频会议, **DNS**, Internet电话

协议层 服务模型



OSI 7层模型



TCP/IP 4层模型

逻辑通信

分布式

各层：

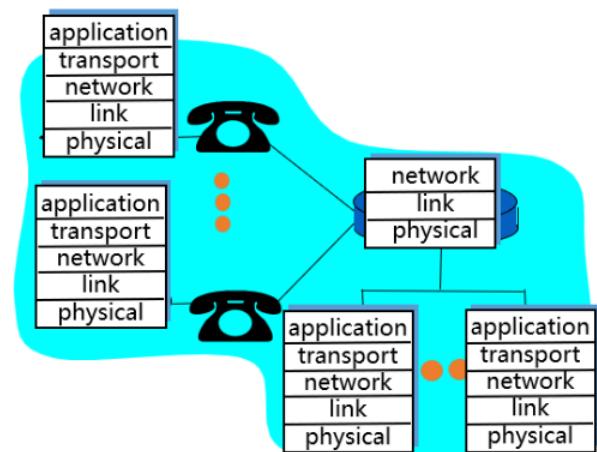
- 分布式
- 在各节点的网络实体(entities) 实现了各层的功能

主机实现5层功能，路由器和交换机实现2-3层功能。

- 网络**实体**完成功能动作，**对等实体**交换消息

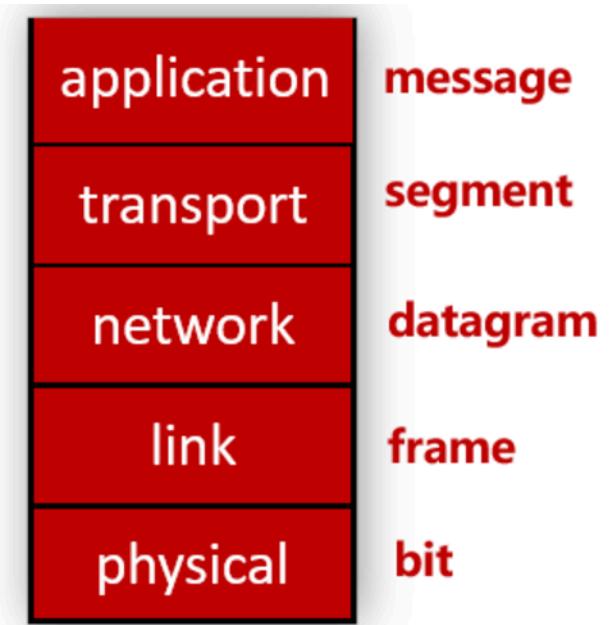
实体：定义自身功能的硬/软件的集合

对等实体：两台计算机上同一层所属的**程序、进程或实体**称为该层的**对等程序、对等进程或对等实体**。



Internet 协议栈

- **application:** 支持网络应用，报文传送
 - HTTP, IMAP, SMTP, DNS
- **transport:** 主机进程-进程之间数据传送
 - TCP, UDP
- **network:** 源-目的主机之间数据报路由
 - IP, routing protocols
- **link:** 邻居节点之间数据传递
 - Ethernet, 802.11 (WiFi), PPP
- **physical:** bits “on the wire”



因特网协议栈分为几个层次？分别是哪几个层次(按自顶向下的顺序描述)？简单描述每个层次的主要功能。

因特网协议栈分为 5 个层次，分别是：应用层、运输层、网络层、链路层和物理层。

每个层次主要实现的功能如下：

应用层：提供各种应用，传输**应用层报文**。

运输层：主机进程间**报文段**传送。

网络层：实现**数据报**主机到主机之间的运送服务

链路层：**相邻网络节点间**的数据帧传送。

物理层：**物理介质上**的**比特**传送。

实体、协议、服务和服务访问点

- **实体 (entity)**：表示任何可发送或接收信息的硬件或软件进程。
- **协议**：控制**两个对等实体**进行通信的规则的集合。
- 在协议的控制下，两个对等实体间的通信使得本层能够**向上一层提供服务**。
- 要实现本层协议，还需要使用**下层所提供的服务**。
- 在**同一系统中相邻两层**的实体进行交互（即交换信息）的地方，通常称为**服务访问点 SAP (Service Access Point)**。
- SAP 是一个抽象的概念，它实际上就是一个**逻辑接口**。

协议

其实现保证了能够向上一层提供服务。

对上面的服务用户是透明的。

是“水平的”

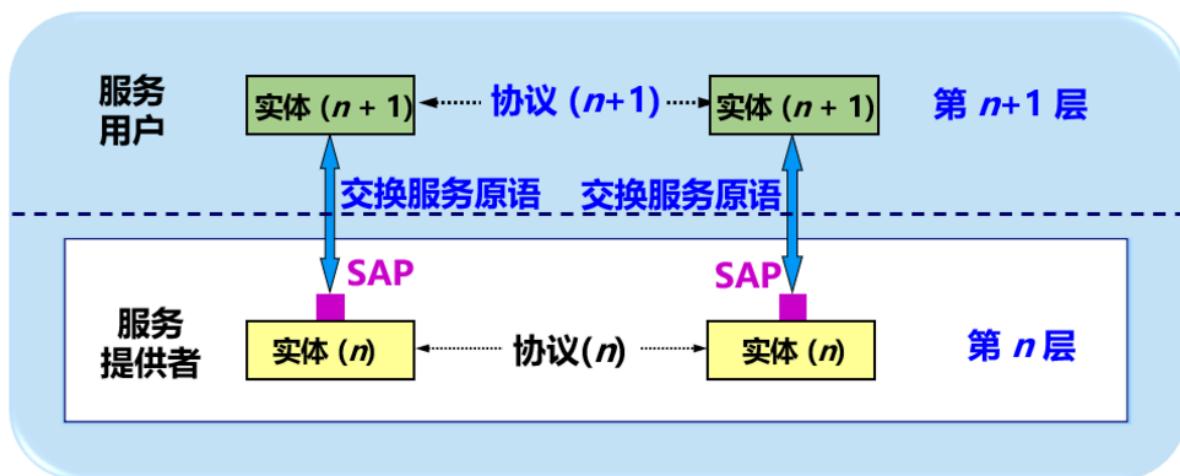
服务

上层使用服务原语获得下层所提供的服务。

上面的服务用户只能看见服务，无法看见下面的协议。

是“垂直的”

相邻两层之间的关系



1.7 攻击威胁下的网络

攻击方式：

- 植入恶意软件
- 攻击服务器和网络基础设施
- 嗅探分组
- 伪装
- 修改或删除报文

1、植入恶意软件

病毒&蠕虫

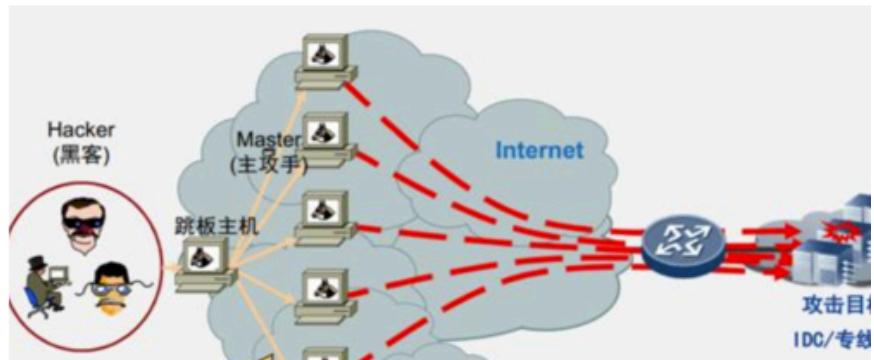
简单描述病毒和蠕虫之间的共同点和主要区别。最近爆发的勒索恶意软件如果细分，是属于病毒还是蠕虫？

病毒和蠕虫都是自我复制的，一旦感染上一台主机，就会寻求进入更多的主机。

主要区别是：病毒是一种需要某种形式的用户交互来感染用户设备的恶意软件。而蠕虫是一种无需任何明显用户交互就能进入设备的恶意软件。最近爆发的勒索恶意软件属于蠕虫类型的恶意软件。

2. 攻击服务器和网络基础设施

- **拒绝服务攻击 (Denial of Service , DoS)** : 攻击者通过海量的伪造分组使得合法用户无法正常使用资源（服务器、带宽）
- 三种类型
 - 弱点攻击
 - 带宽洪泛
 - 连接洪泛



3. 嗅探分组

- 分组嗅探器 (packet “sniffing”): 记录每个流经的分组拷贝的被动接收机
- 容易受到攻击的网络：无线网络和以太网 LAN



4 . 伪装

- 生成具有任意源地址、分组内容和目的地址的分组，然后将这个人工制作的分组传输到因特网中
- 将具有虚假源地址的分组注入因特网的能力称为IP哄骗(spoofing)
- 解决方法采用端点鉴别机制

5.修改或删除报文

- 中间人攻击 (man-in-middle attack)



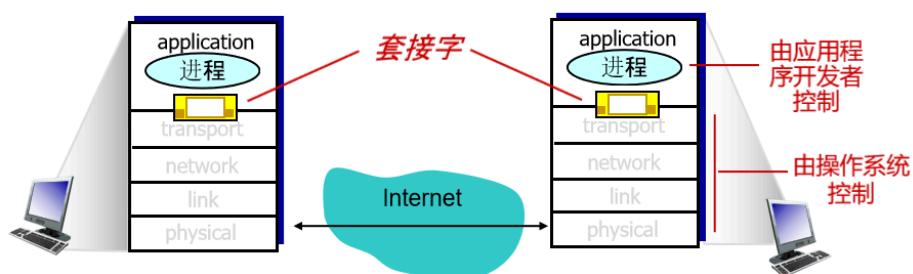
二、应用层

网络应用程序体系结构

进程与计算机网络的接口-套接字

进程与计算机网络的接口-套接字

- 套接字又叫做应用程序编程接口API
 - 用户通过API对传输层的控制仅限于：
 - 选择传输协议
 - 设定几个参数
- ◆ 进程通过它的套接字在网络上发送和接收报文
 - ◆ 套接字类比于门户
 - 发送进程把报文推出门户
 - 发送进程假定门户到另外一侧之间有运输设施，该设施可以传送报文到接收进程



因特网运输协议提供的服务：TCP&UDP

TCP服务

- **面向连接的服务**: 在客户机程序和服务器程序之间必须建立连接
- **可靠的传输服务**: 接收和发送进程间
- **流量控制**: 发送方不会淹没接收方
- **拥塞控制**: 网络出现拥塞时抑制发送进程
- **没有提供**: 时延保证, 最小带宽保证, 安全性

UDP 服务

- **不可靠数据传输**
- **没有提供**: 建立连接, 可靠性, 流量控制, 拥塞控制, 时延和带宽保证

因特网应用：应用层协议，传输协议

应用	应用层协议	下面的传输协议
电子邮件	SMTP [RFC 2821]	TCP
远程终端访问	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
文件传输	FTP [RFC 959]	TCP
流媒体	通常专用 (e.g. RealNetworks)	TCP or UDP
因特网电话	通常专用 (e.g., Skype)	典型用 UDP

Web应用和HTTP协议

常用术语

- 网页 (Web页, 或称文档) 由许多对象组成。
 - Web对象就是文件, 可以是HTML文件, JPEG图像, Java applet, 音频文件...
 - 多数网页由单个基本HTML文件和若干个所引用的对象构成
 - 每个对象被一个URL(Uniform Resource Locator, 统一资源定位符)寻址
协议类型://主机名:端口//路径和文件名
 - 举例URL: `http://www.someschool.edu:8000/someDept/pic.gif`


协议类型	主机名 (即服务器)	端口	路径名和文件
------	------------	----	--------

HTTP: 超文本传输协议

HIP概述

HTTP: 超文本传输协议

hypertext transfer protocol

- Web的应用层协议
 - client/server模式
 - client: 浏览器browser请求, 接收, “解释显示” Web对象
 - server: Web服务器响应请求, 发送 Web对象
 - HTTP 1.0: RFC 1945
 - HTTP 1.1: RFC 2616



HTTP连接

非持久HTTP连接

- 每个TCP连接上只传送一个对象，下载多个对象需要建立多个TCP连接
- **HTTP/1.0使用非持久**

持久HTTP连接

- 一个TCP连接上可以传送多个对象
- **HTTP/1.1默认使用持久HTTP连接**

HTTP连接

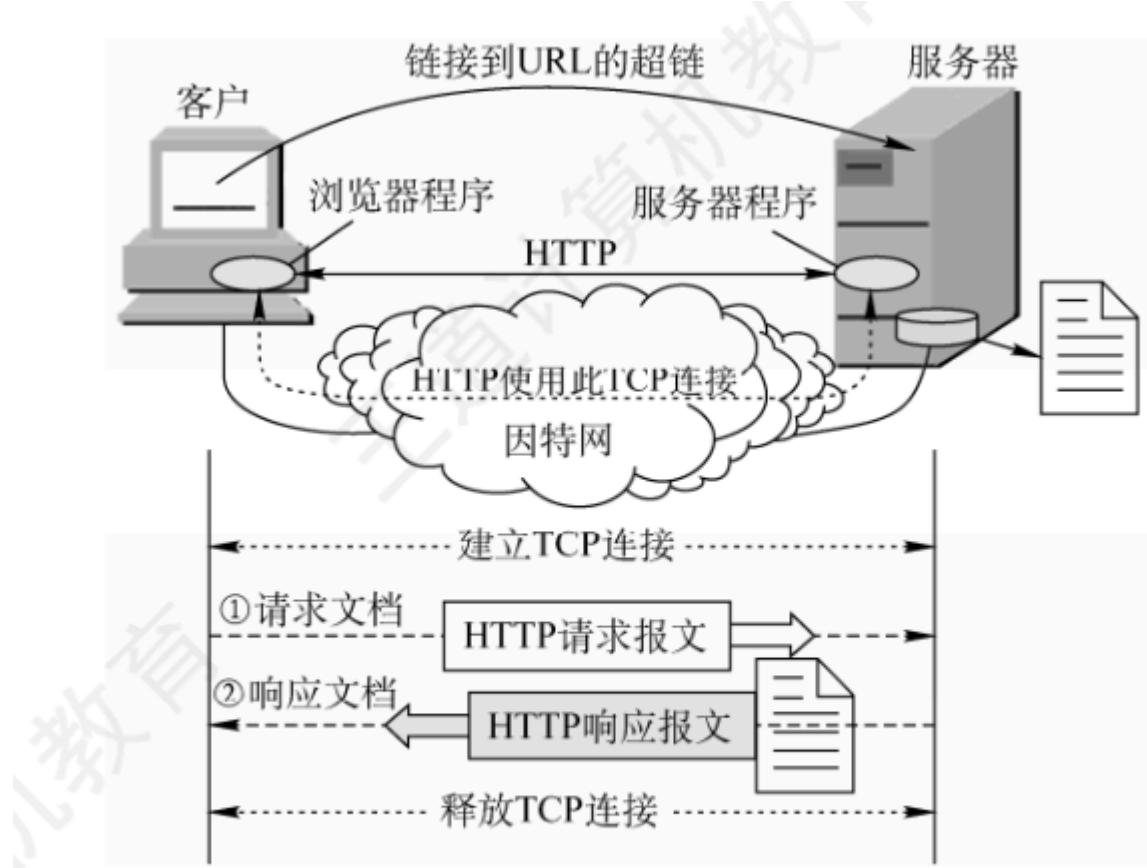


图 6.11 万维网的工作过程

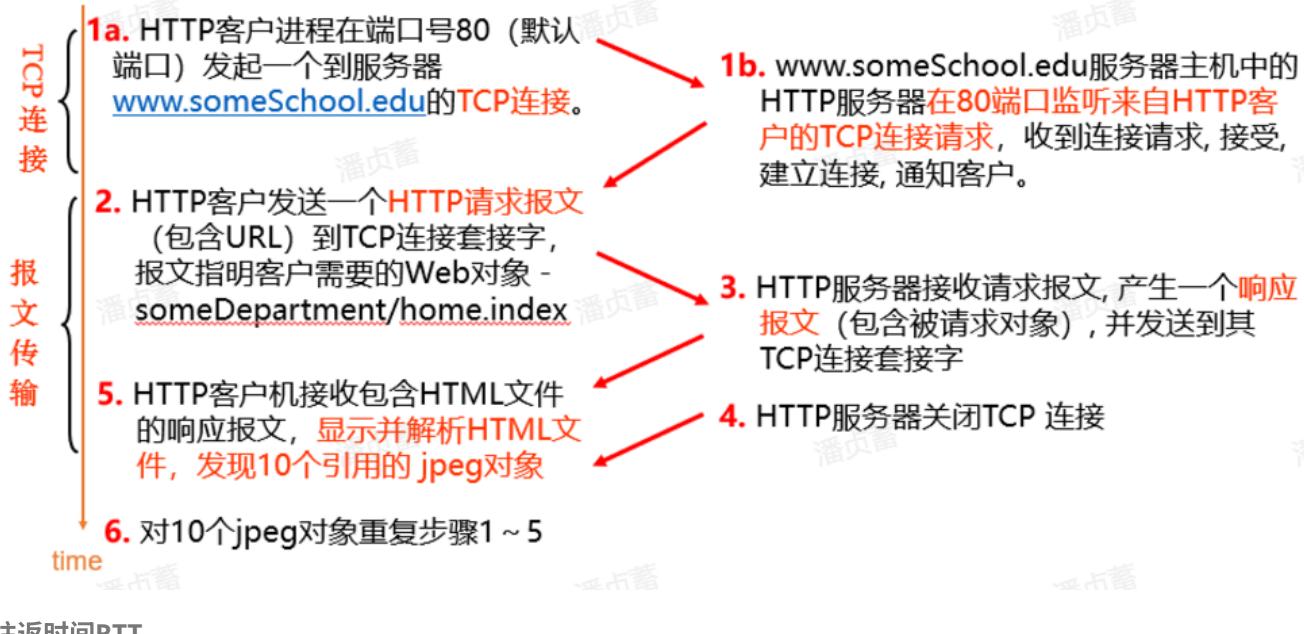
非持久HTTP连接

Web应用和HTTP协议

非持久HTTP连接

假设用户输入URL <http://www.someSchool.edu/someDepartment/home.index>,

网页由1个HTML文件, 和10个jpeg图像构成



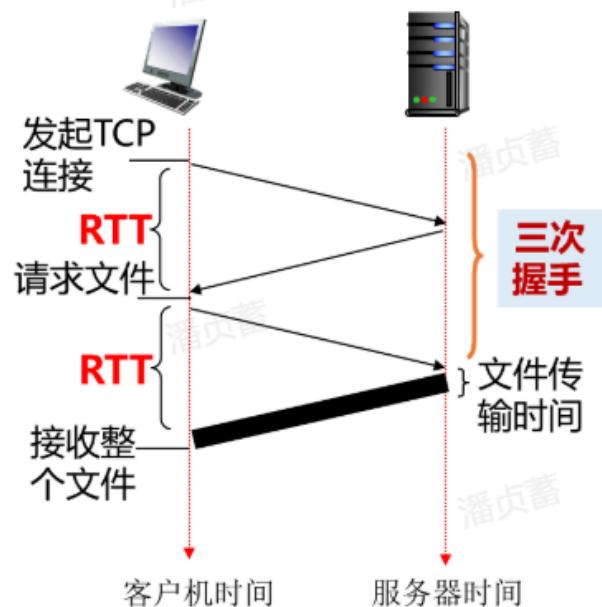
往返时间RTT

定义往返时间RTT(Round-Trip Time):

1个分组从客户主机到服务器再回到客户主机所花费的时间。

HTTP 响应时间(每个对象):

- 1个RTT用于建立TCP连接
- 1个RTT用于HTTP请求/响应消息的交互
- 对象/文件的传输时间



非持久HTTP响应时间:

$$\text{total} = 2\text{RTT} + \text{file transmit time}$$

TCP连接的“三次握手”过程

- 客户机发送一个TCP连接请求报文
- 服务器回送一个TCP确认响应报文
- 客户机向服务器发送一个包含“HTTP请求”与“TCP确认”的报文

持久HTTP连接

持久HTTP连接

非持久HTTP连接的问题

- 每个对象需要2个RTT
- OS必须为每个TCP连接分配主机资源
- 大量客户的并发TCP连接形成服务器的严重负担

持久HTTP连接

- 服务器发送响应消息后保持连接
- 同一客户/服务器的后续HTTP消息继续在该连接上传送



不带流水线的持久HTTP连接

- 客户先前响应消息收到，才发出新的请求消息
- 每个引用对象经历1个RTT

带流水线的持久HTTP连接

- HTTP/1.1默认使用
- 客户遇到1个引用对象就发送请求消息
- 所有引用对象只经历1个RTT

非流水线vs流水线

简单描述和比较持续连接的两种方式。

非流水线方式：客户机只能在前一个响应接收到之后才能发出新的请求。

特点：客户机为每一个引用对象的请求和接收都使用一个 RTT 时延。**会浪费一些服务器资源**

源：服务器在发送完一个对象，等待下一个请求时，**会出现空闲状态**。

流水线方式：客户机可一个接一个连续产生请求（只要有引用就产生）。服务器一个接一个连续响应请求，发送相应用对象。

特点：节省 RTT 时延，可能所有引用对象只花费一个 RTT。TCP 连接空闲时间很短。

非持久连接VS持久连接

简单描述非持续(久)连接和持续(久)连接。非持续(久)连接的主要缺点是什么？

非持续 HTTP 连接：每个 TCP 连接上只传送一个 Web 对象，只传送一个请求/响应对。

持续 HTTP 连接：传送多个请求/响应对，一个 TCP 连接上可以传送多个 Web 对象

非持续(久)连接的主要缺点是：**服务器负担重**。每一个对象的传输时延长：包含两个 RTT 时延

命题追踪 ► HTTP/1.1 页面请求时间的分析（2011、2022）

HTTP/1.1 默认使用持续连接。持续连接又分为非流水线和流水线两种工作方式。对于**非流水线方式**，客户在**收到前一个响应后才能发出下一个请求**，服务器在发送完一个对象后，其**TCP 连接就处于空闲状态**，浪费了服务器资源。对于流水线方式，客户可以连续发出对各个对象的请求，服务器就可连续响应这些请求。若所有的请求和响应都是连续发送的，则引用所有对象共计经历 1RTT 延迟，而不是像非流水线方式那样，每个对象都必须有 1RTT 延迟。这种方式减少了 TCP 连接中的空闲时间，提高了效率。此外，因为 HTTP 是基于 TCP 的，所以每 RTT 内传送的数据量还要受到 TCP 发送窗口的限制。

HTTP请求页面时间分析

命题追踪 ► HTTP 页面请求时间的分析 (2020、2024)

对于非持续连接，每个网页元素对象（如 JPEG 图形、Flash 等）的传输都需要单独建立一个 TCP 连接，如图 6.12 所示（第三次握手的报文段中捎带了客户对万维网文档的请求）。请求一个万维网文档所需的时间是该文档的传输时间（与文档大小成正比）加上两倍往返时间 RTT（一个 RTT 用于 TCP 连接，另一个 RTT 用于请求和接收文档）。每请求一个对象都导致 $2 \times \text{RTT}$ 的开销，此外每次建立新的 TCP 连接都要分配缓存和变量，使万维网服务器的负担很重。为了减小延时，浏览器通常会建立多个并行的 TCP 连接以同时请求多个对象。

所谓持续连接，是指万维网服务器在发送响应后仍然保持这条连接，使同一个客户和该服务器可以继续在这条 TCP 连接上传送后续的 HTTP 请求报文和响应报文，如图 6.13 所示。

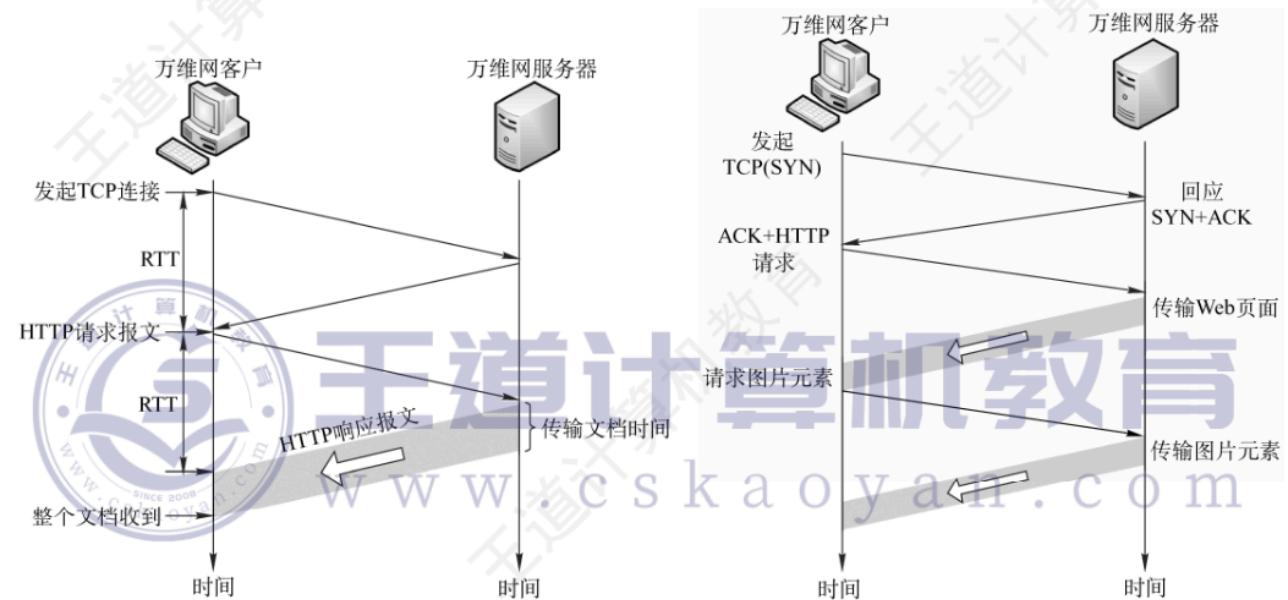


图 6.12 请求一个万维网文档所需的时间

图 6.13 使用持续连接（非流水线）

HTTP报文结构

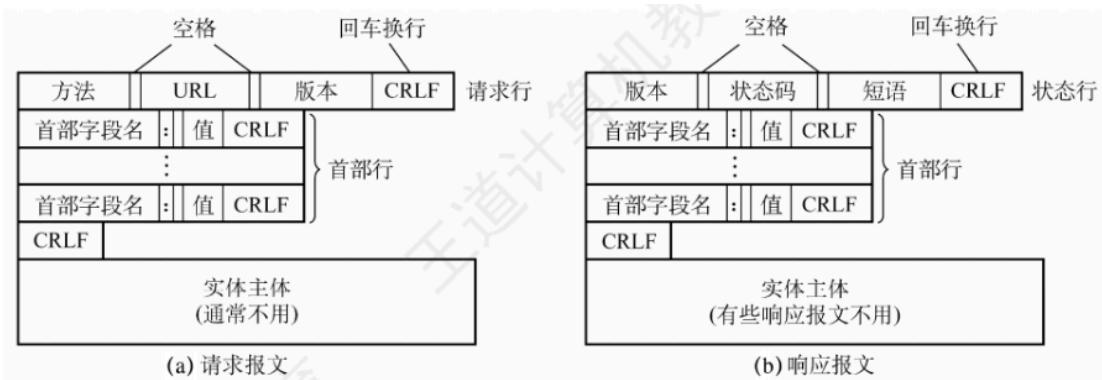


图 6.14 HTTP 的报文结构

从图 6.14 可以看出，两种报文都由三个部分组成，两者格式的区别就是开始行不同。

开始行：在请求报文中的开始行称为请求行，而在响应报文中的开始行称为状态行。开始行的三个字段之间都以空格分隔，最后的“CR”和“LF”分别代表“回车”和“换行”。

首部行：用来说明**浏览器、服务器或报文主体的一些信息**。首部可以有几行，但也可以不使用。在每个首部行中都有首部字段名和它的值，每一行的结束都要有“回车”和“换行”。整个首部行结束时，还有一空行将首部行和后面的实体主体分开。

实体主体：在请求报文中一般不用这个字段，而在响应报文中也可能没有这个字段。

请求报文的“请求行”有三个内容：方法、请求资源的 URL 及 HTTP 的版本。其中，“方法”是对所请求对象进行的操作，这些方法实际上也就是一些命令。表 6.1 列出了常用的几种方法。

常用方法

GET请求指定的页面信息，并返回实体主体。

HEAD类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头

POST向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

PUT从客户端向服务器传送的数据取代指定的文档的内容。

DELETE请求服务器删除指定的页面。

CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

OPTIONS允许客户端查看服务器的性能。

TRACE回显服务器收到的请求，主要用于测试或诊断。

PATCH是对 PUT 方法的补充，用来对已知资源进行局部更新。

表 6.1 HTTP 请求报文中常用的几个方法

方法(操作)	意 义
GET	请求读取由 URL 所标志的信息
HEAD	请求读取由 URL 所标志的信息的首部
POST	给服务器添加信息(如注释)
PUT	在指明的 URL 下存储一个文档
DELETE	删除指明的 URL 所标志的资源
CONNECT	用于代理服务器

下面是一个典型的 HTTP 请求报文：

```
GET /bbs/index.htm HTTP/1.1 {指明方法“GET”、相对 URL、HTTP 版本}
Host: www.cskaoyan.com {指明服务器的域名}
Connection: Keep-Alive {要求服务器在发送完被请求的文档后保持这条连接}
User-Agent: Mozilla/5.0 {表明用户代理是浏览器 Mozilla/5.0}
Accept-Language: cn {表示用户希望优先得到中文版本的文档}
```

第 1 行是请求行，它使用了相对 URL，因为下面的首部行给出了服务器的域名。第 3 行告诉服务器使用持续连接，表示浏览器要求服务器在发送完被请求的文档后保持这条 TCP 连接，若要求使用非持续连接，则对应的首部行应为“Connection: close”。

响应报文HTTP response message

潘贞蓄



响应状态码

HTTP 响应的状态码



勺响应消息的第一行。

马：

200 OK

请求成功, 所请求信息在响应消息中返回

301 Moved
Permanently

所请求的对象已永久迁移, 新的URL在本响应消息的 (location:) 头部指出

400 Bad
Request

该请求不能被服务器解读

404 Not
Found

服务器上不存在所请求文档

505 HTTP
Version Not
Supported

服务器不支持请求报文使用的HTTP版本

■ HTTP/1.1 的问题：一条 TCP 连接串行处理多个对象请求

✓ 图 1 和图 3：

- 客户端用**一条 TCP 连接**发出多个 GET 请求（请求 O_4 、 O_3 、 O_2 、 O_1 ）
- 如果 O_1 是一个视频文件（非常大），其他较小的对象 (O_2 、 O_3 、 O_4) **只能等 O_1 传完才能开始传**
- 这是因为 **HTTP/1.1 使用 FCFS (先来先服务) 队列机制**，不能并行
- 所以出现了 **HOL 阻塞现象**：

mathematica

复制 编辑

O_2 、 O_3 、 O_4 明很小，却被大文件 O_1 阻挡

HTTP/2 的解决方式：多路复用、帧划分、优先级支持

图 2 和 图 4：

HTTP/2 为了提升性能，引入了以下改进：

① 对象“分帧”处理：

- 将每个对象 ($O_1 \sim O_4$) **拆分成许多小帧 (frame)**
- 每个对象变成多个帧组成的“流”

② 多路复用 (Multiplexing) :

- 所有对象帧可以在一条 TCP 连接中交错发送
也就是说：
 - O_2 先发一点
 - 接着发 O_4 的几帧
 - 然后发 O_3
 - 最后 O_1 的大文件慢慢传
- 无需等待队头对象完成传输，有效缓解 HOL 阻塞

③ 支持优先级：

- 客户端可以告诉服务器：**哪个对象更重要**
- 服务器按优先级发送对象，提升体验 (网页结构比图片更优先)

重点问题：HTTP/2 的局限

- 虽然 HTTP/2 引入了**多路复用**（多对象并行传输），但它仍然基于 **TCP**。
- 一旦 TCP 的某个数据包丢失，**整个连接都要等丢包重传**，会造成 **所有对象都被阻塞 (HOL 阻塞复现)**。
- 浏览器为了缓解这个问题，有时仍然会开多个 TCP 连接（浪费资源）。

✓ HTTP/3 的改进:

▲ 从 TCP 切换到了 UDP + QUIC 协议

- QUIC 是谷歌主导开发的新传输层协议，运行在 UDP 之上，支持：
 - 多路复用
 - 加密 (TLS 1.3)
 - 拥塞控制
 - 更灵活地应对丢包——不会阻塞所有对象传输

✚ 好处：

- 每个对象的传输独立，不再因为一个包丢了就卡所有资源
- 更快握手建立连接 (1-RTT)
- 内建 TLS 加密，比 HTTPS 更快更安全

HTTP/1, HTTP/2, HTTP/3

HTTP/1.0: RFC 1945

(May 1996)

HTTP/1.1: RFC 2616

(June 1999)

HTTPS: RFC 2818

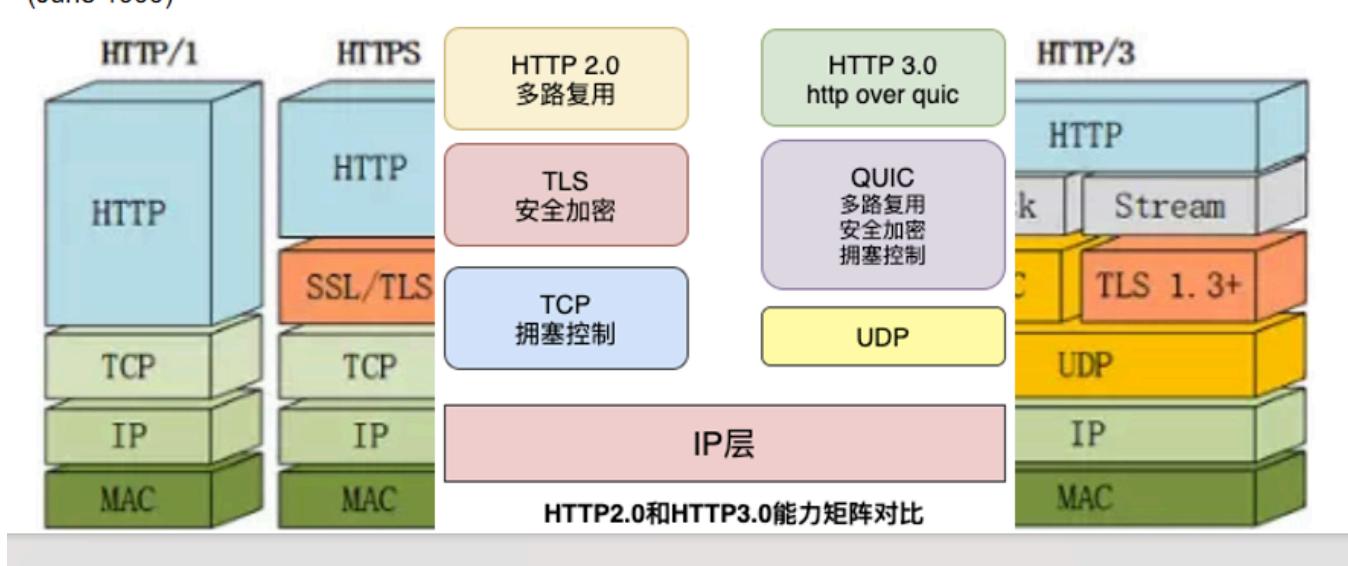
(May 2000)

HTTP/2: RFC 7540

(May 2015)

HTTP/3: RFC 9114

(June 2022)



✿ 第二张图：HTTP/1 到 HTTP/3 的技术演进（图解）

砖块 各协议栈层级对比（从底到顶）

层级	HTTP/1 & HTTP/2	HTTP/3
MAC	网络硬件接口（如网卡）	一样
IP	网络层	一样
传输层	TCP（有顺序、重传）	UDP + QUIC（更灵活控制）
加密层	SSL/TLS	QUIC 内建 TLS 1.3 加密
应用层	HTTP/1、HTTP/2	HTTP/3

🚀 演进路径总结：

协议版本	发布时间	特点
HTTP/1.0	1996	每个资源一个连接，效率极低
HTTP/1.1	1999	支持持久连接，但仍有队头阻塞
HTTP/2	2015	多路复用、二进制帧封装，仍基于 TCP
HTTP/3	2022	基于 UDP+QUIC，彻底解决 HOL 问题

🎯 你该记住的重点对比：

特性	HTTP/1.1	HTTP/2	HTTP/3
传输协议	TCP	TCP	UDP (上面运行 QUIC)
多路复用	✗	✓	✓
队头阻塞 (HOL)	严重	减轻但仍存在	<u>彻底解决</u>
加密方式	可选 (SSL/TLS)	TLS 强制加密	内建 TLS 1.3 加密
丢包影响	阻塞整个连接	阻塞整个连接	<u>只影响相关流</u>
建立连接效率	慢 (需握手多次)	改进但仍有延迟	更快 (1-RTT 连接)

Cookie

原因？

HTTP GET/响应交互是无状态的，不保存客户信息。

Cookie：允许Web站点跟踪、识别用户；服务器可以限制用户访问，或把内容与用户身份关联。

包括四部分

包括四个部分：

- 在**HTTP响应报文**中有一个cookie 首部行
- 在**HTTP请求报文**中有一个cookie 首部行
- **用户主机**中保留有一个 **cookie 文件**并由浏览器管理
- **Web站点的后端数据库**保存cookie

不利于隐私保护

Web缓存

也叫代理服务器，是能够代表起始服务器来满足 HTTP请求的网络实体。

. 什么是 Web 缓存？简单描述 Web 缓存的作用。

Web 缓存器(Web cache): 也叫代理服务器，是能够代表起始服务器来满足 HTTP 请求的网络实体。

保存最近请求过的对象的副本，如果后续有对同一个对象的请求，则直接发送缓存的副本。

使用 Web 缓存具有以下优点：

减少对客户机请求的响应时间

减少内部网络与接入链路上的通信量，能从整体上大大降低因特网上的 Web 流量。

条件GET方法

■ Web 应用和 HTTP 协议



条件GET方法

□ 高速缓存：

□ 减少响应时间；

□ 存放在缓存中的对象拷贝可能是旧的，即保存在起始 Web 服务器中的对象可能已经被修改。

□ 条件GET目的：

□ 证实缓存器中的对象是否为最新

□ Web 服务器回发响应报文：包括对象的最后修改时间

Last-modified: <date>

□ 缓存器：在请求报文中包含对象最后修改时间，**If-modified-since: <date>**

□ Web 服务器：如果对象是最新的则响应报文中不包含对象

304 Not Modified (HTTP/1.1 304 Not Modified)，并且实体为空。

证实其保存的对象是否为最新

原理

6. 在使用 Web 缓存时，缓存器采用了什么技术来证实其保存的对象是否为最新的？描述相关技术的原理。

Web 缓存使用条件 GET 方法，来证实其保存的对象是否为最新的

Web 服务器回发响应报文：包括对象的最后修改时间：Last-modified: date1

缓存检查 Web 服务器中的该对象是否已被修改，发送一个条件 GET 请求报文：报文中包含 If-modified-since: date1 首部行

告诉服务器，仅当自指定日期之后该对象被修改过，才发送该对象。

若 Web 服务器中的该对象未被修改，则响应报文含有 304 Not Modified，并且实体为空。

过程

Q：描述使用 Web 缓存服务器后网页的访问过程以及使用条件 GET 请求更新对象的过程。

A：客户的所有网页请求都送达 WEB 缓存服务器

WEB 缓存服务器先查询本地是否具有请求对象，如果有且没有超时(或过期)则直接返回给客户，

如果 WEB 缓存服务器本地没有所请求的对象，则转发请求到起始网页服务器，并接收网页响应，然后缓存在本地并转发给客户

如果本地存在请求的网页但超时(或过期)，则 WEB 缓存服务器使用条件 GET 进行本地缓存更新；

web 服务器收到条件 GET 请求后判断自己是否对网页进行了修改，如果没有修改，则只返回未修改的响应报头，否则返回整个网页文件。

简要过程说明：

1. 所有客户端请求都先发送到 Web 缓存服务器。

2. 缓存服务器检查本地是否已有请求的网页副本：

- 有并且未过期：直接返回给客户端。
- 没有该副本：向源服务器转发请求，获取网页，缓存本地后再返回客户端。
- 有但已过期：向源服务器发送条件 GET 请求 (If-Modified-Since)。

3. 源服务器处理条件 GET 请求：

- 如果网页没有更新：返回 304 Not Modified，客户端继续使用本地缓存。
- 如果网页已更新：返回最新网页内容，缓存更新并发送给客户端。

总结一句话：

Web 缓存服务器会尽量使用本地缓存来响应请求，只有在无缓存或缓存过期时才会向源站请求，通过条件 GET 请求判断是否需要更新资源，以提高效率和节省带宽。

因特网中的电子邮件

电子邮件

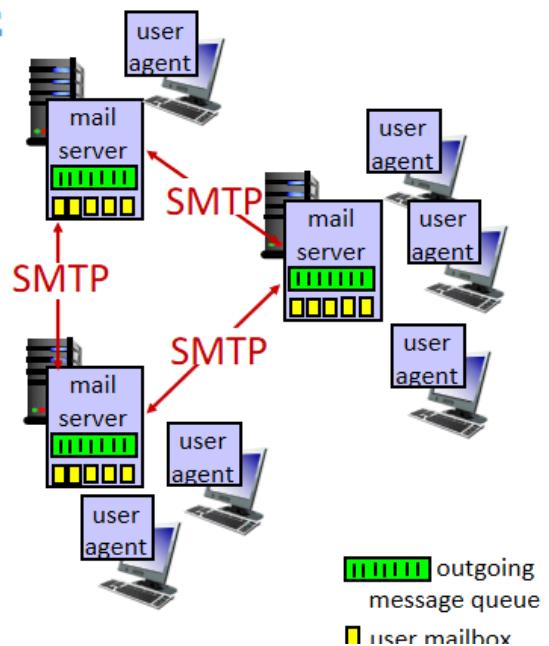
电子邮件

3个主要组成部分:

- 用户代理 user agents
- 邮件服务器 mail servers
- 简单邮件传送协议 SMTP

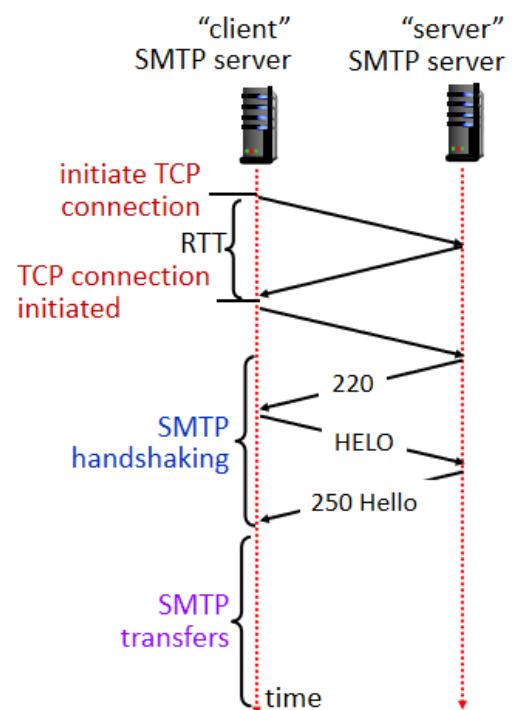
用户代理 User Agent

- 也就是邮件阅读器, 如Outlook, foxmail等
- 允许用户阅读、回复、转发、保存、编辑邮件消息
- 传出/传入消息存储在服务器上



电子邮件: SMTP [RFC 5321]

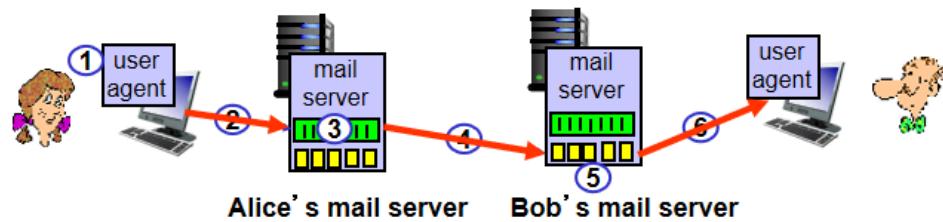
- 客户使用TCP来可靠传输邮件消息到服务器端口号25
 - **直接传送**: 从发送服务器到接收服务器
- 传输的3个阶段
 - SMTP 握手 (问候)
 - SMTP 消息传输
 - SMTP 结束
- 命令/应答交互 (类似 HTTP)
 - commands: **ASCII文本格式**
 - response: 状态码及其短语



发送报文过程

例子: Alice发送邮件消息到Bob

- 1) Alice启动邮件代理，提供接收方的邮件地址，撰写邮件
- 2) Alice的用户代理把报文发给其邮件服务器，放在发送队列中
- 3) Alice邮件服务器的SMTP的客户机侧创建与Bob的邮件服务器的TCP连接，经过应用层握手
- 4) SMTP通过TCP连接发送报文
- 5) Bob的邮件服务器接收并将该报文放入Bob的邮箱
- 6) Bob调用其用户代理来读报文



写->发->建->连->收->读

✉ 对应步骤详解 (结合图中序号)

步骤	关键词	含义
①	写	Alice 用邮件客户端写信，填 Bob 的地址
②	发	邮件代理将信交给 Alice 的邮件服务器，放入 发送队列
③	建	Alice 邮件服务器通过 SMTP 协议准备连接 Bob 的邮件服务器
④	连	通过 TCP 连接 传送邮件报文
⑤	收	Bob 的邮件服务器接收邮件，存入 Bob 的邮箱中
⑥	读	Bob 用自己的邮件客户端读取邮件

SMTP: 总结

与HTTP的比较:

- SMTP 使用**持久连接**
- SMTP 要求邮件消息(header & body) 必须是**7-bit ASCII**
- SMTP 服务器使用 CRLF,CRLF 来判断邮件消息的结束

• HTTP: 拉协议

- 用户使用HTTP从服务器**拉取**信息。其TCP连接是由想获取文件的机器发起。

• SMTP: 推协议

- 发送邮件服务器把文件**推向接收邮件服务器**, 其TCP连接是由要发送文件的机器发起。
- 都有ASCII 命令/应答交互, 状态码
- HTTP: 每个对象封装在它各自的HTTP响应消息中发送
- SMTP: 一个邮件内各个对象置于同一个邮件消息的多部分发送

邮件访问协议

邮件访问协议

- SMTP: 递送/存储邮件消息到接收者邮件服务器
- 邮件访问协议: 从服务器获取邮件消息
 - POP: Post Office Protocol 邮局协议[RFC 1939]110端口号
 - 身份认证 (代理 <->服务器) 并 下载邮件消息
 - 包含三个阶段:
 - 授权, 事务处理, 更新, POP是没有状态的, 当断开连接之后, 需要重新登录
 - IMAP: Internet Message Access Protocol [RFC 3501] 143端口
 - 更多功能特征 (更复杂!)
 - 允许用户像对待本地邮箱那样**操纵远程邮箱的邮件**
 - HTTP: 也可以通过HTTP来进行邮件的接收

SMTP vs HTTP

SMTP 与 HTTP 的简单对比

相同点

都用于从一台主机向另一台主机**传送文件**

持久 HTTP 和 SMTP **都使用持久连接**。

不同点

HTTP 是拉协议：**其 TCP 连接是由想获取文件的机器发起**。

SMTP 是推协议：**其TCP 连接是由要发送文件的机器发起**。

SMTP 使用 7 位 ASCII 码格式，HTTP 数据没有该限制。

对含有文本和图形 (或其他媒体类型的) 文档：HTTP 把每个对象封装在它各自的HTTP 响应报文中发送；电子邮件则把所有报文对象放在一个报文中。

POP3

POP3（邮局协议 第3版）是一种**电子邮件接收协议**

三、POP3 的典型特性

特性	说明
 接收邮件	只能收邮件， 不能发邮件 （发邮件使用 SMTP 协议）
 默认删除	邮件下载到本地后，默认从服务器删除（可设置保留）
 非持久会话	连接一次就断开，不保持持续连接
 本地保存为主	适合一个客户端 + 单台设备使用场景， 不适合多设备同步 （推荐 IMAP）

四、POP3 三个工作状态（协议设计层面）

1. 授权状态（Authorization）

- 用户输入用户名、密码进行登录验证

2. 事务状态（Transaction）

- 成功登录后，可以查看、下载、删除邮件等操作

3. 更新状态（Update）

- 断开连接时，服务器执行删除操作并更新邮箱状态

POP3 和 IMAP

POP3的更多细节 (缺陷)

- 先前例子使用 “Download-and-delete”，**用户读取邮件后，服务器不再保存**
- **用户无法在邮件服务器上远程创建文件夹**
- Bob换客户端后不能再读邮件
- “Download-and-keep”模式：在不同客户机上的邮件拷贝
- **POP3的会话是无状态的**

IMAP

- 保存所有邮件消息在一个位置：**服务器**
- 允许用户在服务器的各文件夹中管理邮件消息
- IMAP跟踪用户会话的状态信息：
 - 文件夹和邮件消息IDs与文件夹名字的映射

“邮件服务器之间转发电子邮件可以采用 POP3 协议”

X 错误！

- POP3 是**接收邮件协议**，用于**客户端从服务器取邮件**；
- 服务器之间不使用 POP3。

辨析

协议	功能
SMTP	发邮件 (客户端和服务器间 & 服务器之间)
POP3	接收邮件 (一次性下载)
IMAP	接收邮件 (支持多端同步、在线查看)
MIME	邮件内容格式规范 (不是传输协议)

DNS:英特网的目录服务

主要功能

请描述 DNS 所提供的服务，以及两种解析（查询）方式

DNS 最基本的服务是：提供主机名到 IP 地址的转换。

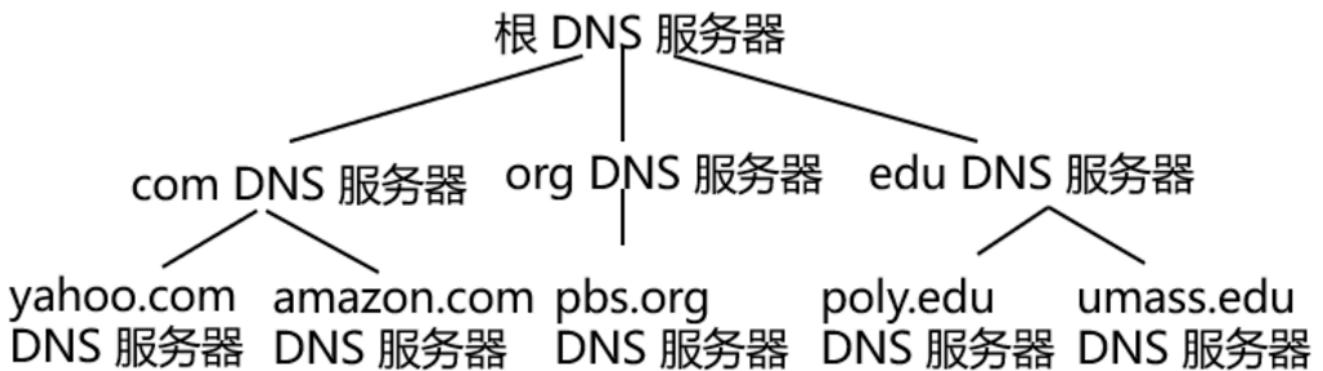
主机别名服务：应用程序可以调用 DNS 来获得主机别名对应的规范主机名以及主机的 IP 地址。

邮件服务器别名：电子邮件应用程序可以调用 DNS，对提供的邮件服务器别名进行解析，以获得该主机的规范主机名及 IP 地址。

负载分配：当客户对映射到某地址集合的名字发出一个 DNS 请求时，该 DNS 服务器用 IP 地址的整个集合进行响应，但在每个回答中循环这些地址次序，可以实现在所有冗余服务器之间循环分配负载。

查询方法包括递归解析和迭代解析。

分层架构



DNS

域名系统

Domain Name System:

- **分布式数据库**: 一个由分层 DNS服务器实现的分布式数据库
- **应用层协议**: DNS服务器实现域名转换 (域名/地址转换)

为什么不集中式DNS?

- 单点故障
- 巨大访问量
- 远距离集中式数据库
- 维护

不可扩展!

四种类型域名服务器

- **根域名服务器 (Root DNS servers)**
- **顶级域名服务器 (Top-level domain (TLD) servers)**
- **权威域名服务器 (Authoritative DNS servers)**
- **本地域名服务器 (Local DNS server)**

当一个主机发出DNS查询报文时，这个报文就首先被送往该主机的本地域名服务器。

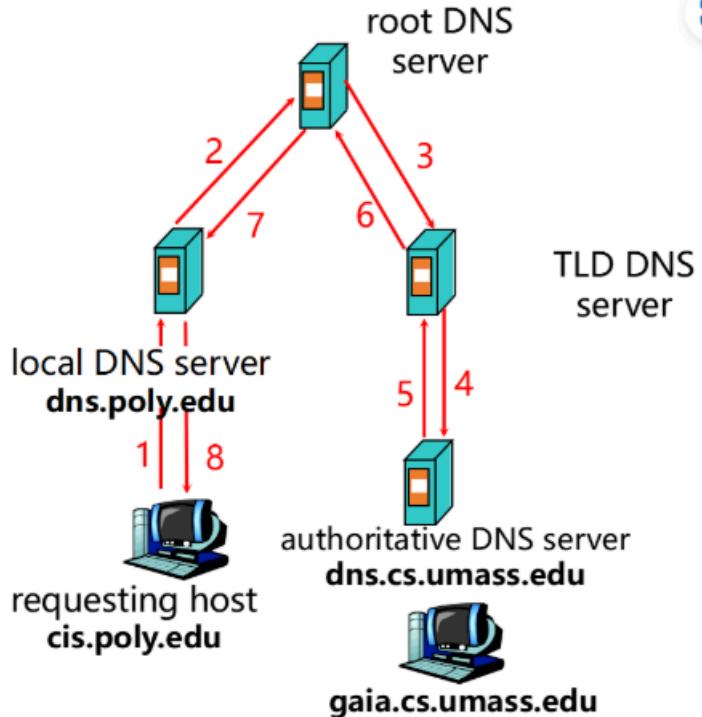
起着代理的作用，转发请求到层次结构中。在用户的计算机中设置网卡的“Internet协议 (TCP/IP) 属性”对话框中设置的首选DNS服务器即为本地域名服务器。本地域名服务器离用户较近，一般不超过几个路由器的距离。

查询方式

DNS查询方法一

递归查询(recursive query):

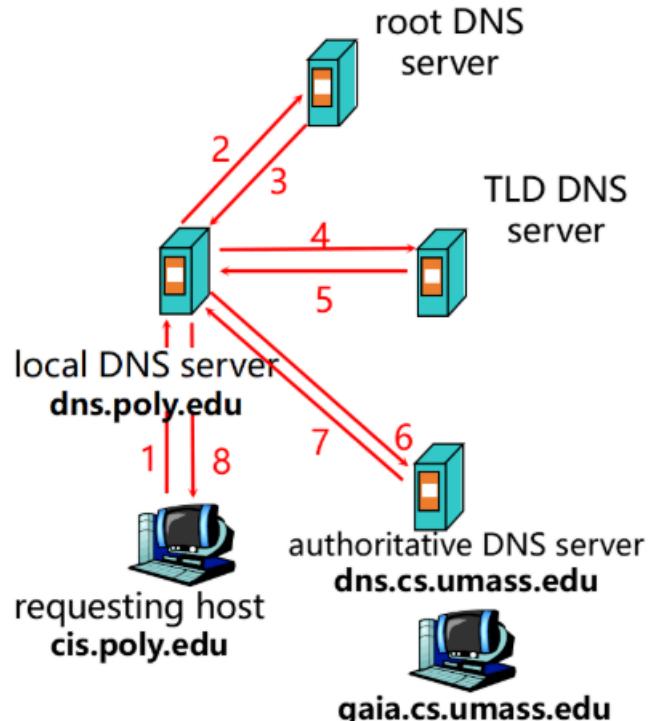
- 名字解析的负担交给被查询的名字服务器
- 被查询的名字服务器负载重?



DNS查询方法二

迭代查询(iterated query):

- 被查询的名字服务器 回复可以被查询的名字服务器的IP地址
- “我不知道它的名字，但是你可以问服务器xx”



✓ 一、核心概念简述

查询方式	递归查询 (Recursive)	迭代查询 (Iterative)	对比
谁负责到底	上一级帮你查到底	上一级只告诉你下一步去哪问	
查询者	DNS 客户端 (浏览器/操作系统)	本地 DNS 服务器	

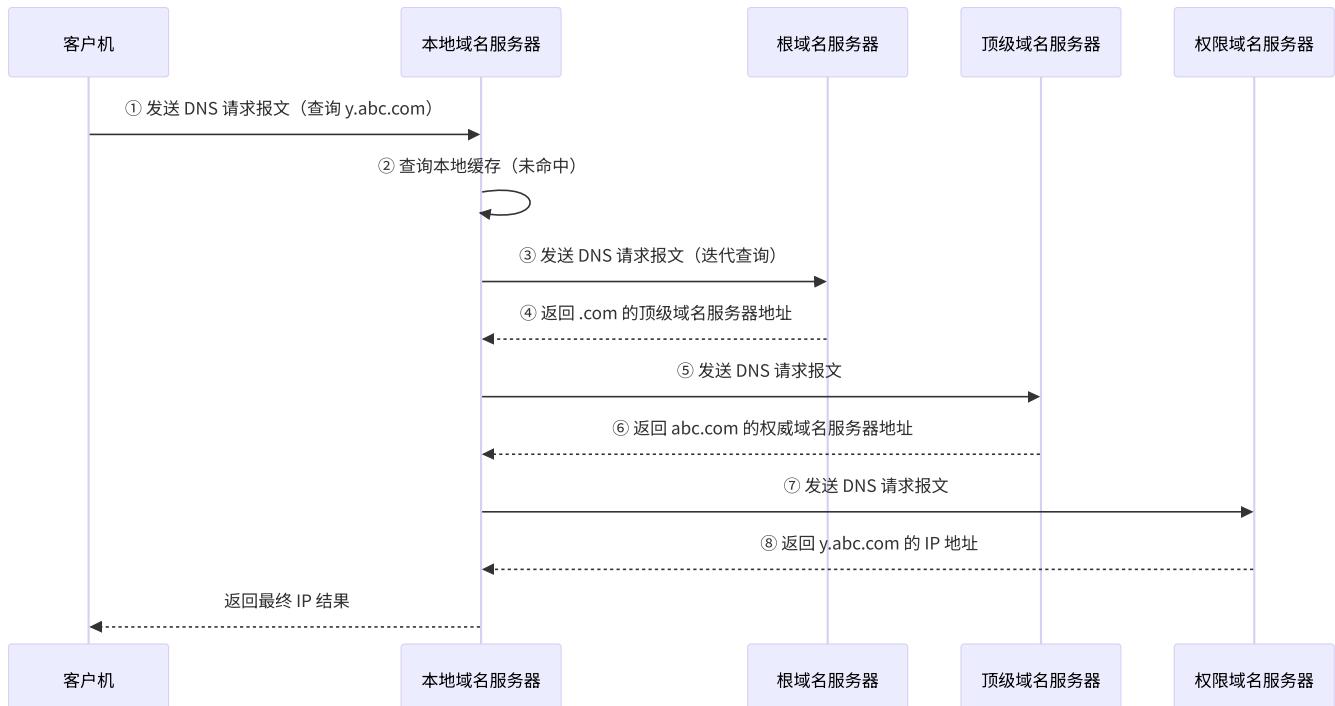
📦 三、DNS 中的对应角色

步骤	递归查询	迭代查询
用户 → 本地域名服务器	递归：用户让本地帮我查到底	一般就是递归（用户只发一次请求）
本地域名服务器 → 根服务器等	本地服务器递归查到底 (很少这么做)	本地服务器自己迭代地逐级去问

👉 所以：

**用户和本地域名服务器之间用递归，
本地域名服务器向上问（根/TLD/权威）时用迭代。**

域名解析过程



缓存

为了提高 DNS 的查询效率，并减少因特网上的 DNS 查询报文数量，在域名服务器中广泛地使用了高速缓存，用来缓存最近查询过的域名的相关映射信息。这样，当另一个相同的域名查询到达该 DNS 服务器时，该服务器就能直接提供所要求的 IP 地址。因为主机名和 IP 地址之间的映射不是永久的，所以 DNS 服务器将在一段时间后丢弃高速缓存中的信息。在主机中同样也需要高速缓存，许多主机在启动时从本地域名服务器下载域名和地址的全部数据库，维护存放自己最近使用的域名的高速缓存，并且只在从缓存中找不到域名时才使用域名服务器。

DNS攻击

1、DNS劫持

✓ 4. 域名劫持 (Hijacking)

攻击者控制域名注册商账号，修改域名的 IP 指向。

- 举例：劫持 `baidu.com` 的 DNS 设置，把 IP 改为恶意服务器。
- 影响极大，用户访问官网也可能被引导到恶意网站。

2、缓存投毒

1. DNS 缓存投毒 (Cache Poisoning)

目的：把合法域名指向恶意 IP。

- 举例：攻击者向本地 DNS 缓存注入：

www.bank.com → 6.6.6.6 (钓鱼网站)

- 用户访问时被重定向到钓鱼页面，输入账号密码被骗。
- 特点：**不攻击网站本身，而是攻击用户“如何到达”网站的路线。

3、DDOS攻击 (Distributed Denial of Service)

3. DNS 反射攻击 (放大 DDoS)

利用“开放DNS服务器”为攻击者发起**大规模反射流量**，攻击受害者主机。

- 攻击者发送大量 DNS 请求，但把**源 IP 伪装成受害者 IP**。
- 所有 DNS 响应流量会被反射到受害者，造成瘫痪。
- 关键：**DNS 报文小请求 → 大响应，具备“放大效果”

4、DNS欺骗

2. DNS 欺骗 (Spoofing)

实时**伪造 DNS 响应**，在合法服务器响应前抢先发回假答案。

- 举例：你发出 www.baidu.com 的 DNS 请求，攻击者伪造一个应答抢先返回错误 IP。
- 类似缓存投毒，但更“即时”，不依赖缓存服务器。

内容分发网络CDN



什么是 CDN (Content Distribution Network)

CDN 的中文名是**内容分发网络**，它的本质是：

| 在全球多个地点部署很多服务器，把内容提前“复制一份”，靠近用户存储，从而加快访问速度、减少压力。

为什么需要



为什么需要 CDN?

课件中提到几个挑战，举例说明：



情景：Bob 想看一个在线视频（比如 Netflix）

- 如果全世界的人都从 **一个中心服务器** 请求视频：
 - 🚨 服务器压力太大（单点故障）
 - 🚧 网络路径太长（延迟高）
 - 📈 多用户同时看，带宽爆炸，速度卡顿



解决方案：用 CDN

- Netflix 把视频放到多个地点的 CDN 节点服务器上
- Bob 在中国访问时，**从最近的 CDN 服务器获取视频**
- 📈 更快！更稳定！全球用户访问也能分流



CDN 的工作原理（流程）

以下是课件中 Bob 访问视频的完整流程：

1. **Bob 打开 netcinema.com, 点击视频链接**
2. 他的电脑向本地 DNS 服务器请求解析该链接
3. netcinema 的 DNS 回复一个新地址（跳转到 CDN 链接）：
👉 比如 `http://KingCDN.com/NetC6y&B23V`
4. 本地 DNS 再去找 KingCDN.com 的 DNS 服务器获取真正的 IP
5. 得到某个最近的 CDN 节点地址（靠近 Bob）
6. Bob 和这个 CDN 节点建立 TCP 连接并发起 HTTP 请求
7. 视频数据从 CDN 节点传回 Bob 的浏览器

两种部署方式



两种 CDN 部署方式

类型	解释	举例说明
深度部署（deep deployment）	把 CDN 节点部署到每个接入网络里（离用户最近）	Akamai
浅层部署（bring CDN into ISP）	在骨干网边缘放少量大节点，邀请 ISP 接入	Limelight 等

提升速度、降低压力、增强稳定性

三、运输层

功能:1.应用进程之间的逻辑通信

2. 复用和分用
3. 差错检测
4. 提供面向连接和无连接的传输协议

运输层服务

传输层服务和协议

- 在两个不同的主机上运行的应用程序之间提供**逻辑通信**
- **传输层协议运行在端系统**
 - 发送方: 将应用程序报文分成数据段传递给网络层,
 - 接受方: 将数据段重新组装成报文传递到应用层
- 不只一个传输层协议可以用于应用程序
 - 因特网: TCP 和 UDP



两个基本的传输层协议

- **TCP: Transmission Control Protocol**
 - 可靠的、按序的交付
 - 提供拥塞控制
 - 提供流量控制
 - 需要连接建立
- **UDP: User Datagram Protocol**
 - 不可靠、不按序交付
 - “尽力而为”的IP不提供不必要的服务的扩展
- **不提供的服务:**
 - 时延保证
 - 带宽保证

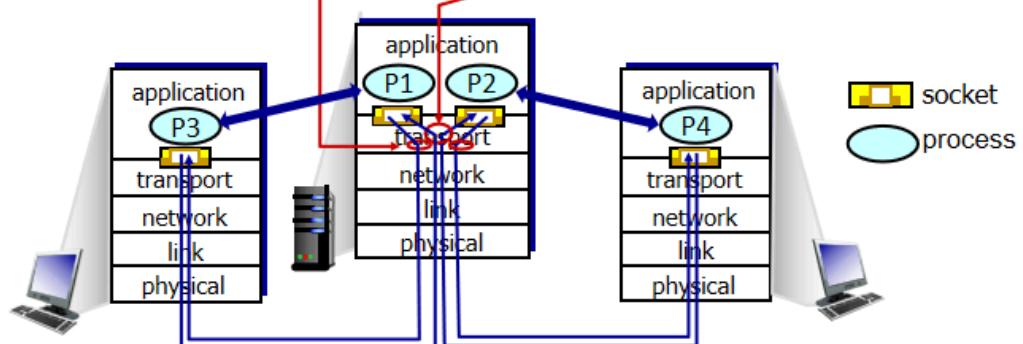
多路复用和多路分解

在发送主机多路复用:

从多个套接字收集数据, 用首部封装数据, 然后将报文段传递到网络层。

在接收主机多路分解:

使用头部信息将接收到的报文段传递到正确的套接字。



✓ 总结整张图的流程:

1. 多个进程 → 传输层收集数据 → 加首部 → 发给网络层 (多路复用)
2. 另一端传输层收到报文段 → 看端口号 → 分发给目标进程 (多路分解)



最终总结一句话：

多路复用 / 分解 是靠报文中的 IP 地址和端口号实现的，
UDP 用“目的地址+目的端口”就够了，
TCP 要用“源地址+源端口+目的地址+目的端口”来唯一识别连接。

1. 多路复用与多路分解

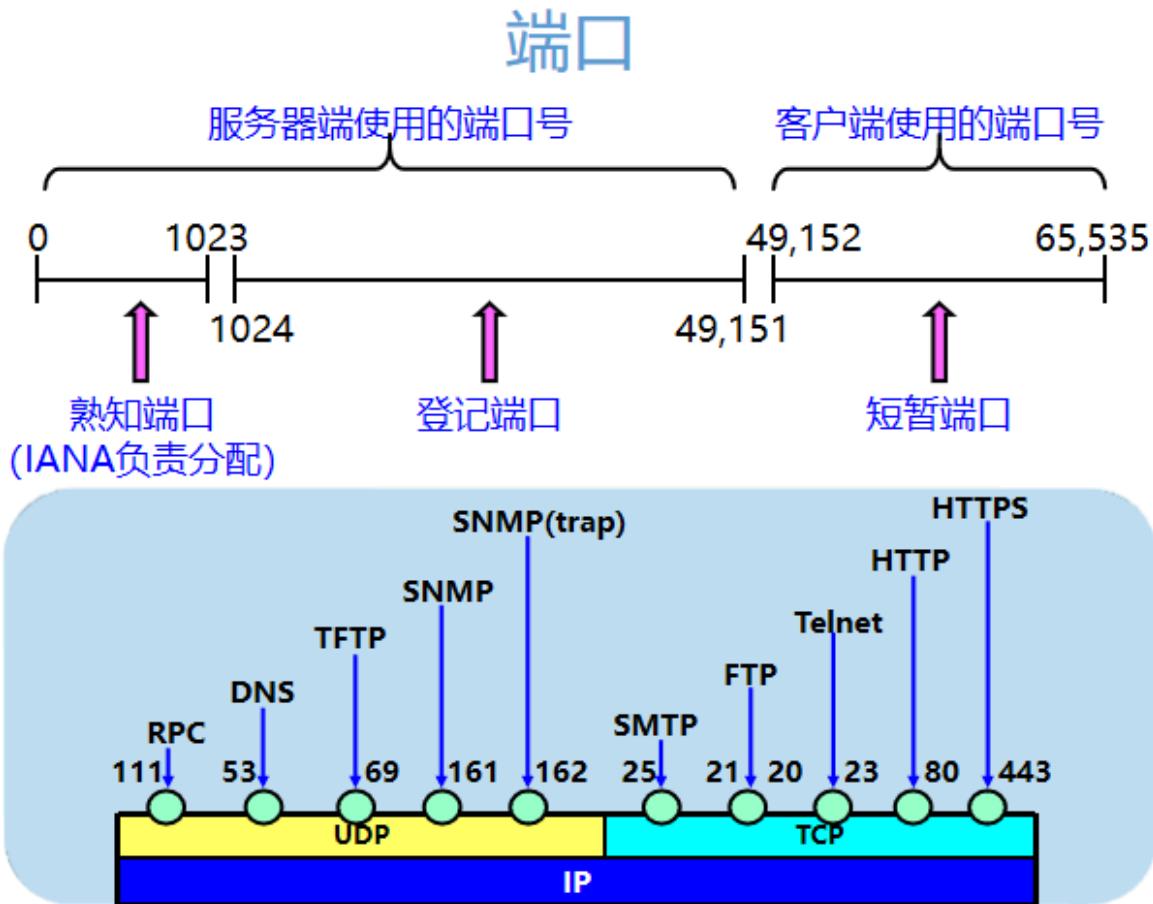
- 多路复用/分解如何工作? (报文段头部字段实现)
 - 主机收到IP数据报
 - 每个数据报有源IP地址, 目的IP地址
 - 每个数据报搬运一个报文段每个报文段有源和目的端口号 (回忆: 对于特定应用程序具有周知端口号)
 - 主机用IP地址和端口号指明报文段属于哪个合适的套接字
- 无连接多路分解 (UDP 套接字: 目的IP地址, 目的端口号)
- 面向连接的多路分解 (TCP套接字: 源IP地址、源端口号、目的IP地址、目的端口号)

端口



什么是端口号?

- 端口号是一个 **16位无符号整数 (0 ~ 65535)**
- 用来唯一标识一个主机内的通信进程 (配合 IP 地址)
- TCP 和 UDP 协议各自独立使用一套端口号



图中下半部分（彩色块）：端口号对应协议图示

- 黄色是 UDP 端口使用情况 (如 DNS、TFTP、SNMP)
- 蓝绿色是 TCP 协议常用端口 (如 HTTP、HTTPS、FTP、Telnet)
- 两种协议都基于 IP 层
- 注意：一个进程可以分别监听 TCP 和 UDP 的同一端口号，它们互不冲突

客户端端口（图右上）

- 客户端通常不需要绑定固定端口
- **系统自动分配临时端口 (49152~65535) **来发起连接
- 举例：你访问百度，浏览器临时使用 50123 端口发送请求 → 百度服务器的 80 端口

客户端用随机端口发起连接，传输层通过它完成多路分解

套接字

3. 套接字

在网络中通过 IP 地址来标识和区别不同的主机，通过端口号来标识和区分一台主机中的不同应用进程，端口号拼接到 IP 地址就构成套接字（Socket）。在网络中采用发送方和接收方的套接字来识别端点。套接字，实际上是一个通信端点，即

$$\text{套接字 (Socket)} = (\text{IP 地址} : \text{端口号})$$

它唯一地标识网络中的一台主机上的一个应用进程。

在网络通信中，主机 A 发给主机 B 的报文包含目的端口号和源端口号，源端口号是“返回地址”的一部分，即当 B 需要发回一个报文给 A 时，B 到 A 的报文中的目的端口号便是 A 到 B 的报文中的源端口号（完全的返回地址是 A 的 IP 地址和源端口号）。

无连接多路分解

原理

第一张图：无连接多路分解的原理

✓ 多路分解是怎么做的？

当主机收到 UDP 报文段时：

1. 检查报文段的“目标端口号”
2. 操作系统会查找有没有绑定这个端口号的 UDP 套接字（DatagramSocket）
3. 如果找到了，就把报文段交给对应的应用程序进程去处理

▲ 注意重点：

即使收到的数据包：

- 来自不同的 IP 地址
- 来自不同的源端口号

只要它们的“目标端口号”相同 → 都会被送到相同的套接字对象

◀ 创建 UDP 套接字时怎么做?

1. 必须要明确：绑定本地端口号

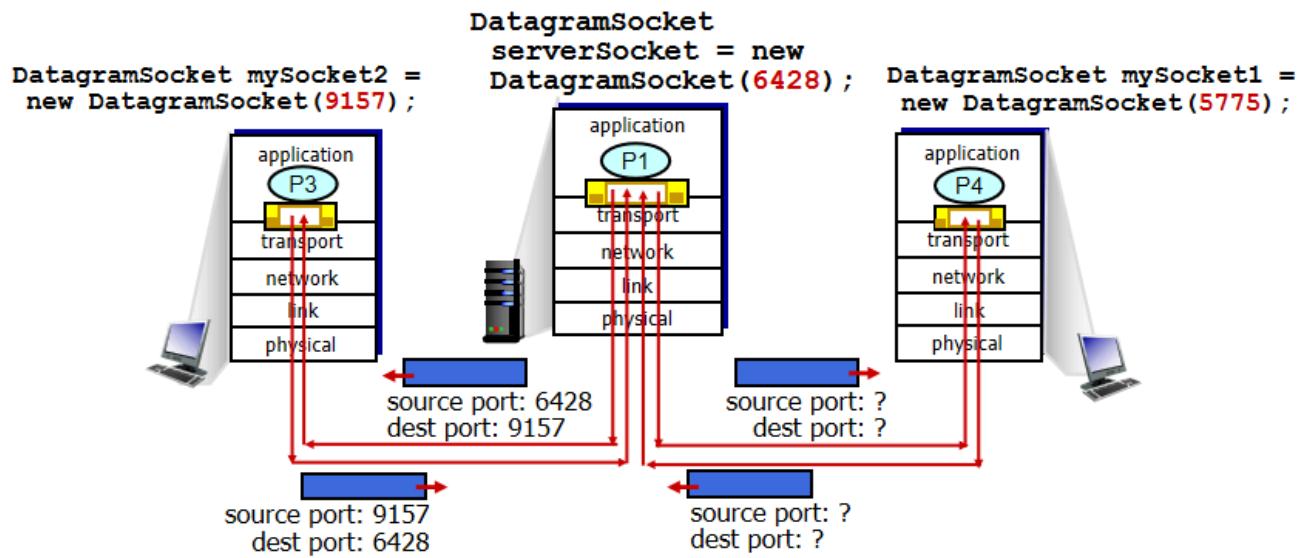
例如：`new DatagramSocket(12534)` 就表示这个进程“监听端口号 12534”

2. 当我们发送数据时，还需要指定：

- 目标 IP 地址
- 目标端口号
- 系统会把这些封装进报文头部，用来告诉对方“你应该把数据发回到这儿”

例子

无连接多路分解例子



请求报文段中提供返回地址（包括IP地址和端口号）

实际发生了什么?

客户端 mySocket2 向服务器发请求:

- 报文段头部内容:
 - source port: 9157
 - dest port: 6428
- 中间主机上的服务器会检查 6428 端口，找到绑定的 socket
- 接收后能知道**报文是从哪台主机的哪个端口发来的**

服务器回应:

- 把返回数据发向:
 - IP地址 = 客户端 IP
 - 端口号 = 报文中的 source port (比如 9157)
- 于是**客户端的 socket (绑定了 9157) 就能接收到回应数据**

为什么说是无连接？

为什么说这是“无连接”?

- **没有先建立连接** (不像 TCP 要三次握手)
- 每次收发都是独立的
- 只要目标 IP 和目标端口号对得上，数据就能发出去 / 收进来

面向连接的多路分解

💡 先理解“什么是面向连接的多路分解”？

当多个 TCP 客户端同时连接到服务器时，服务器要能识别并区分每一个连接，让它们互不干扰。

四元组标识

✓ TCP 套接字用“四元组”来唯一标识一个连接

组成部分	含义
源 IP 地址	发起连接的客户端 IP
源端口号	客户端使用的临时端口号
目的 IP 地址	服务器 IP
目的端口号	通常是 80 (HTTP) 或其他服务端口

多路复用

多路复用：接收主机使用这四个值来将报文段定位到适当的套接字

✓ 多路分解：接收端根据这四个值来找到“对应哪个连接（socket）”

也就是说：

- 三个客户端即使都连到同一个服务器的 80 端口，只要来源 IP 或来源端口不同，它们就属于不同连接
- 服务器必须区分这些连接 → 给每个都分配一个独立的 TCP 套接字

Web 服务器示例 (Apache、Nginx等) :

- 每个 HTTP 请求都会建立一个 TCP 连接 (除非是持久连接)
- 所以：
 - **非持久 HTTP**: 一次连接一个套接字
 - **持久连接** (HTTP/1.1 开始支持) : 可复用 TCP 连接处理多个请求

多路复用和多路分解小结

多路复用和多路分解小结

- 多路复用、多路分解：基于报文段、数据报头部字段值
- **UDP**：仅使用目的端口号进行多路分解
- **TCP**：使用四元组进行多路分解：源、目的IP地址及端口号
- 多路复用/多路分解发生在**所有层**

下面关于源端口号和目的端口号的描述中，正确的是（ ）。

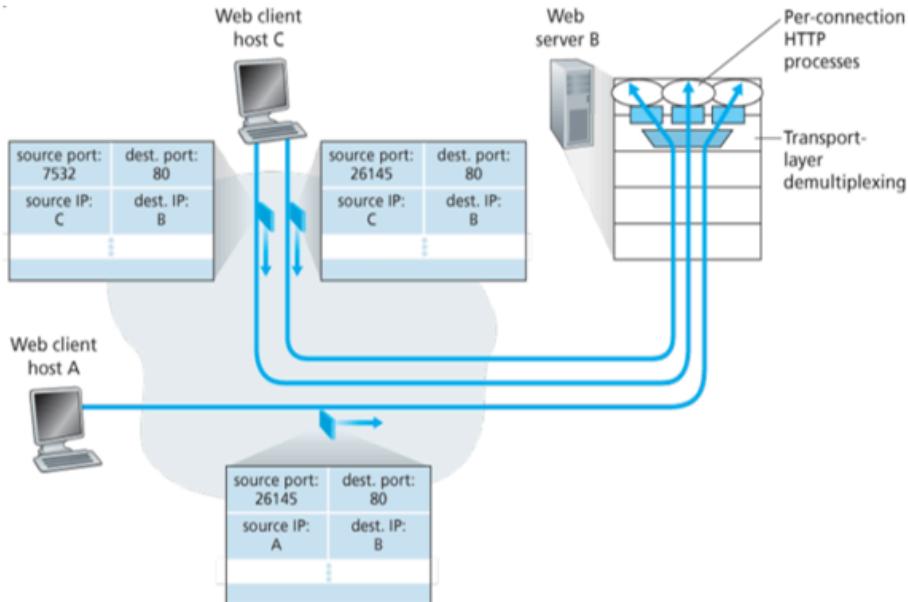
- A 在TCP/UDP报文中，源端口号和目的端口号不能相同
- B 在TCP/UDP报文中，源端口号和目的端口号可以相同，用来表示发回给自己的数据
- C 在TCP/UDP报文中，源端口号和目的端口号可以相同，因为虽然端口号一样，但其所在的主机IP地址会不同
- D 以上描述均不正确

C:

真正的因果关系是：

源端口号和目的端口号是否可以相同，**不是因为 IP 不同**，
而是因为 TCP/UDP 用的是“IP + 端口”的组合来唯一识别一个连接或进程。

考虑下图：从服务器返回客户进程的报文段中的源端口号和目的端口号是多少？
在承载运输层报文段的网络层数据报中，IP地址是多少（哪个主机的IP）？



在承载运输层报文段的网络层数据报中：

- “源 IP 地址是：服务器 B 的 IP”
- “目的 IP 地址是：客户端 A 的 IP”

● UDP 是“无连接”的

- 发送数据不需要建立连接，也不会维持连接
- 接收时只需要知道“这包数据是给哪个进程的”
- 所以只看：**目的 IP + 目的端口** 就够了

服务器只要有一个进程监听某个端口，比如 53 (DNS)，
收到任何发给 IP=B、端口=53 的数据，都交给 DNS 应用处理就行，**不需要区分谁发来的**。

无连接传输: UDP

UDP 只在 IP 的数据报服务之上增加了很少一点的功能，即端口的功能和差错检测的功能。

特点

命题追踪 ► UDP 的特点 (2014)

- 1) UDP 无须建立连接。因此 UDP 不会引入建立连接的时延。
- 2) 无连接状态。TCP 需要在端系统中维护连接状态。此连接状态包括接收和发送缓存、拥塞控制参数和序号与确认号的参数。而 UDP 既不维护连接状态，又不跟踪这些参数。因此，当某些专用服务器使用 UDP 时，一般都能支持更多的活动客户机。
- 3) UDP 的首部开销小。TCP 有 20B 的首部开销，而 UDP 仅有 8B 的开销。
- 4) UDP 没有拥塞控制，因此网络中的拥塞不会影响源主机的发送速率。某些实时应用要求源主机以稳定的速率发送数据，能容忍一些数据的丢失，但不允许有太大的时延。
- 5) UDP 支持一对一、一对多、多对一和多对多的交互通信。

在 UDP 上实现可靠传输

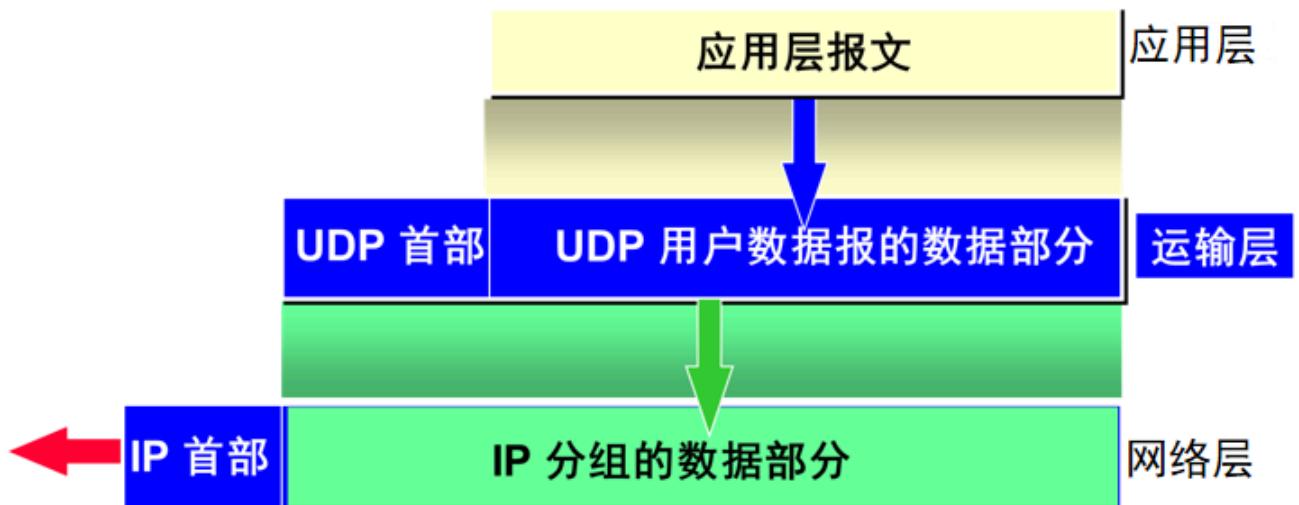
- 在应用层增加可靠性
- 在应用层增加拥塞控制

UDP是面向报文的

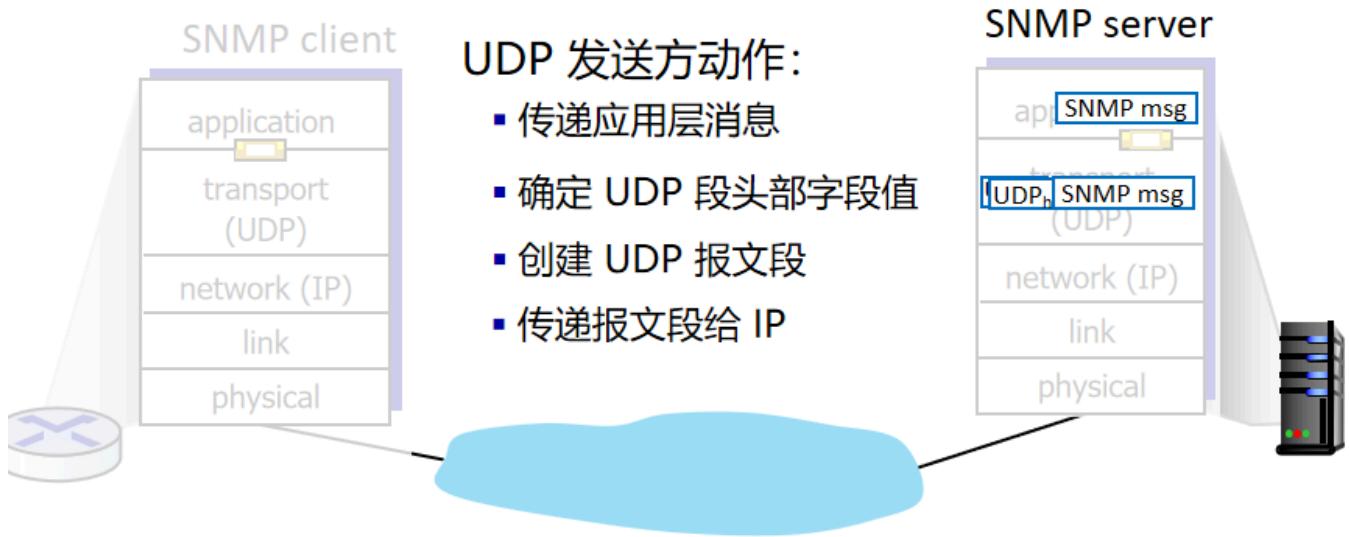
UDP 是面向报文的。发送方 UDP 对应用层交下来的报文，在添加首部后就向下交付给 IP 层，一次发送一个报文，既不合并，又不拆分，而是保留这些报文的边界；接收方 UDP 对 IP 层交上来 UDP 数据报，在去除首部后就原封不动地交付给上层应用进程，一次交付一个完整的报文。因此报文不可分割，是 UDP 数据报处理的最小单位。因此，应用程序必须选择合适大小的报文，若报文太长，则 UDP 把它交给 IP 层后，可能导致分片；若报文太短，则 UDP 把它交给 IP 层后，会使 IP 数据报的首部的相对长度太大，两者都会降低 IP 层的效率。

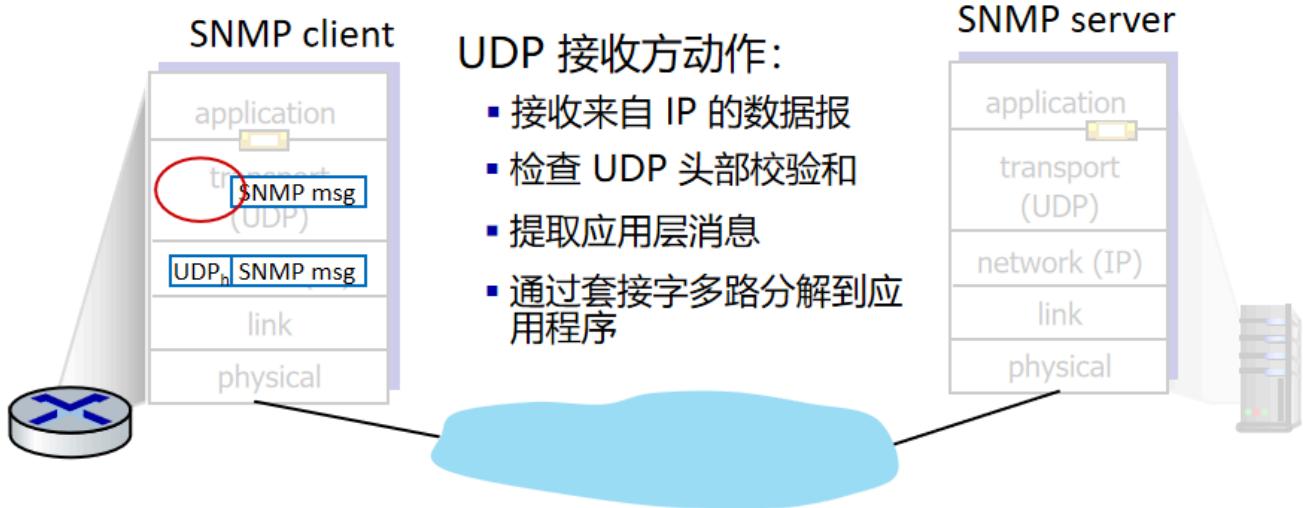
- UDP 的首部开销小，只有 8 个字节。
- UDP 是面向报文的。
 - 发送方 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。
 - 应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文。
 - 接收方 UDP 对 IP 层交上来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。
 - 应用程序必须选择合适大小的报文。

UDP 是面向报文的



UDP：传输层动作





🧠 整体流程总结口诀:

发送方:

应用生成报文 → UDP 加头封装 → IP 层发送

接收方:

IP 层收到 → UDP 解包校验 → 多路分解 → 应用层处理

UDP 首部格式

命题追踪 ► UDP 首部格式及各字段意义 (2018)

- 1) 源端口。源端口号。在需要对方回信时选用，不需要时可用全 0。
- 2) 目的端口号。这在终点交付报文时必须使用到。

命题追踪 ► UDP 首部的长度 (2021)

- 3) 长度。UDP 数据报的长度（包括首部和数据），其最小值是 8（仅有首部）。
- 4) 检验和。检测 UDP 数据报在传输中是否有错。有错就丢弃。该字段是可选的，当源主机不想计算检验和时，则直接令该字段为全 0。

当传输层从 IP 层收到 UDP 数据报时，就根据首部中的目的端口，把 UDP 数据报通过相应的端口，上交最后的终点——应用进程，如图 5.3 所示。



图 5.2 UDP 数据报格式

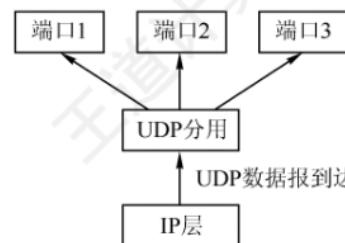
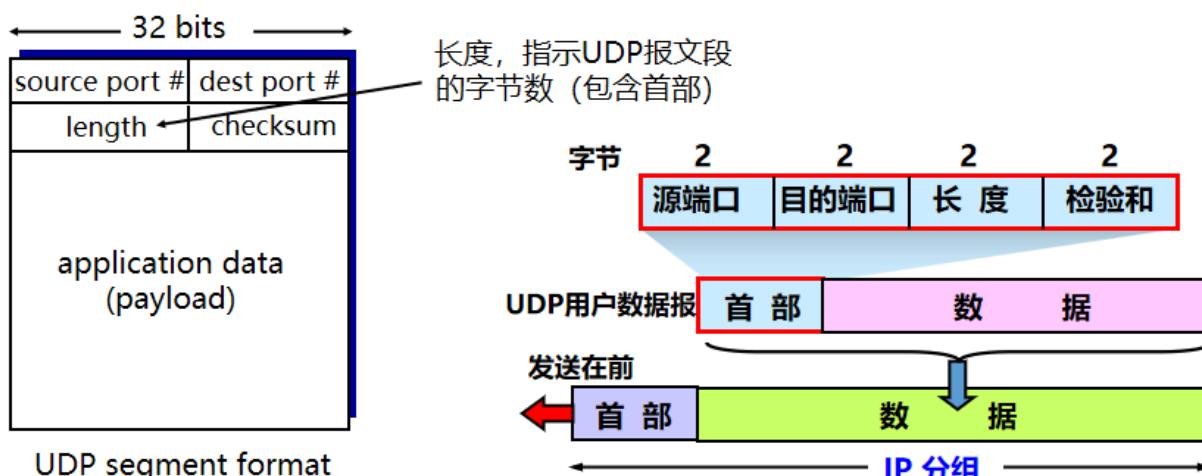


图 5.3 UDP 基于端口的分用

若接收方 UDP 发现收到的报文中的目的端口号不正确（不存在对应于端口号的应用进程），则就丢弃该报文，并由 ICMP 发送“端口不可达”差错报文给发送方。



用户数据报 UDP 有两个字段：**数据字段**和**首部字段**。首部字段有 8 个字节，由 4 个字段组成，每个字段都是两个字节。长度是首部和数据的总长度，故最小值为 8。

UDP 校验和

目标：对传输的报文段进行差错检测



工作流程：

发送方：

1. 把数据看作一系列 **16位整数**
2. 所有字段**按位相加** (1 的补码和)
3. **求反码**作为校验和 (即补 1)
4. 把这个值放进 UDP 报文段的**校验和字段**

接收方：

1. 再次将收到的所有字段 (包括收到的校验和字段) 相加
2. 结果应为全 1 (二进制全是 1, 说明无错)
3. 若不为全 1 → 说明报文有问题 (出错)

详细步骤

发送方:

【1】先把全 0 放入校验和字段

原因：为了计算时不让“校验和值自己影响自己”，所以先用 0 占位。

【2】将 UDP 报文 (首部 + 数据) 看成由许多 16位的整数 串联起来

- 所有内容都要按 每16位 (二进制) = 2字节为单位读取
- 报文不够偶数字节的，需要填充

【3】如果 UDP 报文数据部分不是偶数个字节

- 就在最后补一个全 0 字节
- 目的：凑成完整的 16 位整数，便于后面加法处理
- 注意：这个补 0 字节只是为了计算，不会实际发送！

【4】把所有字段 (首部 + 数据) 视为 16 位整数，做加法

- 用一的补码 (即普通加法 + 进位回卷) 方式累加所有字段
- 累加和得到一个 16 位的值

【5】对结果取反 (按位求反)

- 这就是校验和值 (checksum)
- 把它写入 UDP 首部的校验和字段

接受方：

- 【1】接收方收到报文后，把报文连同校验和字段一起拿来重新做一遍加法
 - 加法方式和发送方一样 (1的补码加法)
 - 如果原来报文没错，那么加上校验和值后，总和就应该是全 1 (二进制的 0xFFFF)
- 【2】若和不是全 1，说明数据在传输中发生了损坏
 - 接收方就应当丢弃这个 UDP 报文
- 【3】如果校验和验证成功 (和为全 1)
 - 说明该报文没有检测到错误，可以交给上层应用程序处理

例子 1：没有错误的数据包（校验成功）

发送数据：

UDP 报文数据字段 = `0x1234` 和 `0xABCD`

步骤：

1. 加和： $0x1234 + 0xABCD = 0xBE01$

2. 求反码 (1 的补码)：

反码 = `0x41FE`

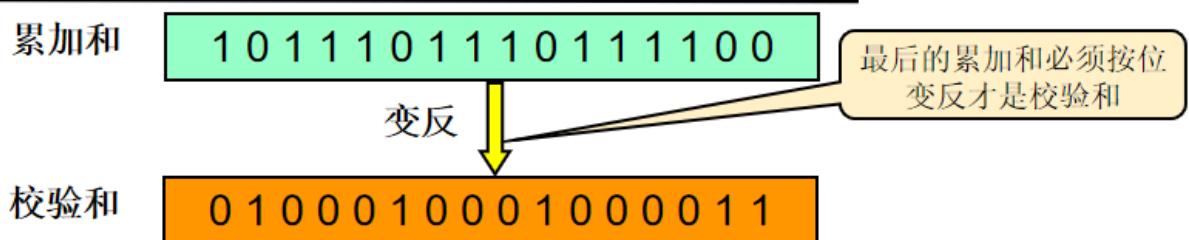
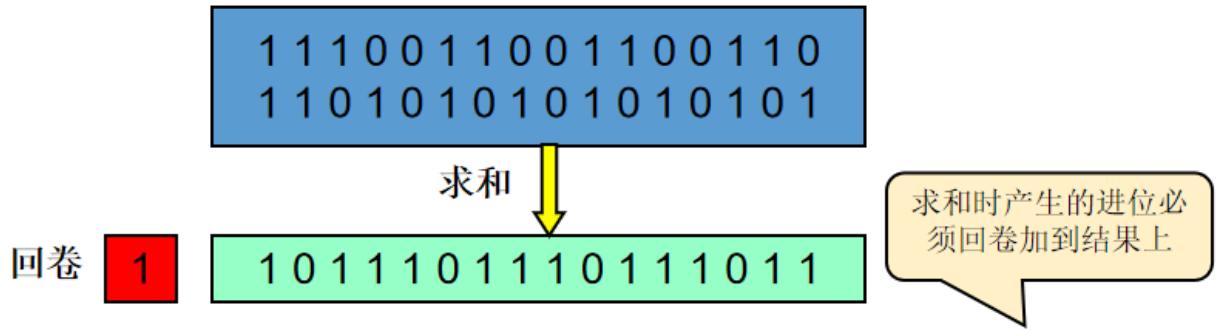
3. 发送方把 `0x41FE` 放入校验和字段

接收方验证：

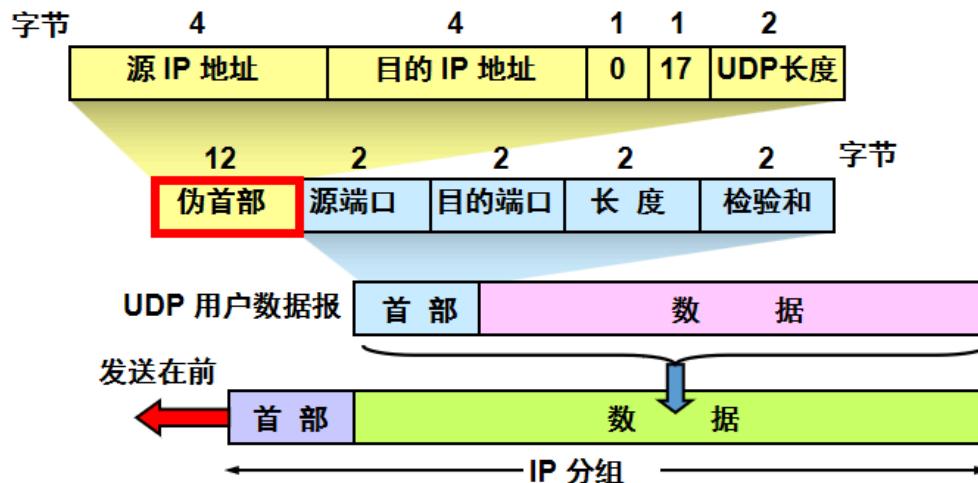
- 把 $0x1234 + 0xABCD + 0x41FE = 0xFFFF$  全 1，说明无错

UDP 的校验和机制 无法检测“同位互换”类的错误，它只检测“整体和变了没有”

Internet 校验和: 例子



在计算检验和时，临时把“伪首部”和 UDP 用户数据报连接在一起。伪首部仅仅是为了解释检验和。

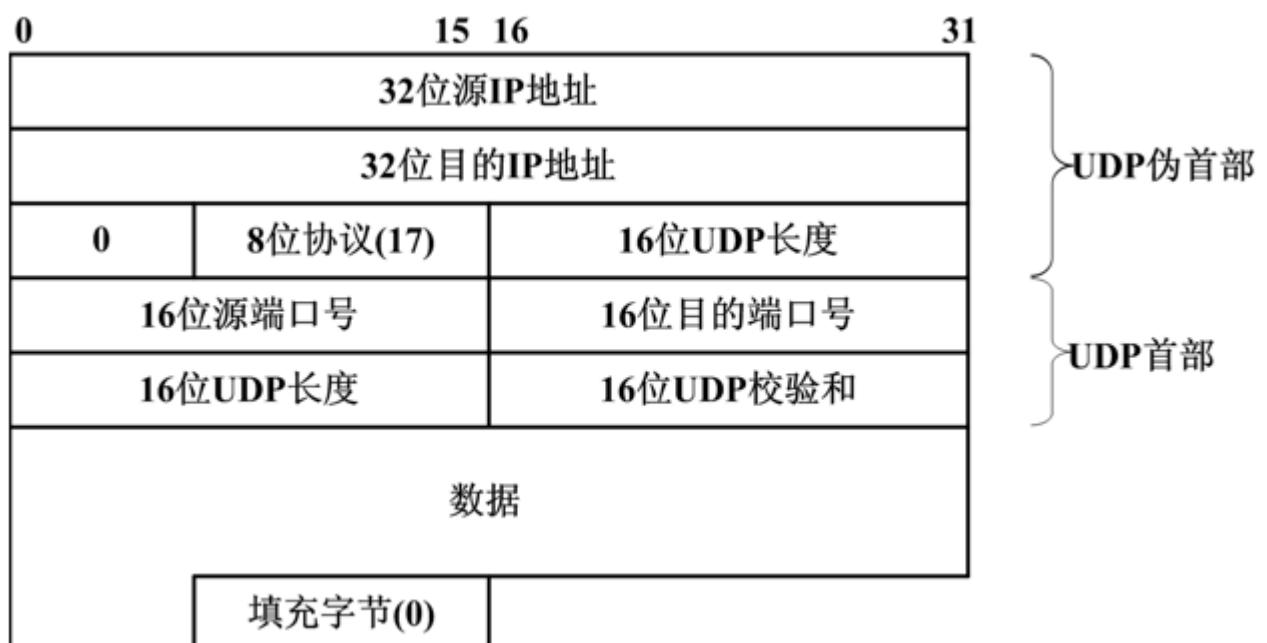


注：伪首部中的 UDP 长度，和首部中的长度一致

伪首部

图中伪首部结构详解 (共 12 字节)

字段	长度	说明
源 IP 地址	4字节	由 IP 层提供, 表示发送主机的 IP
目的 IP 地址	4字节	表示接收主机的 IP
保留字段 (全0)	1字节	恒为 0, 用于对齐
协议号	1字节	协议类型号, UDP 的值是 17 (0x11)
UDP 长度字段	2字节	UDP 报文的总长度 (首部 + 数据), 与 UDP 首部中该字段相同



计算 UDP 检验和实例

12 字节 伪首部	153.19.8.104				10011001 00010011 → 153.19
	171.3.14.11				00001000 01101000 → 8.104
	全 0	17	15		10101011 000000011 → 171.3
	1087	13			00001110 00001011 → 14.11
	15	全 0			00000000 00010001 → 0 和 17
	数据	数据	数据	数据	00000000 00001111 → 15
	数据	数据	数据	全 0	00000100 00111111 → 1087
					00000000 00001101 → 13
					00000000 00001111 → 15
					00000000 00000000 → 0 (检验和)
					01010100 01000101 → 数据
					01010011 01010100 → 数据
8 字节 UDP 首部	15	全 0			01001001 01001110 → 数据
	数据	数据	数据	全 0	01000111 00000000 → 数据 和 0 (填充)
填充					
按二进制反码运算求和 将得出的结果求反码					
10010110 11101101 → 求和得出的结果 01101001 00010010 → 检验和					

UDP为什么要用该和的反码，即为什么不直接使用该和？

理由如下：

1. 反码 = 让接收方更容易验证是否出错

- 因为当接收方把所有数据（含校验和）重新加起来时，**若没错结果就应该是全 1 (即 11111111)**
- 这种结果非常容易判断，一看是不是全 1 就知道有没有错

2. 使用反码能自动检测出多数单比特或短串错误

- 如果你用原码，加上去可能结果不等于特定值，不好判断

✓ 所以：

使用反码是为了使“校验成功条件非常清晰”（结果为全1）

不一定检测出

？ 2 比特错误呢？

⚠ 不一定检测得出！

- 如果两位错误“互相抵消”（比如一个加 1，一个减 1），就可能通过检验
- 所以 UDP 校验和**不能100%保证检错**，但对大部分传输错误是有效的

一个 UDP 用户数据报的首部十六进制表示是：

06 32 00 45 00 1C E2 17

- (1) 试求源端口、目的端口、用户数据报的总长度、数据部分长度。
- (2) 这个用户数据报是从客户发送给服务器还是服务器发送给客户？

数据部分长度 = 总长度 - 首部长度 = 28 - 8 == 20 字节**

看目的端口

UDP优点

- 协议结构简单，开销小

首部仅 8 字节，远小于 TCP (20 字节起)

没有连接建立/管理的额外负担

- 无需建立连接，低延迟

无需三次握手 (No handshake)

没有 RTT (往返时延) 开销

适合对实时性要求高的应用 (如语音、视频、DNS、游戏)

- 网络服务受到威胁时仍可工作

UDP 由于无连接、无状态、不重传、无窗口控制，即使网络受到攻击、拥塞或失效，也不易被拖死、能持续提供部分功能，这就是它“在受威胁下仍可工作”的根本原因。

- 有一定的可靠性（校验和）

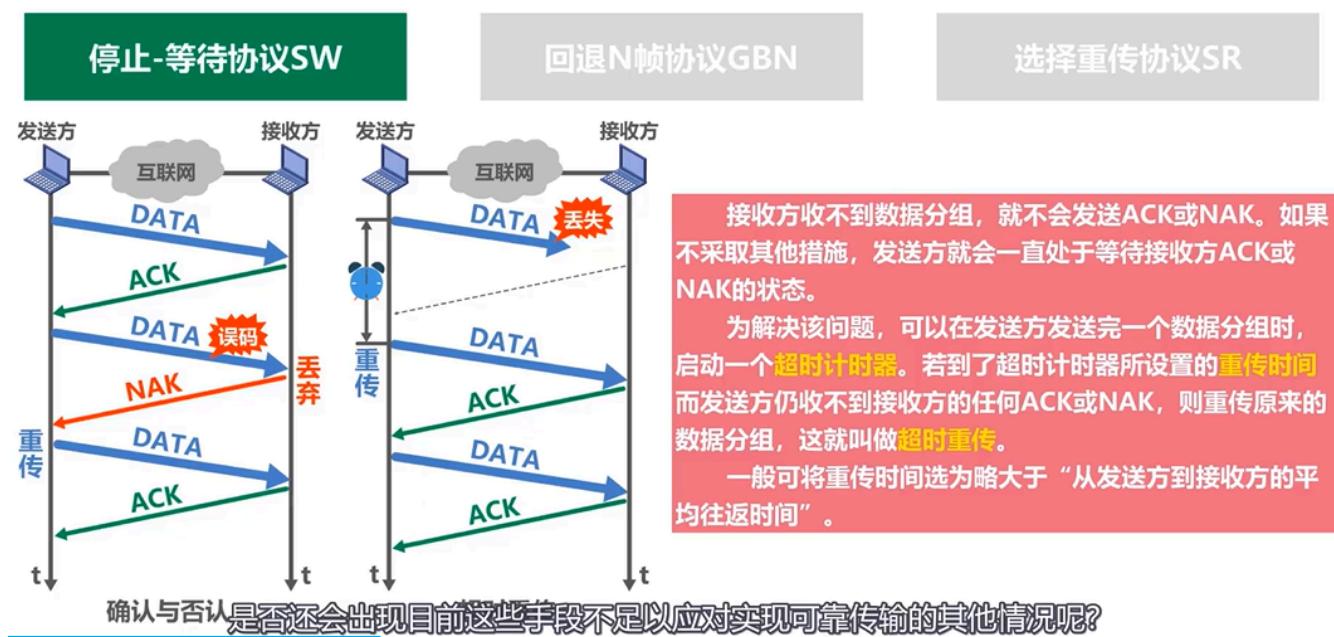
可靠数据传输原理

从应用角度看，可靠传输是“抽象通道”；
但实际上底层信道是不可靠的，
所以我们必须通过发送方和接收方共同运行的协议（rdt）+状态机建模，
来实现丢包重传、差错控制、顺序交付等机制，确保数据可靠到达。

实现机制

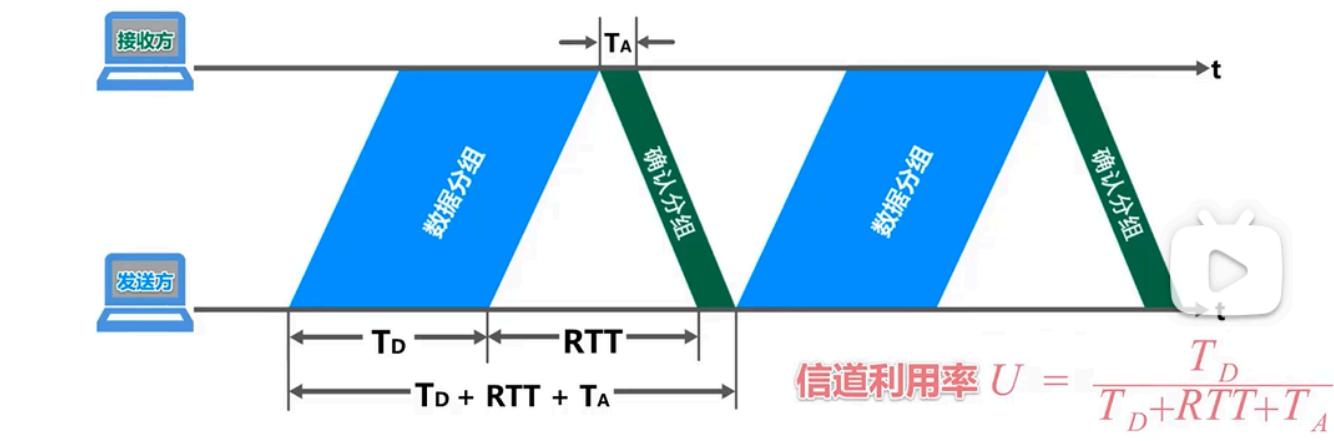
停止-等待协议SW

3.4.2 可靠传输的实现机制 —— 停止-等待协议SW(Stop-and-Wait)



信道利用率

停止-等待协议的信道利用率



停止-等待协议的信道利用率很低 若出现超时重传，则信道利用率更低

回退N帧协议GBN



3.4.3 可靠传输的实现机制——回退N帧协议GBN(Go-Back-N)

发送方

- 发送窗口尺寸 W_T 的取值范围是 $1 < W_T \leq 2^n - 1$
其中，n是构成分组序号的比特数量。
 - $W_T = 1$ 停止-等待协议
 - $W_T > 2^n - 1$ 接收方无法分辨新、旧数据分组
- 发送方可在未收到接收方确认分组的情况下，将序号落在发送窗口内的多个数据分组全部发送出去；
- 发送方只有收到对已发送数据分组的确认时，发送窗口才能向前相应滑动；
- 发送方收到多个重复确认时，可在重传计时器超时前尽早开始重传，由具体实现决定。
- 发送方发送窗口内某个已发送的数据分组产生超时重发时，其后续在发送窗口内且已发送的数据分组也必须全部重传，这就是回退N帧协议名称的由来。

这就是回退N帧协议名称的由来。

接收方

- 接收方的接收窗口尺寸 W_R 的取值范围是 $W_R = 1$
因此接收方只能按序接收数据分组。
- 接收方只接收序号落在接收窗口内且无误码的数据分组，并且将接收窗口向前滑动一个位置，与此同时给发送方发回相应的确认分组。为了减少开销，接收方不一定每收到一个按序到达且无误码的数据分组就给发送方发回一个确认分组，
 - 而是可以在连续收到好几个按序到达且无误码的数据分组后（由具体实现决定），才针对最后一个数据分组发送确认分组，这称为累积确认；
 - 或者可以在自己有数据分组要发送时才对之前按序接收且无误码的数据分组进行捎带确认；
- 接收方收到未按序到达的数据分组，除丢弃外，还要对最近按序接收的数据分组进行确认；

回退N帧协议在流水线传输的基础上利用发送窗口来限制发送方连续发送数据分组的数量，是一种连续ARQ协议。

Go-Back-N 所使用的技术

- 序号
- 累积确认
- 校验和
- 定时器（超时判断）
- 重传（超时重传）

选择重传协议SR

注意：

选择重传协议为了使发送方仅重传出现差错的分组，接收方不能再采用累积确认，而需要对每个正确接收到的数据分组进行逐一确认！

3.4.4 可靠传输的实现机制 —— 选择重传协议SR(Selective Request)

发送方	接收方
<ul style="list-style-type: none">■ 发送窗口尺寸W_T的取值范围是 $1 < W_T \leq 2^{n-1}$ 其中，n是构成分组序号的比特数量。<ul style="list-style-type: none"><input type="checkbox"/> $W_T = 1$ 与停止-等待协议相同<input type="checkbox"/> $W_T > 2^{n-1}$ 接收方无法分辨新、旧数据分组■ 发送方可在未收到接收方确认分组的情况下，将序号落在发送窗口内的多个数据分组全部发送出去；■ 发送方只有按序收到对已发送数据分组的确认时，发送窗口才能向前相应滑动；若收到未按序到达的确认分组时，对其进行记录，以防止其相应数据分组的超时重发，但发送窗口不能向前滑动。	<ul style="list-style-type: none">■ 接收窗口尺寸W_R的取值范围是 $1 < W_R \leq W_T$<ul style="list-style-type: none"><input type="checkbox"/> $W_R = 1$ 与停止-等待协议相同<input type="checkbox"/> $W_R > W_T$ 无意义■ 接收方可接收未按序到达但没有误码并且序号落在接收窗口内的数据分组；<ul style="list-style-type: none"><input type="checkbox"/> 为了使发送方仅重传出现差错的分组，接收方不能再采用累积确认，而需要对每个正确接收到的数据分组进行逐一确认！■ 接收方只有在按序接收数据分组后，接收窗口才能向前相应滑动。

选择性重传 Selective repeat

- 流水线传输（Pipelining）：同时有多个传输中的分组
- 接收方分别对所有正确接收的分组发送确认（ACK）
 - 根据需要缓存数据包，以便按顺序传递给上层协议
- 发送方：
 - （理论上）维护每个未确认分组的定时器
 - 超时：重新传输与超时相关联的单个未确认的分组
 - （理论上）维护对 N 个连续序列号的“窗口”
 - 限制流水线传输中“在途”数据包的数量在此窗口内

Q: 序列大小和窗口大小之间需要什么关系才可以避免场景 (b) 中的问题?

A: 窗口长度小于等于序号空间的一半!

JAVA FOR BEGINNERS

Selective Repeat 所使用的技术

- 序号
- 逐个确认 (接收方缓存)
- 校验和
- 定时器
- 重传 (超时重传、**选择重传**)

流水线协议：总结

GBN

- 发送方在管道中可以有多达 N 个未应答的分组
- 接收方只发送累积确认 (*cumulative ack*)
 - 表明接收方已正确接收到序号为 N 及以前的所有分组
- 发送方对最早未确认的分组进行定时器计时
 - 当定时器超时，重新发送所有未确认的分组

SR

- 发送方在管道中可以有多达 N 个未应答的分组
- 接收方为每个分组发送确认 (*individual ack*)
- 发送方对每一个未确认的分组维护一个定时器
 - 当定时器超时，只发送未确认的分组

面向连接的传输TCP

TCP报文段

5.3.2 TCP 报文段

TCP 传送的数据单元称为报文段。TCP 报文段既可用来运载数据，又可用来建立连接、释放连接和应答。一个 TCP 报文段分为首部和数据两部分，整个 TCP 报文段作为 IP 数据报的数据部分封装在 IP 数据报中，如图 5.5 所示。其首部的前 20B 是固定的。TCP 首部最短为 20B，后面有 $4N$ 字节是根据需要而增加的选项，长度为 4B 的整数倍。

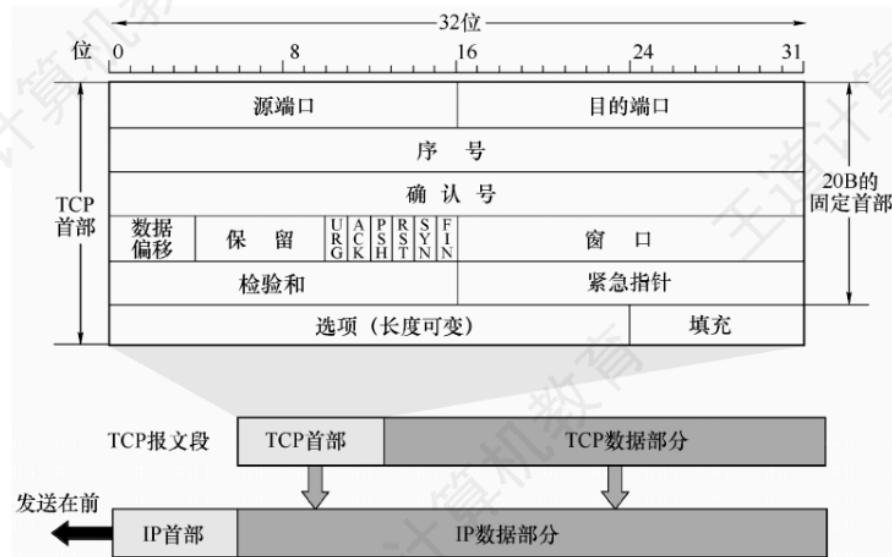
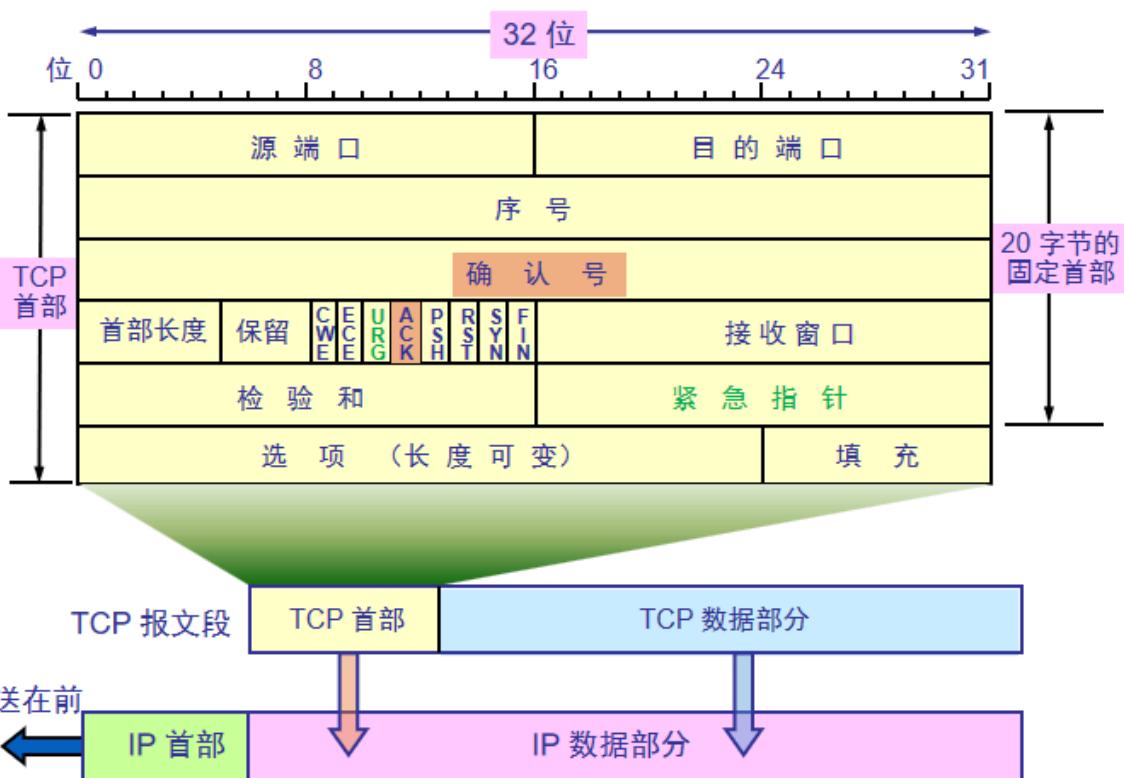


图 5.5 TCP 报文段



源端口目的端口各占2B 端口是运输层与应用层的服务接口。运输层的复用和分用功能通过端口实现。

命题追踪**TCP 首部中序号、确认号的含义 (2009、2016)**

2) 序号。占 4B，范围为 $0 \sim 2^{32} - 1$ ，共 2^{32} 个序号。TCP 连接中传送的字节流中的每个字节都要按顺序编号，序号字段值指的是本报文段所发送的数据的第一个字节的序号。

例如，一报文段的序号字段值是 301，而携带的数据共有 100B，表明本报文段的数据的最后一个字节的序号是 400，因此下一个报文段的数据序号应从 401 开始。

3) 确认号。占 4B，是期望收到对方下一个报文段的第一个数据字节的序号。若确认号为 N ，则表明到序号 $N - 1$ 为止的所有数据都已正确收到。

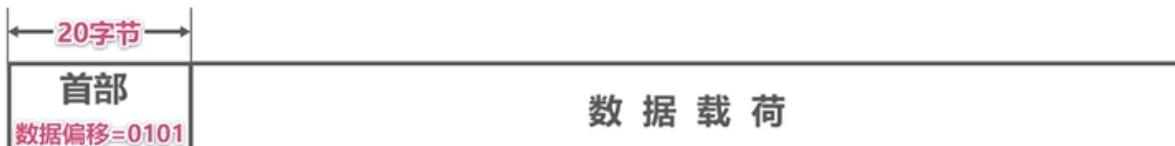
例如，B 正确收到了 A 发送过来的一个报文段，其序号字段是 501，而数据长度是 200B（序号 501~700），这表明 B 正确收到了 A 发送的到序号 700 为止的数据。因此 B 期望收到 A 的下一个数据序号是 701，于是 B 在发送给 A 的确认报文段中把确认号置为 701。

首部长度：占 4 位，指出 TCP 报文段的首部长度。单位是 32 位字（以 4 字节为计算单位）。由于 TCP 报文段包含选项，其首部长度可变。

这个字段实际上是指出 TCP 报文段的首部长度。

首部固定长度为 20 字节，因此数据偏移字段的最小值为 $(0101)_2$

首部最大长度为 60 字节，因此数据偏移字段的最大值为 $(1111)_2$



保留：占 6 位，保留为今后使用，但目前应置为 0。

CWR(Congestion Window Reduced)：CWR 标志与后面的 ECE 标志用于 IP 首部的 ECN 字段。ECE 标志为 1 时，则通知对方将拥塞窗口变小。

ECE(ECN-Echo)：ECE 标志表示 ECN-Echo。置为 1 会通知通信对方，从对方到这边的网络有拥塞。在收到数据包的 IP 首部中 ECN 为 1 时将 TCP 首部中的 ECE 设置为 1。

紧急 URG：控制位。当 URG = 1 时，表明紧急指针字段有效，告诉系统此报文段中有紧急数据，应尽快传送（相当于高优先级的数据）。

确认 ACK：控制位。只有当 ACK = 1 时，确认号字段才有效。当 ACK = 0 时，确认号无效。

推送 PSH (PuSH)：控制位。接收 TCP 收到 PSH = 1 的报文段后，就尽快（即“推送”向前）交付接收应用进程，而不再等到整个缓存都填满后再交付。

复位 RST (ReSeT)：控制位。当 RST = 1 时，表明 TCP 连接中出现严重差错（如主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。

RST置1还用来拒绝一个非法的报文段或拒绝打开一个TCP连接。

同步 SYN (SYNchronization)：控制位。同步 $SYN = 1$ 表示这是一个连接请求或连接接受报文。当 $SYN = 1$, $ACK = 0$ 时，表明这是一个连接请求报文段。当 $SYN = 1$, $ACK = 1$ 时，表明这是一个连接接受报文段。

终止 FIN (FINish)：控制位。用来释放一个连接。 $FIN=1$ 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

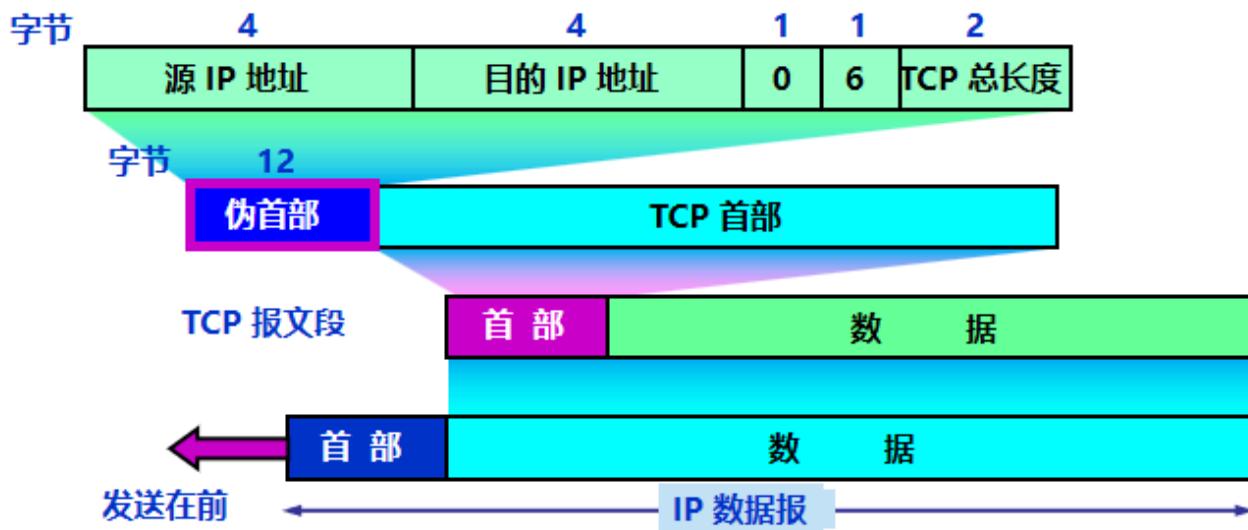
窗口：占 2 字节。窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量（以字节为单位）。

窗口字段明确指出了现在允许对方发送的数据量。窗口值经常在动态变化。

例如，设确认号是 701，窗口字段是 1000。这表明，从 701 号算起，发送此报文段的一方还有接收 1000 字节数据（字节序号为 701~1700）的接收缓存空间。

检验和：占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

一样，要在 TCP 报文段的前面加上 12B 的伪首部（只需将 UDP 伪首部的协议字段的 17 改成 6，UDP 长度字段改成 TCP 长度，其他的和 UDP 一样）。



紧急指针：占 2 字节。在 $URG = 1$ 时，指出本报文段中的紧急数据的字节数（紧急数据结束后就是普通数据），指出了紧急数据的末尾在报文段中的位置。

当发送方有紧急数据时，可将紧急数据插队到发送缓存的最前面，并立刻封装到一个TCP报文段中进行发送。紧急指针会指出本报文段数据载荷部分包含了多长的紧急数据，紧急数据之后是普通数据。

填充：使整个 TCP 首部长度是 4 字节的整数倍。

选项。长度可变，最长可达 40B。当不使用选项时，TCP 首部长度是 20B。TCP 最初只规定了一种选项，即最大报文段长度（Maximum Segment Size, MSS），它是 TCP 报文段中的数据字段的最大长度。之后又陆续增加了窗口扩大、时间戳等选项，具体请参见教材。

MSS (Maximum Segment Size)

是 TCP 报文段中的数据字段的最大长度。

数据字段加上 TCP 首部才等于整个的 TCP 报文段。

所以，MSS 是“TCP 报文段长度减去 TCP 首部长度”。

高级

选项 (2) : 最大报文段长度 MSS

- **最大报文段长度 MSS (Maximum Segment Size) 是每个 TCP 报文段中的数据字段的最大长度。**
- **与接收窗口值没有关系。**



$$\text{TCP 报文段长度} = \text{数据字段长度} + \text{TCP 首部长度}$$

$$\text{数据字段长度} = \text{TCP 报文段长度} - \text{TCP 首部长度}$$

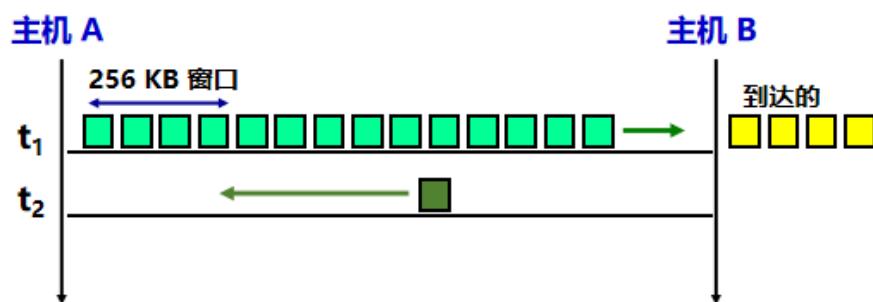
选项 (2) : 最大报文段长度 MSS

不能太小	不能太大	应尽可能大
<ul style="list-style-type: none">• 网络利用率降低。• 例如：仅 1 个字节。利用率就不会超过 $1/41$。	<ul style="list-style-type: none">• 开销增大。• IP 层传输时要分片，终点要装配。• 分片传输出错时，要整个分组。	<ul style="list-style-type: none">• 只要在 IP 层传输时不再分片。• 默认值 = 536 字节。<ul style="list-style-type: none">◆ 报文段长度 = $536 + 20 = 556$ 字节。◆ IP 数据报长度 = 576 字节。

- **最大报文段长度MSS选项**: TCP报文段数据载荷部分的最大长度。
- **窗口扩大选项**: 为了扩大窗口 (提高吞吐率) 。
- **时间戳选项**:
 - 用来计算往返时间RTT
 - 用于处理序号超范围的情况，又称为防止序号绕回PAWS。

■ 选择确认选项

选项 (3) : 窗口扩大



窗口扩大选项: 占 3 字节，其中一个字节表示移位值 S 。

新的窗口值位数从 16 增大到 $(16 + S)$ ，相当于把窗口值向左移动 S 位。

移位值允许使用的**最大值是 14**，窗口最大值增大到 $2^{(16 + 14)} - 1 = 2^{30} - 1$ 。

窗口扩大选项可以在双方初始建立 TCP 连接时进行协商。

选项 (8) : 时间戳

- 占 10 字节。最主要的 2 个字段：
时间戳值字段 (4字节) 和**时间戳回答字段** (4字节) 。
- 2 个主要功能：
 1. 计算往返时间 RTT
 2. 防止序号绕回 PAWS (Protect Against Wrapped Sequence numbers)。
 - 序号重复时，为了使接收方能够把**新报文段**和**迟到很久的旧报文段**区分开，可以在报文段中加上时间戳。

处理失序报文段？

Q: 接收方如何处理失序报文段？

A: TCP规范没有说明，由实现者自行选择实现：抛弃/缓存

1. 为什么在TCP首部中有一个首部长度字段，而UDP的首部中就没有这个字段？
2. 一个TCP报文段的数据部分最多有多少个字节？为什么？
3. 如果用户要传送的数据的字节长度超过TCP报文字段中的序号字段可能编出的最大序号，问还能否用TCP来传送？

因为 TCP 首部长度是可变的，而 UDP 首部长度是固定的。

最大为：65535（IP最大） - TCP首部长度

TCP 序号字段是 **32 位**，所以能表示的最大值为：

$$2^{32} = 4294967296 \approx 4GB$$

表面上只能给一个连接传送 4GB 数据？**错！**

可以

实际上：

TCP 序号是 **模 2^{32} 的循环序号**

也就是说：序号到 $2^{32}-1$ 后，再从 0 开始重新编号

Q: 如何设置 TCP 超时值？

- 比 RTT 长，但 RTT 是变化的！
- 太短：过早超时，产生不必要的重传
- 太长：对报文段的丢失响应太慢

Q: 如何估计 RTT？

- SampleRTT：从发送报文段到接收到ACK的测量时间
 - 忽略重传
- SampleRTT 会变化，希望估计的 RTT “较平滑”
 - 平均最近的多次测量值，并不仅仅是当前 SampleRTT

TCP 不直接使用单次 RTT 作为超时标准，而是通过平滑平均值 EstimatedRTT + 安全裕度 DevRTT × 4 来智能设定超时重传间隔，兼顾响应性和稳定性。

TCP 可靠数据传输机制

TCP 可靠数据传输

- TCP 在 IP 不可靠服务的基础上创建可靠数据传输服务
 - 流水线发送报文段
 - 累积确认
 - 使用单个重传计时器
- 重传触发事件
 - 超时事件
 - 重复 ACK

后面先考虑简化的 TCP 发送方：

- 忽略重复 ACK
- 忽略流量控制，拥塞控制

5.7 TCP可靠传输的实现

- 虽然发送方的发送窗口是根据接收方的接收窗口设置的，但在同一时刻，**发送方的发送窗口并不总是和接收方的接收窗口一样大。**
 - 网络传送窗口值需要经历一定的时间滞后，并且这个时间还是不确定的。
 - 发送方还可能根据网络当时的拥塞情况适当减小自己的发送窗口尺寸。
- 对于**不按序到达的数据应如何处理**，TCP并无明确规定。
 - 如果接收方把不按序到达的数据一律丢弃，那么接收窗口的管理将会比较简单，但这样做对网络资源的利用不利，因为发送方会重复传送较多的数据。
 - TCP通常对不按序到达的数据是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再**按序交付上层的应用进程**。
- TCP要求接收方必须有**累积确认和捎带确认机制**，这样可以减小传输开销。接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便捎带上。
 - 接收方不应过分推迟发送确认**，否则会导致发送方不必要的超时重传，这反而浪费了网络的资源。

TCP标准规定，确认推迟的时间不应超过0.5秒。若收到一连串具有最大长度的报文段，则必须每隔一个报文段就发送一个确认[RFC 1122]。
 - 携带确认实际上并不经常发生，因为大多数应用程序很少同时在两个方向上发送数据。
- **TCP的通信是全双工通信**。通信中的每一方都在发送和接收报文段。因此，每一方都有自己的发送窗口和接收窗口。在谈到这些窗口时，一定要弄清楚是哪一方的窗口。

5.7 TCP可靠传输的实现

【2011年 题40】主机甲与主机乙之间已建立一个TCP连接，主机甲向主机乙发送了3个连续的TCP段，分别包含300字节、400字节和500字节的有效载荷，第3个段的序号为900。若主机乙仅正确接收到第1个和第3个段，则主机乙发送给主机甲的确认序号是 **B**

- A. 300 B. 500 C. 1200 D. 1400

【解析】





第三张图：TCP接收方：ACK生成建议 (RFC 5681)

这张图是空的表格，应该是用于填写“接收方什么时候产生 ACK”：

我们补全一些常见的情况（你可以抄到表里）：

Event at receiver	TCP receiver action
收到一个按顺序的 segment	立即发送 ACK, ACK = 该段最后一个字节 + 1
收到一个乱序 segment (超前或延后)	立即重复发送对上一个有序段的 ACK
收到的数据段填满一个窗口	立即发送 ACK
没有新数据到达, 但有应用读取了数据	发送 ACK, 告知可用窗口变大

这些行为保证了 TCP 的 **可靠性和拥塞控制**。

快速重传

即向连接传输：TCP

TCP 快速重传

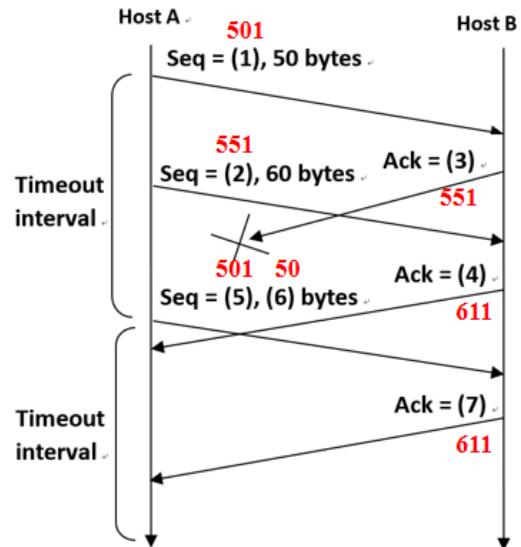
 University of Electronic Science and Technology

- 超时触发重传存在问题：
超时周期往往太长
 - 重传丢失报文之前要等待很长时间，因此增加了网络的时延
- 发送方可以在超时之前通过检测重复的 ACK 检测报文段丢失
 - 发送方常常一个接一个地发送很多报文段
 - 如果报文段丢失，则发送方将可能接收到很多重复的 ACKs
- 如果发送方收到 3 个对同样报文段的确认（在收到第一个确认后又收到 3 个 ACK），则发送方认为该报文段之后的数据已经丢失。
 - 启动**快速重传**：在定时器超时之前重发丢失的报文段

主机A和B通过一个TCP连接通信，且主机B已经收到了来自A的直到字节500的所有字节。假定主机A随后向B发送两个报文段。第一个报文段和第二个报文段分别包含了50和60字节的数据：

- 1、第一个报文段和第二个报文段的序号分别是(1)和(2)。
- 2、无论何时B收到A的报文段，它都会发送确认，确认号是(3)和(4)。
- 3、假定两个报文段按序到达B，第一个确认丢失，第二个确认在第一个超时间隔后到达A。故主机A需重新发送未收到确认的报文段，重新发送报文段的序号是(5)，字节数是(6)，待主机B收到该重新发送的报文段后，回复确认报文段，其确认号是(7)。

分别给出(1)~(7)中的数值。

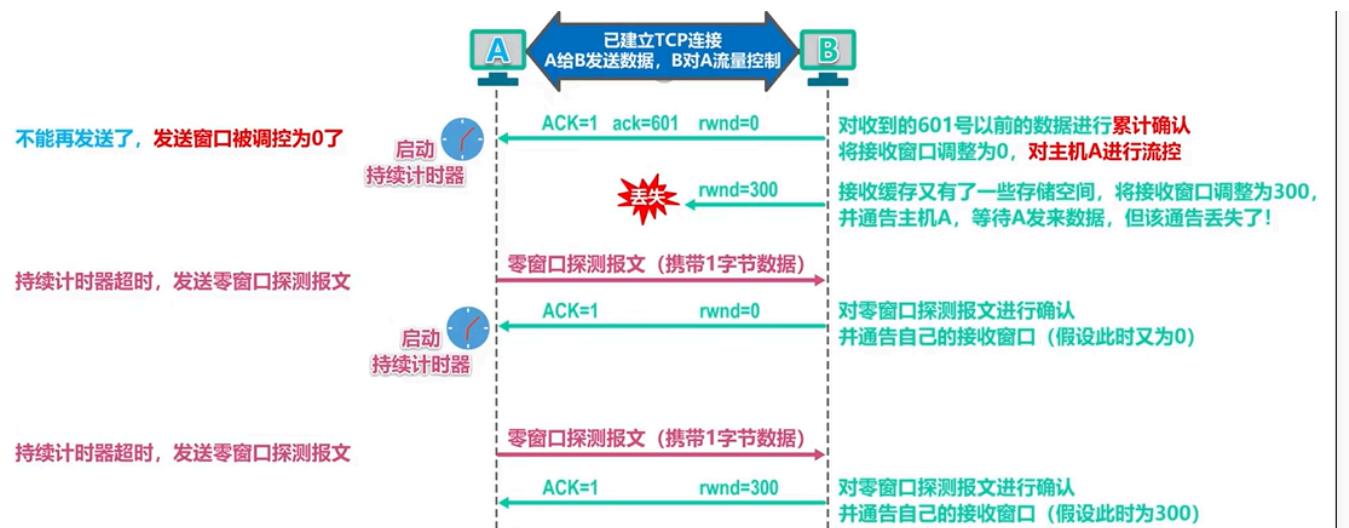


TCP流量控制

持续计时器



收到零窗口启动持续计时器，超时发送零窗口探测报文



5.4 TCP的流量控制

- 一般来说，我们总是希望数据传输得更快一些。
 - 但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。
- 所谓流量控制（flow control）就是让发送方的发送速率不要太快，要让接收方来得及接收。
- 利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。
 - TCP接收方利用自己的接收窗口的大小来限制发送方发送窗口的大小。
 - TCP发送方收到接收方的零窗口通知后，应启动持续计时器。持续计时器超时后，向接收方发送零窗口探测报文。

TCP连接管理

5.8.1 TCP的运输连接管理——TCP的连接建立

- TCP是面向连接的协议，它基于运输连接来传送TCP报文段。
- TCP运输连接的建立和释放是每一次面向连接的通信中必不可少的过程。

- TCP运输连接有以下三个阶段：



- TCP的运输连接管理就是使运输连接的建立和释放都能正常地进行。

TCP的运输连接管理就是使运输连接的建立和释放都能正常地进行。

两次握手建立连接

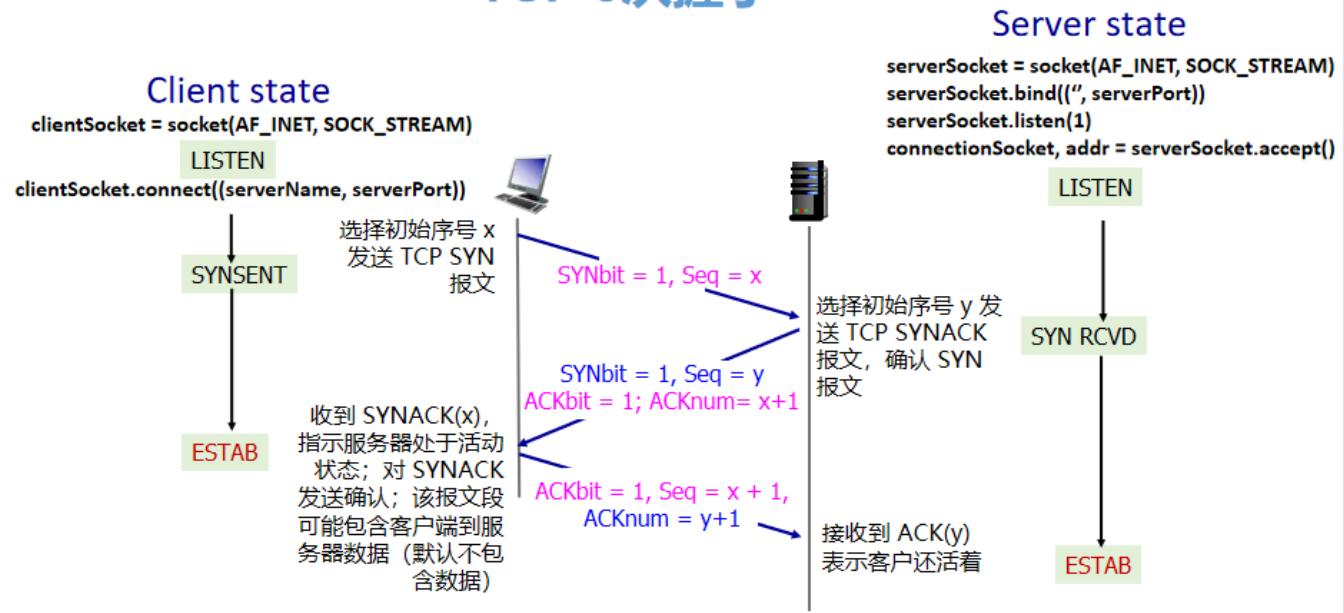
总结:

图示	内容	问题
图3	正常的2次握手连接	没有重传和丢包时是安全的
图4	半开连接	客户端已关闭但服务端还在线
图5	重复数据	服务端接收了旧连接重复数据

所以 TCP 使用三次握手是为了确保连接双方都能确认连接的唯一性与可靠性。

三次握手

TCP 3次握手



◆ Step 1 客户端 → 服务器：

- 发送 `SYN=1`, `SEQ=x`
 - 表示客户端想建立连接，起始序号为 x
-

◆ Step 2 服务器 → 客户端：

- 回复 `SYN=1, SEQ=y, ACK=1, ACKnum=x+1`
 - 表示服务器也要建立连接，并确认客户端的序号
-

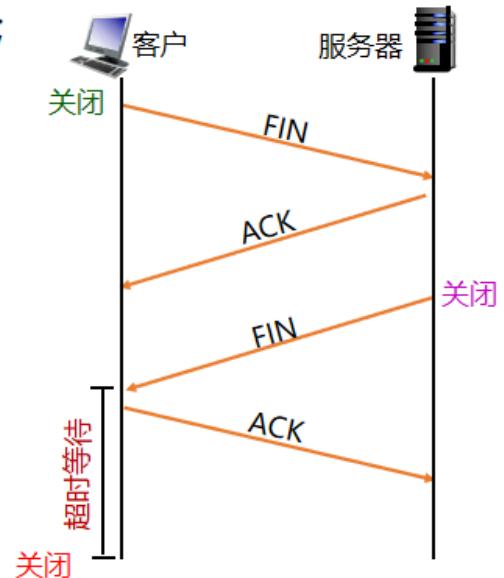
◆ Step 3 客户端 → 服务器：

- 回复 `SYN=0, SEQ=x+1, ACK=1, ACKnum=y+1`
- 表示确认服务器的连接，并可携带数据

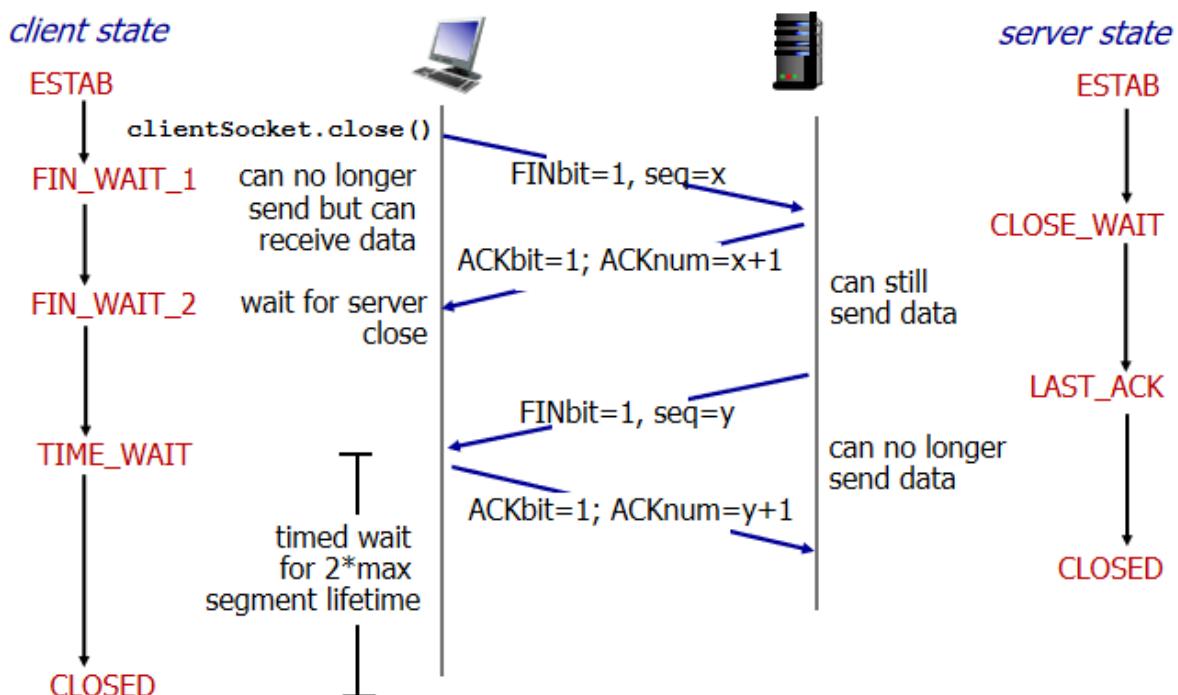
四次挥手

TCP关闭连接——4次挥手

- (1) 客户机向服务器发送TCP FIN控制报文段；
- (2) 服务器收到FIN，回复ACK确认，进入半连接状态；
- (3) 服务器关闭客户机的连接，发送FIN给客户机；
- (4) 客户机收到FIN，回复ACK确认
 - 进入“超时等待” - 将对收到的FIN进行确认
 - 服务器接收ACK，连接关闭



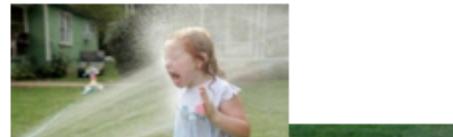
TCP关闭连接——4次挥手



拥塞控制原理

拥塞：

- 非正式地：“太多的源发送太多太快的数据，使网络来不及处理”
- 表现：
 - 丢包（路由器缓冲区溢出）
 - 长时延（路由器缓冲区中排队）
- 不同于流量控制！
- 网络中的前10大问题之一！



拥塞来源

多个主机共享同一链路，发送速率过高

拥塞表现

延迟显著上升，吞吐不再增长

拥塞代价

高延迟，低效传输

拥塞控制目标

维持 $\lambda_{in} \leq R/2$ ，防止队列无穷积累

- 延迟主要来自 排队时延 (queueing delay)



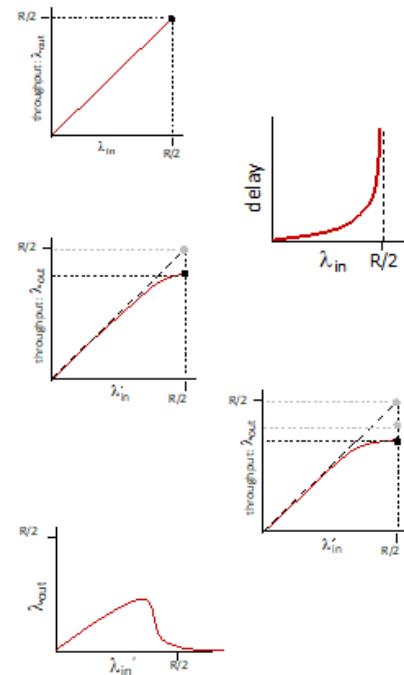
什么是网络吞吐量 (Throughput) ？

吞吐量指的是：

| 在单位时间内，从发送方成功传输到接收方的有效数据的总量。

拥塞的原因和代价

- 吞吐量永远不会超过容量
- 随着容量的临近，延迟增加
- 丢失/重传会降低有效吞吐量
- 不必要的重传会进一步降低有效吞吐量
- 由于下游分组丢失造成上游传输容量/缓冲的浪费



拥塞控制方法

端到端的拥塞控制

端到端的拥塞控制：

- 不从网络中得到明确的反馈
- 从端系统根据观察到的时延和丢失现象推断出拥塞
- **TCP所采用的方法**

网络辅助拥塞控制：

- 路由器利用数据流通过拥塞的路由器，向发送/接收主机提供直接反馈
- 可以指示拥塞级别或明确设置发送速率
- 指示发送方按照一定速率发送
 - TCP ECN, ATM, DECBIT protocols



TCP拥塞控制

AIMD 探测带宽

名称	中文含义	举例说明
AI	加性增加	每 RTT 加 1 个 MSS, 慢慢探测
MD	乘性减少	出现丢包就把窗口减半 (或降为1)
Reno	快速重传时减半	3 个重复 ACK \Rightarrow 减半
Tahoe	超时时直接归 1	超时 \Rightarrow 降为 1 MSS
目标	优雅地利用带宽, 避免拥塞风暴	

📌 图2: AIMD 减速细节 & 不同 TCP 实现

► 减速细节（乘性减少）分两种情况：

● TCP Reno:

| 检测到 3 个重复 ACK (意味着丢了一个包) , 窗口减半

- 不太严重，只是个别丢包，所以 **“减半”**就好

● TCP Tahoe:

| 若因 超时判断发生丢包 (表示网络严重拥塞) , 窗口直接减到 1 MSS

- 相当于重新开始探测网络

🧠 为什么使用 AIMD?

- AIMD 是一种分布式异步算法，**非常稳定和公平**
- 能在多个连接竞争同一条链路时，达到带宽利用的**全局优化**
- 理论上已经被证明在数学上有很好的稳定性

发送逻辑

✉ 二、TCP发送逻辑总结

text

✖ 复制 ✎ 编

发送 $cwnd$ 个字节，在一个 RTT 内等待 ACK，收到后继续发

公式如下：

$$\text{TCP 发送速率 (吞吐量)} \approx \frac{cwnd}{RTT} \text{ bytes/sec}$$

这意味着：

- $cwnd$ 越大 \Rightarrow 发得越多 \Rightarrow 吞吐量越高
- RTT 越大 \Rightarrow 一轮反馈越慢 \Rightarrow 吞吐量越低

✍ 三、发送方的发送条件（发送限制公式）

图中这一行：

text

`LastByteSent - LastByteAcked ≤ cwnd`

意思是：

当前未确认的字节数量（也叫“飞行中的字节数”）必须小于等于 $cwnd$

否则不能再发新数据！



四、如何判断网络出现拥塞？

发送方根据两个事件判断是否发生了拥塞：

1. 超时 (Timeout)

- 一段时间都没收到 ACK
- 表示网络严重阻塞或丢包

2. 重复 ACK 达到 3 次

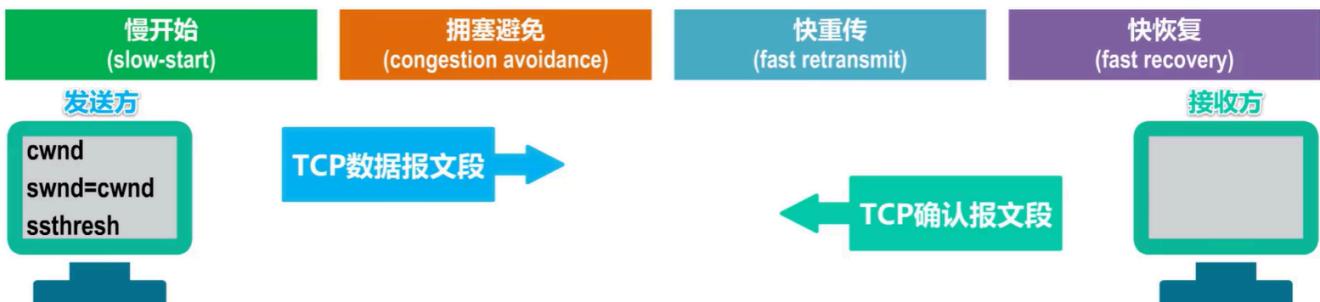
- 多次收到相同的 ACK
- 表示中间某个段丢了，后续段提前收到了
- 一般触发 **快速重传**

TCP 拥塞控制三阶段：

- 慢启动
- 拥塞避免
- i) • 快速恢复

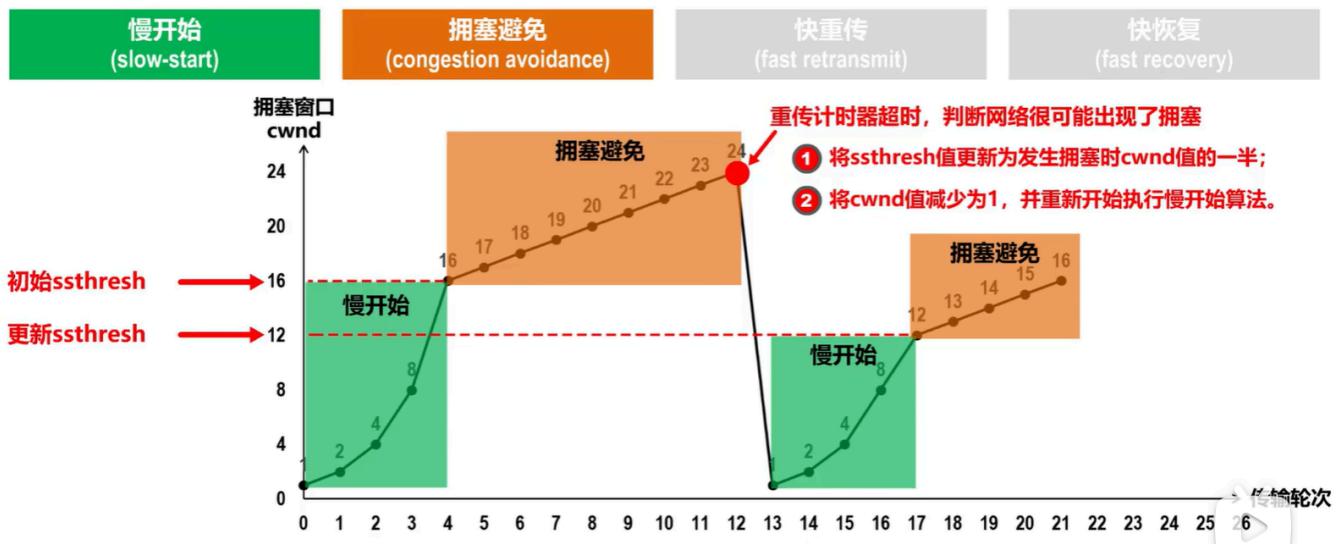
拥塞控制算法

5.5 TCP的拥塞控制



- 发送方维护一个叫做**拥塞窗口cwnd**的状态变量，其值**取决于网络的拥塞程度，并且动态变化**。
 - 拥塞窗口cwnd的维护原则：只要网络**没有出现拥塞**，拥塞窗口就再增大一些；但只要网络**出现拥塞**，拥塞窗口就减少一些。
 - 判断出现**网络拥塞**的依据：没有按时收到应当到达的确认报文（即**发生超时重传**）。
- 发送方将拥塞窗口作为**发送窗口swnd**，即**swnd = cwnd**。
- 维护一个慢开始门限**ssthresh**状态变量：
 - 当**cwnd < ssthresh**时，使用慢开始算法；
 - 当**cwnd > ssthresh**时，停止使用慢开始算法而改用拥塞避免算法；
 - 当**cwnd = ssthresh**时，既可使用慢开始算法，也可使用拥塞避免算法。

5.5 TCP的拥塞控制



- “慢开始”是指一开始向网络注入的报文段少，并不是指拥塞窗口cwnd增长速度慢；
- “拥塞避免”并非指完全能够避免拥塞，而是指在拥塞避免阶段将拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞；使网络比较不容易出现拥塞。

- 慢开始和拥塞避免算法是1988年提出的TCP拥塞控制算法（TCP Tahoe版本）。
- 1990年又增加了两个新的拥塞控制算法（改进TCP的性能），这就是快重传和快恢复（TCP Reno版本）。
 - 有时，个别报文段会在网络中丢失，但实际上网络并未发生拥塞。
 - 这将导致发送方超时重传，并误认为网络发生了拥塞；
 - 发送方把拥塞窗口 $cwnd$ 又设置为最小值1，并错误地启动慢开始算法，因而降低了传输效率。
- 采用快重传算法可以让发送方尽早知道发生了个别报文段的丢失。
- 所谓快重传，就是使发送方尽快进行重传，而不是等超时重传计时器超时再重传。
 - 要求接收方不要等待自己发送数据时才进行捎带确认，而是要立即发送确认；
 - 即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认。
 - 发送方一旦收到3个连续的重复确认，就将相应的报文段立即重传，而不是等该报文段的超时重传计时器超时再重传。

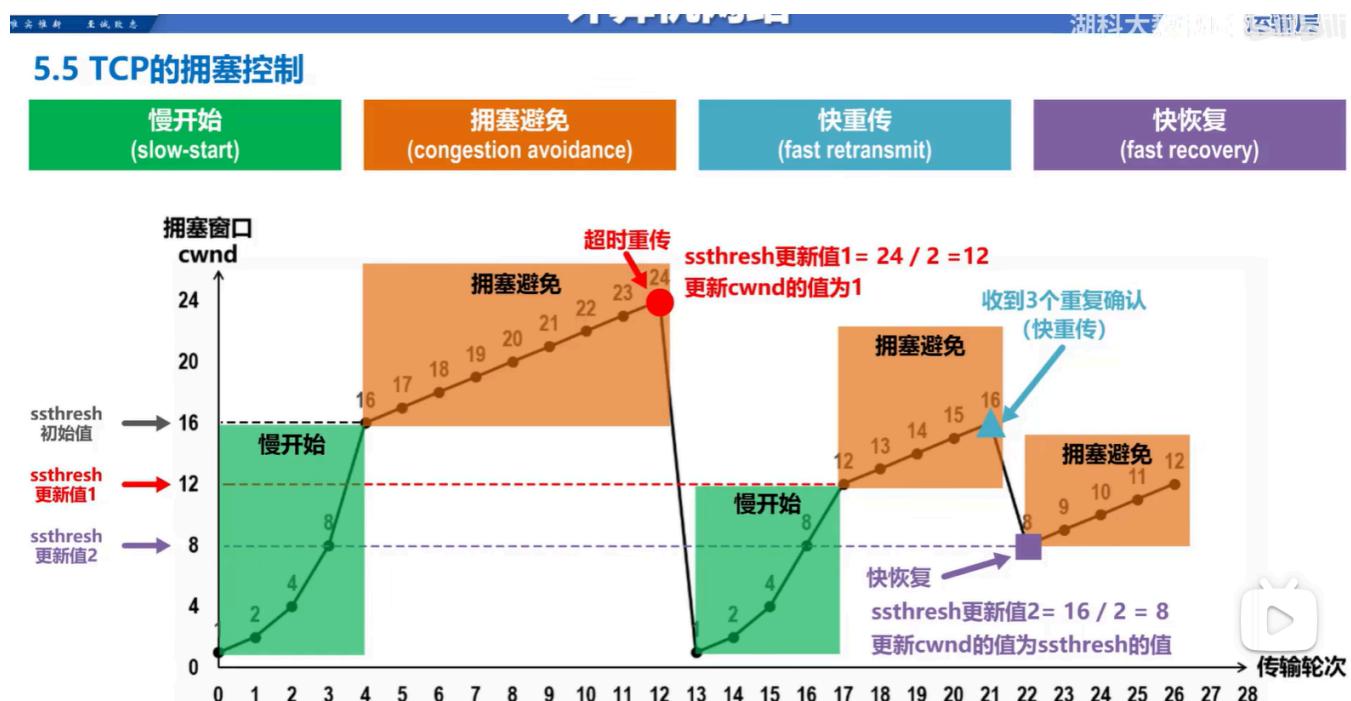
TCP从慢启动、拥塞避免切换到快速恢复

TCP从慢启动、拥塞避免切换到快速恢复

在慢启动或拥塞避免阶段收到三个重复ACK后，

$ssthresh = cwnd/2$, $cwnd = ssthresh + 3*MSS$, 进入快速恢复状态：

- 对于引起TCP进入快速恢复的缺失报文段，对收到的每个重复ACK，
 $cwnd = cwnd + 1*MSS$ ，依旧处于快速恢复状态
- 当收到丢失报文段的ACK（new ACK），降低 $cwnd$, $cwnd = ssthresh$ ，进入拥塞避免状态
- 当出现超时事件， $ssthresh = cwnd/2$, $cwnd = 1*MSS$ ，进入慢启动状态



总结: TCP 拥塞控制

- 初始阈值 $ssthresh = 64KB$ (或16个MSS), 初始窗口 $cwnd = 1$
- 当 $cwnd < ssthresh$ 时, 发送方处于慢启动阶段, $cwnd$ 指数增长
- 当 $cwnd \geq ssthresh$ 时, 发送方处于拥塞避免阶段, $cwnd$ 线性增长
- 当出现3个冗余ACK事件时
 - $ssthresh = cwnd/2$
 - $cwnd = ssthresh + 3*MSS$ (TCP Reno)
 - $cwnd = 1$ (TCP Tahoe)
 - 发方进入快速恢复阶段, 此时每收到一个重复的ACK, $cwnd$ 则增加 1MSS; 如果收到新的ACK, $cwnd = ssthresh$
- 当出现超时事件时, $ssthresh = cwnd/2$, $cwnd = 1$

🧠 拥塞窗口每轮变化原因总结:

阶段	特征	变化方式
慢启动	$cwnd < ssthresh$	每收到1个ACK, +1 MSS (指数增长)
拥塞避免	$cwnd \geq ssthresh$	每 RTT +1 MSS (线性增长)
超时	严重拥塞信号	$cwnd=1$, $ssthresh=cwnd/2$
拥塞恢复	回到慢启动 + 拥塞避免	同上逻辑继续增长

瓶颈链路

✓ 第1、2张图: 瓶颈链路与 TCP 拥塞形成

🌐 关键概念: 瓶颈链路 (Bottleneck Link)

- 是端到端路径中带宽最小、最容易拥塞的那一段链路
- TCP (如 Reno 或 CUBIC) 不断增加发送速率, 直到瓶颈路由器的缓冲区排满导致丢包

🔴 图中现象说明:

- TCP 不知道网络内部情况, 只能通过“丢包”来间接感知是否拥塞
- 红色瓶颈链路处: 队列经常非空, 有时还溢出丢包

图2 insight 点拨:

“increasing TCP sending rate will not increase end-to-end throughput with congested bottleneck”

意思是：

- 一旦瓶颈被“填满”，继续加速发数据也没用，只会增加排队延迟
- 此时吞吐量不会再提升，反而 RTT 变长

👉 **目标：**让“管道刚刚满就好，不要更满”（保持高效但低延迟）

基于延迟的 TCP 拥塞控制

背景：

传统 TCP (Reno/CUBIC) 靠丢包感知拥塞，效率不高；

Delay-based TCP 更“聪明”，它：

提前监测 RTT 增加 → 预测可能拥塞 → 主动减速，避免丢包

显式拥塞通知

(ECN, Explicit Congestion Notification)

核心机制：

传统 TCP 只有“丢包”才能知拥塞，现在网络支持 **主动标记拥塞**：

实现细节（图中）：

1. IP 报文头中有 2 个比特（ECN 字段）
 - 由路由器设置，比如设置为 `ECN=11` 表示**此路径拥塞**
 2. 接收方发现这个标记后，在 TCP ACK 报文中打上 `ECE=1` 返回给发送方
 3. 发送方收到 ECE 后，就知道网络拥塞，**触发减速**
-

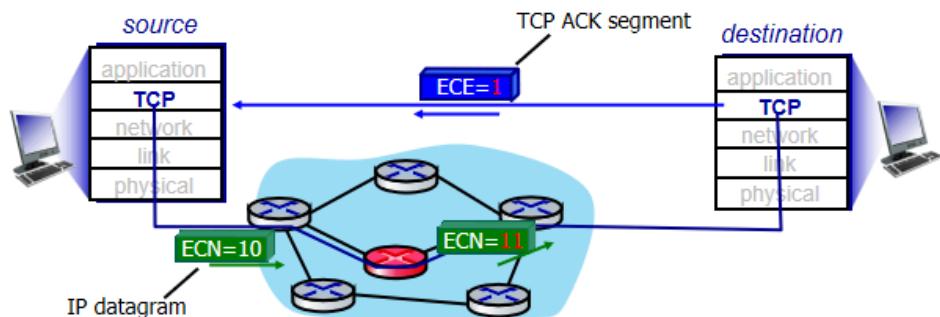
ECN 的优点：

- 不需要丢包也能感知拥塞；
- 响应更灵敏、更节能、更适合数据中心/低延迟应用

Explicit congestion notification (ECN)

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
 - policy to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



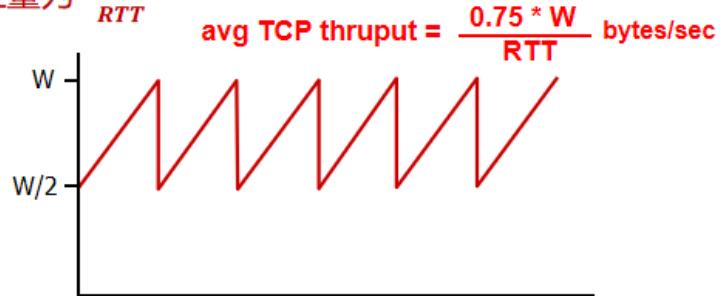
总结五图核心知识点：

主题	机制	特点
瓶颈链路	带宽最小/最容易拥塞的链路	TCP 拥塞关键点
传统 TCP	拥塞控制依赖丢包 (AIMD)	响应慢, 延迟高
Delay-based TCP	利用 RTT 提前预测拥塞	更平稳、低延迟, 如 TCP Vegas、BBR
ECN	网络主动打标, 告知拥塞	不靠丢包, 适合现代高速网络
CUBIC	TCP的一种增强型算法	Linux 默认 TCP, 适用于高带宽高延迟网络

TCP 平均吞吐量

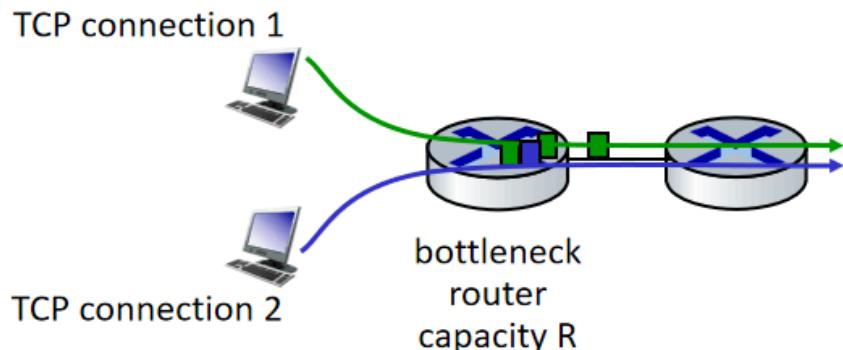
作为窗口长度和RTT的函数，TCP的平均吞吐量是什么？

- 假设忽略慢启动
- 设发生丢包时，窗口长度是W（字节）
- 如果窗口为 W，吞吐量是 $\frac{W}{RTT}$
- 当丢包发生后，窗口降为 $\frac{W}{2}$ ，吞吐量为 $\frac{W}{2RTT}$
- 一个连接的平均吞吐量为 $\frac{0.75W}{RTT}$



TCP公平性

公平目标：如果 K 个 TCP 会话共享带宽为 R 的瓶颈链路，每个会话应有 R/K 的平均链路速率



TCP 的 AIMD（加性增加、乘性减少）机制会自我调整，保证所有连接在长时间运行中趋于公平共享带宽。

四、网络层

虚电路和数据报网络

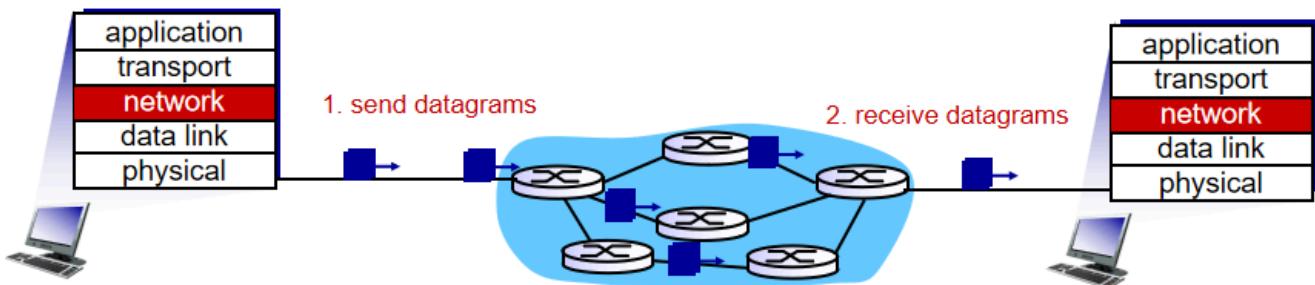
连接和无连接服务

- **数据报** 网络提供网络层的 **无连接** 服务
- **虚电路** 网络提供网络层的 **连接** 服务
- 类比于TCP/UDP的面向连接/ 无连接的传输层服务：
 - 任何网络中的网络层只提供两种服务之一，不会同时提供。
 - ✓ **虚电路网络**：提供**连接服务**。
 - ✓ **数据报网络**：提供**无连接服务**。
 - 传输层：面向连接服务在网络边缘的**端系统**中实现。
 - 网络层：面向连接服务在**端系统及网络核心的路由器**中实现。

数据报网络

数据报网络

- 在网络层无呼叫的过程
- 路由器：不需要维护端到端连接的状态
- 没有网络等级的“连接”的概念
- 使用目的主机的地址进行分组转发



数据报网络的特点

由互连计算机的需求发展而来。与电话网相反。

- 网络层服务模型简单。
- 端系统功能复杂：高层实现许多功能，如按序传送、可靠数据传输、拥塞控制与DNS名字解析等。
- 带来的结果
 - 因特网服务模型提供的服务保证最少（可能没有！），对网络层的需求最小，使得互连使用各种不同链路层技术的网络变得更加容易。
 - 许多应用都在位于网络边缘的主机（服务器）上实现。

路由器的工作原理

两个功能

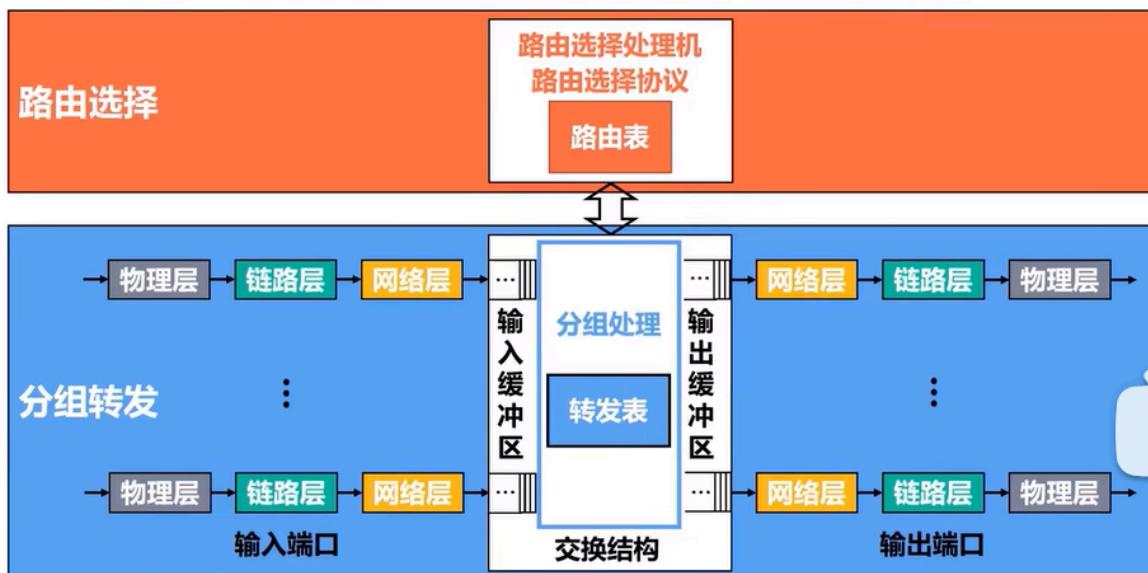
路由器的两个关键功能：

- 运行**路由**算法/协议(RIP, OSPF, BGP)
- **转发**功能：将分组从**路由器的输入链路**传送到适当的**输出链路**。

结构

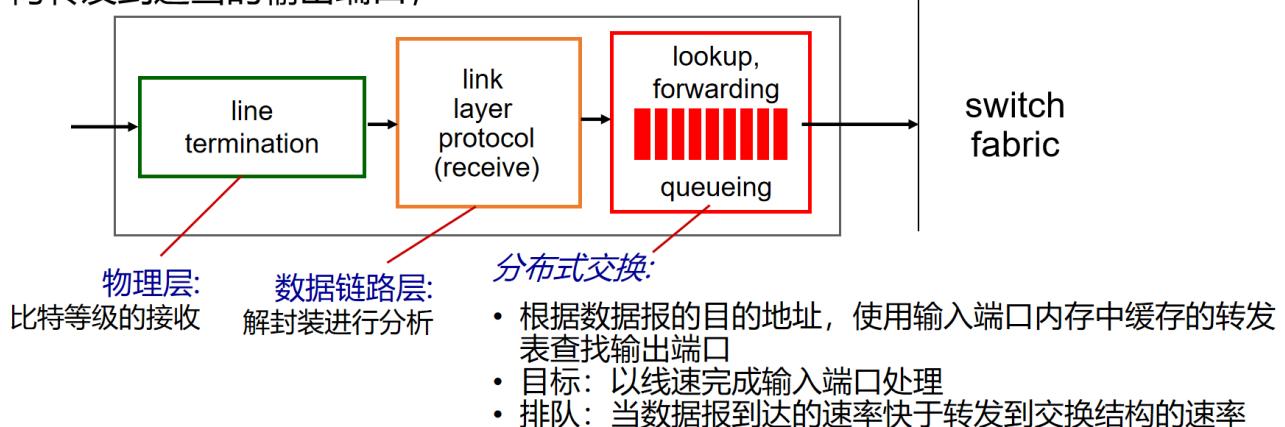
01 路由器的基本工作原理

■ 路由器是一种具有多个输入端口和输出端口的**专用计算机**，其任务是**转发分组**。



输入端口功能

- 第一个**线路端接**模块：将一条**物理链路端接到路由器的物理层**；
- 第二个**数据链路处理**模块：实现路由器的**数据链路层功能**；
- 第三个**查找与转发**模块：实现**查找与转发功能**，以便分组通过路由器交换结构转发到适当的输出端口；



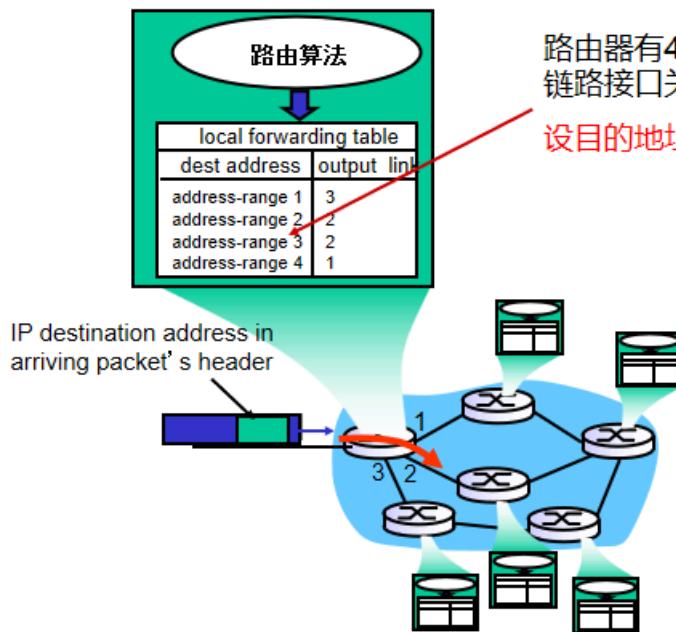
输入端口——查找/转发模块

确定**将一个到达的分组通过交换结构转发给哪个输出端口**。通过**查找转发表**实现，这里的转发表是存储在输入端口的内存中。

▪ 分布式交换：

- ✓ 选路处理器计算转发表，给**每个输入端口存放一份转发表拷贝**。
- ✓ 在**每个输入端口本地做出交换决策**，无须激活中央选路处理器。
- ✓ 可避免在路由器中某个**单点产生**转发处理瓶颈。
- ✓ 目的：以线速完成输入端口的处理
- ✓ 排队：如果数据报到达输入端口的速度快于输入端口将数据报转发到交换结构的速度，就会发生排队

转发表



路由器有4条链路 (0~3)，地址与链路接口关系如下：

设目的地址32位，40亿个可能的IP项

- **路由器转发方法：**根据到达分组的目的地址在转发表中查询，找到相应的输出链路接口，并将分组转发出去。
- **转发表：**每台路由器有一张。目的地址与链路接口的映射表。转发表中的表项数与地址位数有关，每个可能的地址对应一项。

数据报转发表

目的地址范围	链路端口
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

问题：如果地址范围分配没有很合理，会发生什么问题？

问题：如果地址范围分配不合理，会发生什么？

1. 路由表冲突：

- 两个地址段重叠，无法确定到底用哪个接口。
- 如果最长前缀匹配没设置好，可能选择错误的接口。

2. 匹配效率低下：

- 范围分布零散，路由表难以压缩，查表慢，占资源。

3. 浪费地址空间：

- 范围不连续导致地址“碎片化”，不利于聚合与管理。

4. 默认路由被频繁使用：

- 如果匹配不到，就会走默认路由（链路端口3），容易造成错误或网络拥堵。

最长前缀匹配

 **总结重点知识**

项目	含义
数据报转发表	路由器根据目的IP地址查找输出端口
最长前缀匹配	若多条路由可匹配，选匹配位数最多的 (最精确的)
路由表分配不合理的后果	匹配错误、冲突、效率低、地址浪费、默认路由被频繁使用等问题

◆ 为什么用 TCAM?

- **TCAM**: 三态内容可寻址存储器 (Ternary Content Addressable Memory)
 - 每一位可以存 0、1 或 x (don't care)
 - 可以一次并行匹配多个表项。
 - 匹配速度快，一个时钟周期内完成搜索，与表大小无关！
-

◆ 内容地址表 (Content Addressable Table)

- 传统内存是地址查值，而 TCAM 是“值查地址”。
- 举个形象的例子：
 - 普通内存：去图书馆找某本编号为 X 的书。
 - TCAM：告诉图书馆“我要找某本讲路由前缀为 11001000 的书”，图书馆一瞬间告诉你“在第几层第几排”。

数据报分组的传输与转发表维护机制

◆ 路由器的转发表：

- **转发表 (Forwarding Table)** 是路由器用于转发数据报的依据。
- 它只包含转发相关状态信息，比如：
 - 目的前缀 → 输出接口
 - 不负责做全网最短路径的计算。

◆ 路由算法更新转发表

- 路由算法（如 OSPF、RIP）周期性运行（通常 1~5 分钟），重新计算路径，更新转发表。
- 在虚电路网络中（如 ATM），还会因虚电路的建立和拆除动态更新。

◆ 分组可能走不同路径、乱序到达

一个终端系统发送给另一个终端系统的一批分组，可能会走网络中不同的路由路径。

- 这意味着：
 - 分组 A、B、C 虽然顺序发送出去，但到达顺序可能变成 C、A、B。
- 原因：
 - 网络中有多个路径可用，路由器可能根据拥塞情况等做不同的选择。

网络中有多个路径可用，路由器可能根据拥塞情况等做不同的选择。

交换结构

基本原理

核心概念

- **交换结构**：负责把输入端口缓存中的分组转发到合适的输出端口。
- **交换速率 (Switching Rate)**：衡量交换结构性能的指标，通常以“线速”的倍数表示。
 - 举例：如果有 N 个端口，每个都满速传输，则理想交换速率是 N 倍线速。

❖ 三种结构：

1. 内存交换结构 (memory) :

- 输入输出通过内存中转 → 由 CPU 控制。

2. 总线交换结构 (bus) :

- 多端口通过共享总线发送分组。

3. 交叉开关结构 (crossbar) :

- 每个输入可以通过矩阵直达输出, 支持并发、速度快。

早期交换结构--经内存

经内存的交换结构

- 早期用计算机作为路由器时采用的结构(第一代)
- 输入端口与输出端口之间的交换由CPU(选路处理器)控制完成
- 输入端口与输出端口类似I/O设备：
 - 当分组到达输入端口时, 通过中断向选路处理器发出信号, 将分组拷贝到处理器内存中;
 - 选路处理器根据分组中的目的地址查表找出适当的输出端口, 将该分组拷贝到输出端口的缓存中。

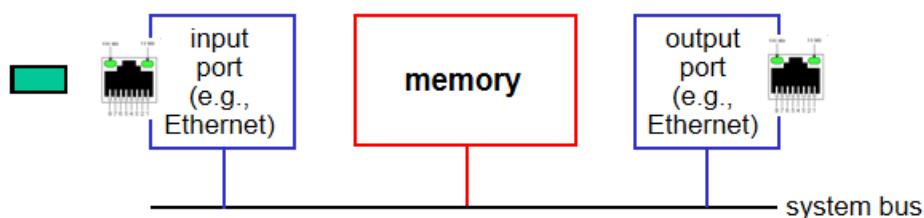




图3：早期的内存交换结构详解



内存交换结构细节

“”

- 这是最早期的做法，输入输出通过**中断 + CPU + 内存**来完成转发。
- 过程：
 - 输入端口接收分组。
 - 中断通知 CPU。
 - CPU 查表，找到目标输出端口。
 - 把数据从输入 → CPU → 内存 → 输出。

⚠ 缺点：速度慢，只能满足低速或教学实验环境。

内存结构的速率瓶颈



问题：

- 每个分组要进一次内存，再出一次内存 ⇒ 消耗两次内存带宽。
- 如果内存带宽是 B ，则**最大吞吐速率为 $B/2$** 。

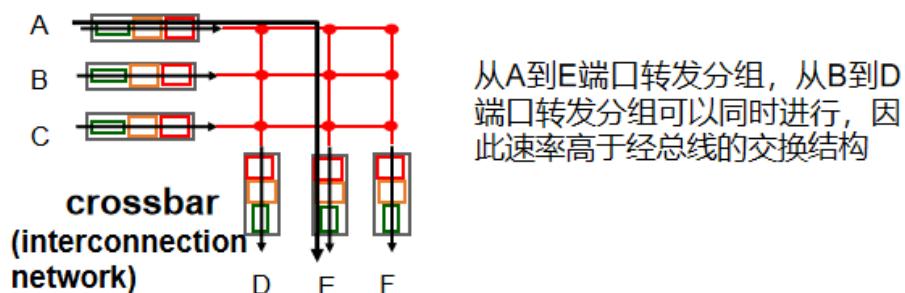
特点：

- 所有输入端口通过一条总线连接输出端口。
- 优点：不需要 CPU 控制，结构简单。
- 缺点：
 - 一次只能有一个分组通过总线。
 - 如果多个输入端口同时发 → 必须排队等待 → 出现瓶颈。

路由器交换带宽受总线速率限制。

经交换矩阵的交换结构

- 纵横式交换机：由 $2n$ 条总线组成， n 个输入端口与 n 个输出端口连接。
- 到达输入端口的分组沿水平总线穿行，直至与所希望的输出端口的垂直总线交叉点：
 - 若该条垂直总线空闲，则分组被传送至输出端口；
 - 否则，该到达的分组被阻塞，必须在输入端口排队。



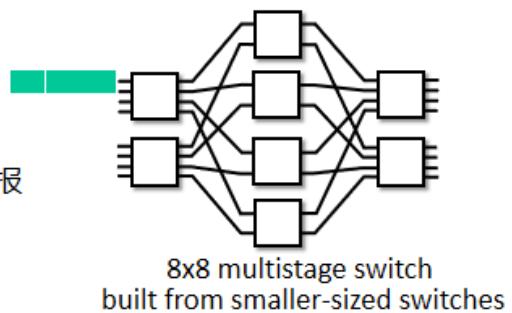
输入可以并发转发

- **纵横式交换机**: 由 $2n$ 条总线组成, n 个输入端口与 n 个输出端口连接。

- 多级交换 (multistage switch): nxn switch from multiple stages of smaller switches

- **高级设计: 利用并行性**

- 在进入时将数据报分段为固定长度的信元 (cell)
- 通过交换矩阵来交换信元, 在出口处重新组装数据报

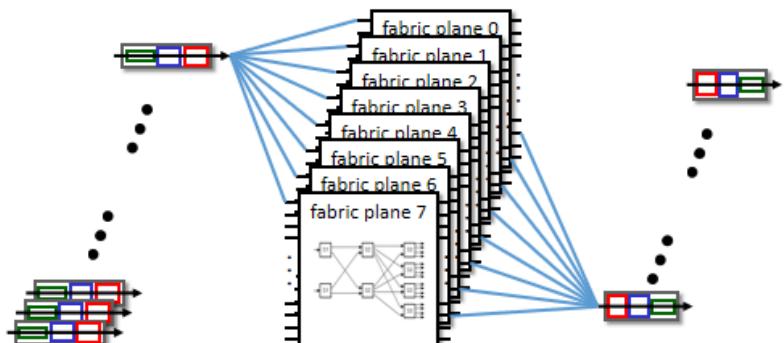


- 通过并行, 使用多个切换“平面”进行扩展:

- 加速, 通过并行扩展

- Cisco CRS router:

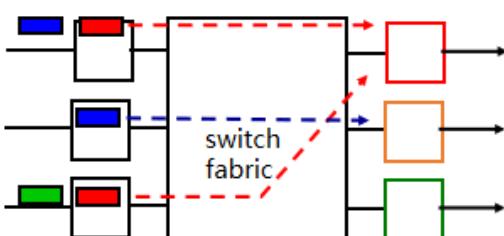
- 基本单位: 8个交换平面
- 每个平面: 3级互连网络 (interconnection network)
- 高达100 Tbps的交换容量



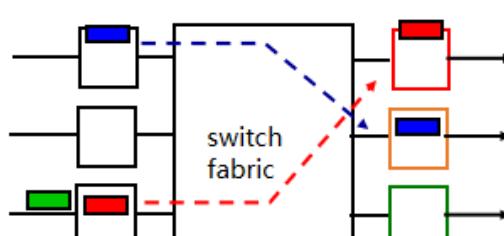
排队 (输入、输出端口)

输入

- 当交换结构的速度慢于输入端口的速度, 就会在输入端口的缓冲区发生排队
 - 会导致排队延时和由于输入缓冲区溢出导致的丢包!
- **线头阻塞 (Head-of-the-Line (HOL) blocking)** : 在队列前面的被阻塞的数据报会阻止队列中的其他数据报被转发。

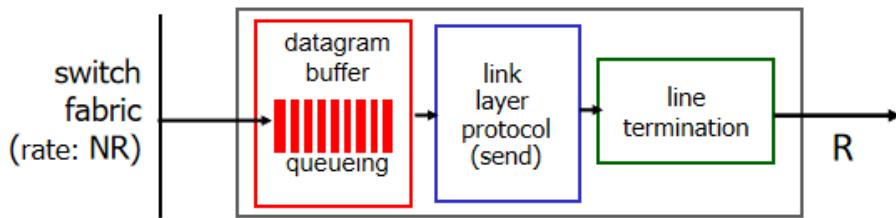


输出端口竞争:
只有一个红色的数据报可以被转移
左下角的红色分组被阻塞



一个分组时间之后: 绿色分组经历了HOL阻塞

输出端口排队



- 取出存放在输出端口内存中的分组，并将其传输到输出链路上。
- 当交换结构将分组交付给输出端口的速率超过输出链路速率，就需要排队与缓存管理功能。
- 当输出端口的缓冲区溢出时，就会出现丢包。

分组通过交换结构被送达输出端口缓冲区 (datagram buffer) 后，要等物理链路空闲才能发出。

如果输出端口来不及处理 (R速率 < 到达速率)，就必须排队。

若缓冲区被占满，就发生丢包 (丢弃分组)。

为什么缓冲区重要？

- 如果传出链路太慢，无法及时发走分组，就会在输出端口缓冲区排队等待。
- 如果等待空间不够，就要丢弃分组 (drop packet)。
- 当经过交换结构到达的速度超过了输出端口的处理线速就会发生排队
- 当输出端口的缓冲区溢出时就会发生丢包

✓ 第一部分：RFC 3439 的经验法则（经典做法）

缓冲区大小 = 典型 RTT × 链路容量

- **RTT (Round Trip Time)** : 一个分组往返一次网络的平均时延，典型值约 250ms。
- **C (链路容量)** : 比如 10Gbps。

1 2 3 4 计算示例：

java

复制 编辑

```
缓冲 = RTT × C  
      = 0.25秒 × 10Gbps  
      = 2.5 Gbit 缓冲区
```

这是最传统的 TCP 拥塞窗口设计参考，用来“撑住整个链路的吞吐”。

第二部分：新建议（针对多TCP流情况）

缓冲大小 = $(RTT \times C) / \sqrt{N}$

- 如果网络中有 N 个独立 TCP 流并发存在，
 - 那么每个流的波动可以相互抵消，
 - 就不需要为每个流都准备满窗口的缓冲。

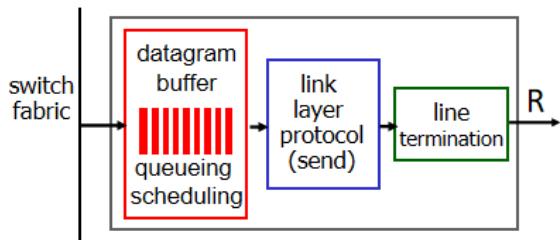
所以：

- 随着流数量 N 增加，总体缓冲需求增长缓慢（按 \sqrt{N} ）。
- 实际部署中，采用这个公式能节省很多内存成本。

第三部分：缓冲太多反而不好

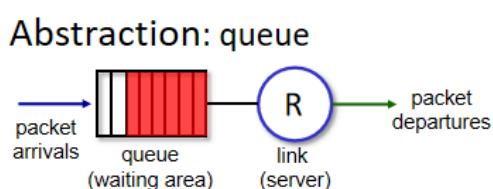
！ 多缓冲 → 高延迟 → 用户体验差

- RTT 变长：**TCP 等协议的控制响应更慢，如慢启动、丢包重传等都变得迟钝。
- 尤其在家庭路由器上，带宽小，但缓冲大时，会造成：
 - 视频播放卡顿
 - 网站加载慢
 - 延迟高导致网游不流畅



缓冲区管理:

- **丢弃:** 当缓冲区已满时，丢弃要添加的数据包
- **尾部丢弃:** 丢弃到达的分组
- **优先级:** 按照优先级丢弃/删除



- **标记:** 标记哪些数据包以指示拥塞 (ECN, RED)

路由器中哪些部分由硬件实现，哪些由软件实现？



硬件实现部分:

- **输入端口、交换结构、输出端口:**



软件实现部分:

- **路由处理器 (Routing Processor) :**

- 运行**路由协议 (如 OSPF、BGP)**、维护路由表、计算转发表。
- SDN 环境下，控制器软件还负责下发规则到设备的“流表”中。

控制平面 vs 数据平面

平面	功能	通常实现方式
数据平面	处理实际的数据包转发	硬件实现 (高速度)
控制平面	计算路径、路由协议、管理规则	软件实现 (灵活性强)

📌 一、基于目的地的转发：

- **传统方式：**只看目的 IP 地址，查找匹配的下一跳和端口。
- 只做简单匹配和输出端口选择。
- 快，但灵活性差。

📌 二、通用转发 (Generic Forwarding) :

- 是**软件定义网络 (SDN)**的重要特征。
- 转发时可以基于更多字段决策，比如：
 - TCP/UDP 源端口号、目标端口号
 - 源 IP、目标 IP、协议类型
 - 多字段组合匹配 ⇒ 可以实现：
 - NAT、负载均衡、防火墙、限速、服务质量管理等。

转发类型	能力
基于目的地址	只判断 IP 地址匹配 → 转发端口
通用转发	IP + 端口 + 协议都能判断 → 可做复杂行为，如限速、防火墙、NAT、封包

网际协议：IPv4、寻址、IPv6及其他

因特网中的网络层协议

包含三大模块

1 Path-selection algorithms (路径选择算法)

- **作用：**决定分组从源到目的地应该走哪条路径。
- **由谁实现？** 路由器的控制平面
- **常见协议：**
 - **OSPF** (开放式最短路径优先) — 内部网关协议
 - **BGP** (边界网关协议) — 跨自治系统选路
 - **SDN Controller**: 软件定义网络中的集中控制器

💡 这些算法运行后会生成 **转发表 (Forwarding Table)**，交给路由器使用。

2 IP Protocol (IP协议)

这是网络层最核心的协议，定义了：

功能	说明
Datagram format	IP数据报格式 (如IPv4的头部结构)
Addressing	IP地址结构 (如IPv4/IPv6)
Handling	处理方式规则 (TTL递减、分片等)

💡 所有数据包在网络层传输时，都是**被封装成IP数据报格式的**。



Forwarding Table (转发表)

- 是网络层数据平面中的关键结构。
- 每一个到达的数据包，其目的 IP 地址会被用来在转发表中查找下一跳接口。
- 由控制平面的路径选择算法事先生成。

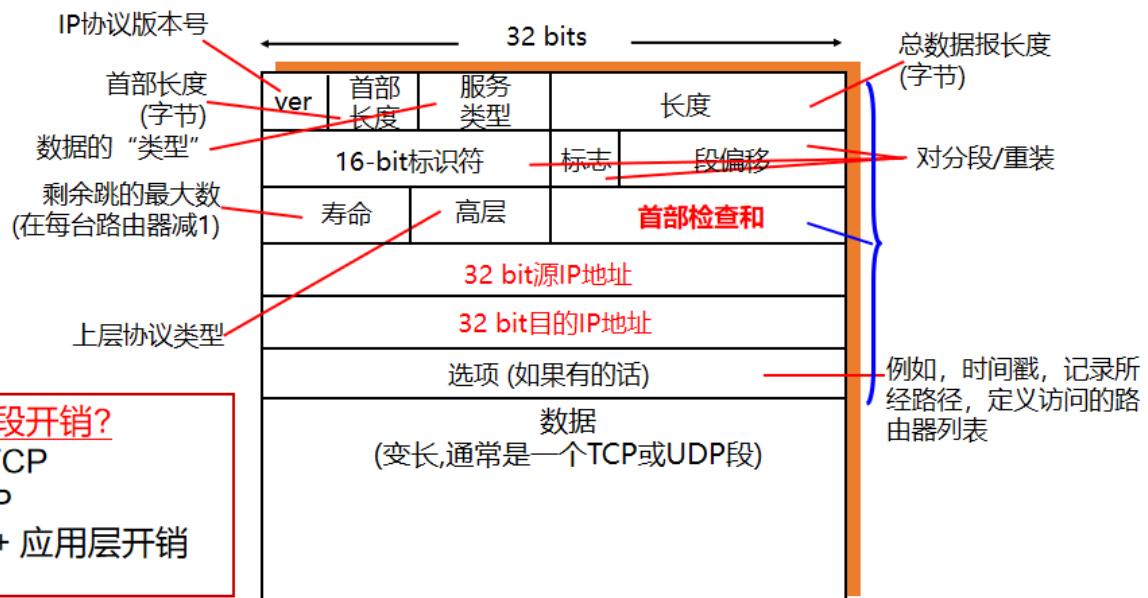
1. OSPF/BGP → 路由算法 → 生成转发表
2. 分组到达路由器，查转发表 → 找到下一跳接口
3. 根据 IP 协议格式封装和处理
4. 出错时发 ICMP 控制信息

IP 数据报格式 (IPv4)

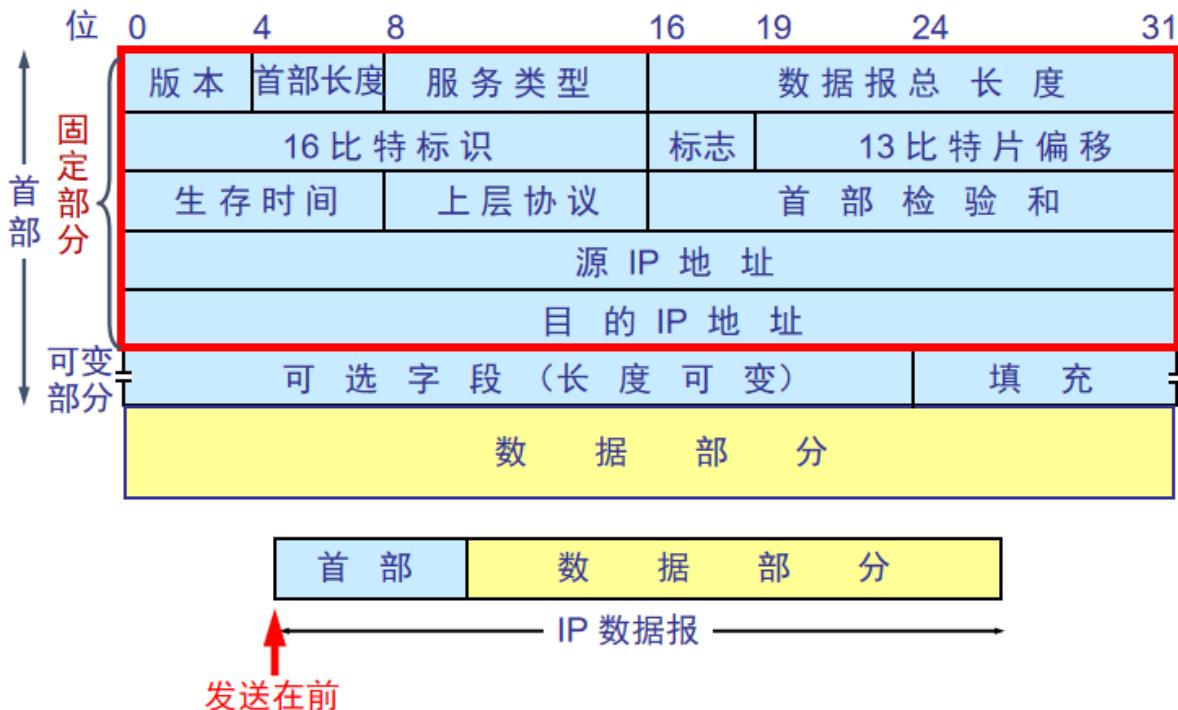
网络云

电子科技大学
University of Electronic Science and Technology of China

IP 数据报格式 (IPv4)



IP 数据报



- 版本——占 4 位，指 IP 协议的版本，目前的 IP 协议版本号为 4 (即 IPv4)。
- 首部长度——占 4 位，可表示的最大数值是 15 个单位(一个单位为 4 字节)，因此 IP 的首部长度的最大值是 60 字节。
- 服务类型——占 8 位，用来获得更好的服务。在旧标准中叫做**服务类型**，但实际上一直未被使用过。1998 年这个字段改名为**区分服务**。只有在使用区分服务 (DiffServ) 时，这个字段才起作用。在一般的情况下都不使用这个字段
- 总长度——占 16 位，指**首部和数据之和**的长度，单位为字节，因此数据报的最大长度为 65535 ($2^{16}-1$) 字节。总长度必须不超过最大传送单元 MTU。
- 标识(identification) 占 16 位，它是一个计数器，用来产生数据报的标识。
- 标志(flag) 占 3 位，目前只有两位有意义。

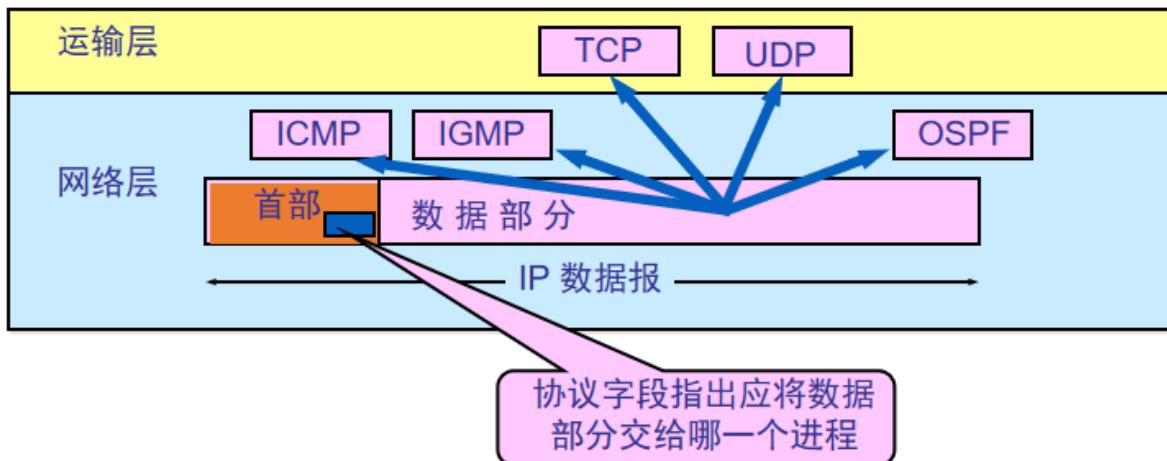
第1位：保留，一般置为0

第2位：“不分片位”，DF (Don't Fragment)。 只有当 DF =0 时才允许分片。 第3位：“更多片位”，MF (More Fragment)。 MF =1 表示后面“还有分片”。 MF =0 表示最后一个分片。

- 片偏移(13 位)指出：较长的分组在分片后，某片在原分组中的相对位置。片偏移以 8 个字节为偏移单位。
- 生存时间(8 位)记为 TTL (Time To Live)，数据报在网络中可通过的路由器数的最大值。

数据报每经过一台路由器，TTL值减1。若TTL=0，则该数据报必须丢弃。

- 上层协议(8位)字段：指出此数据报携带的数据使用何种协议，以便目的主机的 IP 层将数据部分上交给哪个处理过程



例如：协议字段的值为6，表明数据部分要交给TCP；值为17，表明数据部分要交给UDP

协议号是将网络层与运输层绑定到一起的黏合剂；
端口号是将运输层和应用层绑定到一起的黏合剂。

- 首部检验和(16位)字段只检验数据报的首部不检验数据部分。这里不采用CRC检验码而采用简单的计算方法。

为什么TCP/IP在运输层/网络层都执行差错检测？

✓ 1. IP层的差错检测 —— 检查IP首部

- IP层校验和只检查IP报文头部(Header)，而不检查其携带的数据部分。
- 原因：
 - IP头部包含路由、版本、TTL等重要信息，出错会导致包无法送达或错误转发；
 - 而数据部分属于上层(TCP/UDP)的控制范围，IP层不过多干涉。

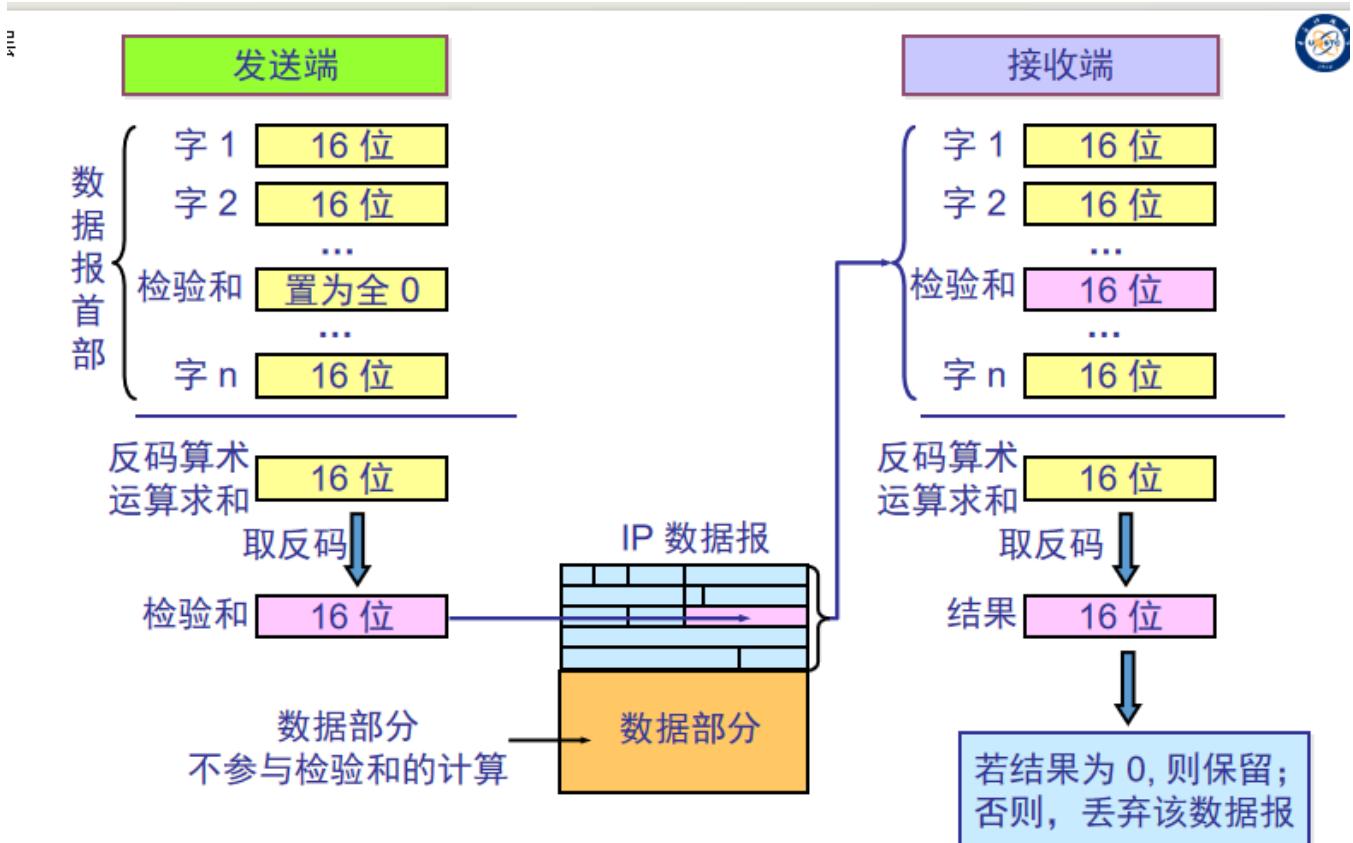
✓ 2. TCP/UDP层的差错检测 —— 检查整个报文段

- TCP/UDP的校验和覆盖的是整个传输层报文段(包括首部+数据)。
- 目的：
 - 检查数据内容是否在传输过程中被损坏；
 - TCP是可靠传输协议，必须保证数据完整性；
 - UDP是无连接的，也需要基本的数据正确性验证。

两者校验和计算的内容不同。IP层只对IP头部计算了校验和，而TCP/UDP对整个TCP/UDP报文段进行了校验和计算。

TCP/UDP和IP不一定属于同一个协议栈。TCP能够运行在其他的协议上（比如ATM, IPX, Apple Talk DDP等），IP携带的数据也不一定必须要传递给TCP/UDP（传输层还有SCTP等协议）。

校验和计算过程



📌 为什么要“反码求和”？

- 它对 1-bit 错误、短的突发错误 非常敏感；
- 更容易实现硬件运算；
- 能在网络传输中快速发现错误。

1. 计算原理上 —— 方法是一样的

三者都使用**16位反码求和** (One's Complement Sum) 来计算校验和：

- 把数据按 16 位划分为若干个字 (word)；
- 所有 word 执行“反码加法”；
- 结果再取反，得出校验和；
- 接收方再做一次加和 + 取反，看是否为全 0。

 所以：底层算法逻辑相同。

! 2. 计算内容上 —— 明显不同

协议	校验范围
IP	仅首部 (Header)
TCP/UDP	首部 + 数据部分 + 伪首部 (pseudo header)

- 源IP地址和目的IP地址都各占 4 字节

1、IP分片是在源主机还是中间路由器中进行？

答：**都有可能**。因为源主机和中间路由器可能运行在不同的链路层协议之上，使得链路层帧所能承载的最大数据量（MTU，最大传输单元）不同，比如以太网 MTU = 1500 Bytes, PPPOE MTU = 1492 Bytes, 而广域网链路的帧可承载的数据一般不超过 576 Bytes。

注意：分片后的数据重新组装只在目的端的IP层。

2、TCP和UDP报文段是否都需要进行IP分片？

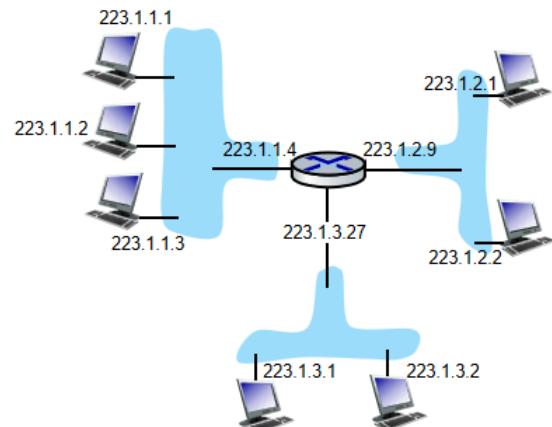
答：**IP分片只针对UDP（以及ICMP）**。对于TCP报文段，不会产生IP分片，因为一旦TCP数据过大，比如超过了MSS，传输层会对TCP报文段进行分段。

注意：IP分片只有第一个片段带有传输层或ICMP首部，其余的片段只有IP头。

IP地址

IP地址

- IP 地址: 分配给主机或路由器接口的标识符
- 接口: 主机/路由器与物理链路之间的边界
 - 路由器有多个接口
 - 主机可以有多个接口
 - 每个接口有一个IP地址
- IP地址有两种：IPV4和IPV6
 - IPV4: 32个二进制位长（4字节），常用点分十进制表示
 - IPV6: 128个二进制位长（16字节）常用冒号分隔表示



dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001
 223 1 1 1

编址方法

编址方法	出现时间	特点	优点	缺点
分类地址	1981	A/B/C类，固定前缀长度	实现简单，早期适用	浪费、僵硬
子网划分	1985	引入子网掩码	网络更灵活，地址利用率提升	仍依赖固定A/B/C类结构
CIDR 无分类编址	1993	IP + 前缀长度表示法	路由聚合、地址精细化分配	配置管理更复杂

IPv4编址

▪ 32比特的二进制表示和点分十进制表示法

- 将4个字节中的每一个字节分别用十进制数来表示，4个十进制数之间用“.”分隔。

如： $223.1.1.1 = 11011111\ 00000001\ 00000001\ 00000001$

223 1 1 1

根据不同的取值范围，早期将IP地址分为五类。IP地址中前5位用于标识IP地址的类别，A类地址的第一位为“0”，B类地址的前两位为“10”，C类地址的前三位为“110”，D类地址的前四位为“1110”，E类地址的前五位为“11110”。其中，A类、B类与C类地址为基本的IP地址。

1. 分类 IP 地址

1. 分类 IP 地址

- 每一类地址都由两个固定长度的字段组成，其中一个字段是**网络号 net-id**，它标志主机（或路由器）所连接到的网络，而另一个字段则是**主机号 host-id**，它标志该主机（或路由器）。
- 两级的 IP 地址可以记为：

IP 地址 ::= { <网络号>, <主机号> }

::= 代表“**定义为**”

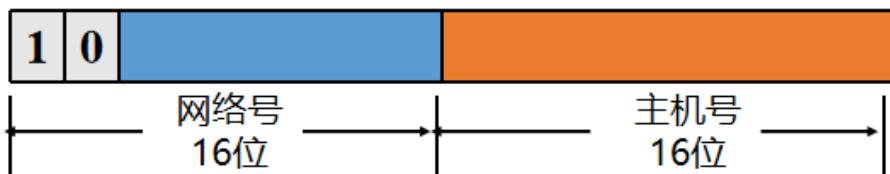


传统的IP地址分类

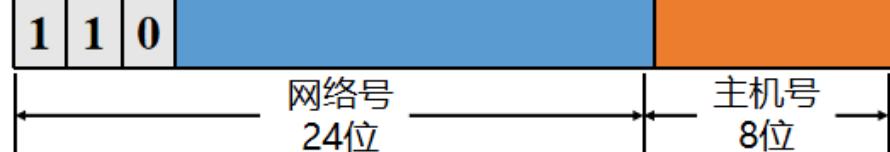
A类地址：



B类地址：



C类地址：



D类地址：



E类地址：



按照第一段（即首字节）的范围，IP 地址被划分为以下几类：

类别	首字节范围	网络号长度	主机号长度	默认子网掩码
A 类	1 – 126	8 位	24 位	255.0.0.0 (/8)
B 类	128 – 191	16 位	16 位	255.255.0.0 (/16)
C 类	192 – 223	24 位	8 位	255.255.255.0 (/24)

特殊IP地址

地址类型	地址范围	作用	能否出现在公网
回环地址	127.0.0.0/8	测试本机、localhost	✗
私有地址	10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	局域网使用	✗ (需NAT)

特殊IP地址

- **0.0.0.0**

这个地址严格上来说都不是真正意义上的IP地址。主要是用来标识不清楚的网络和主机的。系统遇到无法识别的网络或主机的时候会统一的归纳到这个地址

- **255.255.255.255**

这个地址是受限的广播地址。主要指一个网段内的所有主机

重要特点

(1) IP地址是一种分等级的地址结构。

(1) IP地址是一种分等级的地址结构。分两个等级的好处是：

- 第一，IP地址管理机构在分配IP地址时只分配网络号，而剩下的主机号则由得到该网络号的单位自行分配。这样就方便了IP地址的管理。
- 第二，路由器仅根据目的主机所连接的网络号来转发分组（而不考虑目的主机号），这样就可以使路由表中的项目数大幅度减少，从而减小了路由表所占的存储空间。

(2) 实际上IP地址是标志一个主机（或路由器）和一条链路的接口

(2) 实际上IP地址是标志一个主机（或路由器）和一条链路的接口。

- 当一个主机同时连接到两个网络上时，该主机就必须同时具有两个相应的IP地址，其网络号net-id必须是不同的。这种主机称为**多归属主机(multihomed host)**。
- 由于一个路由器至少应当连接到两个网络（这样它才能将IP数据报从一个网络转发到另一个网络），因此一个路由器至少应当有两个不同的IP地址。

(3) 用转发器或网桥连接起来的若干个局域网仍为一个网络，因此这些局域网都具有同样的网络号net-id。

(4) 所有分配到网络号net-id的网络，无论是范围很小的局域网，还是可能覆盖很大地理范围的广域网，都是平等的。

互联网中IP地址

在同一个局域网上的主机或路由器，其IP地址的网络号必须一致

路由器连接多个网络，每个接口都有一个属于对应网络的IP地址。

- 路由器有几个接口，就有几个IP地址，每个IP属于对应的网络。

● 新知识点：“不指明 IP 地址”的链路

- 如果两个路由器直接连接，这段链路可不分配 IP 地址；
- 如果要指明 IP 地址，这段链路也会被认为是一个“逻辑网络”（如图中 N1, N2, N3）；
- 现在实践中常常不再为路由器对接链路分配 IP 地址，以节省地址资源（尤其 IPv4 紧张）。

2. 子网划分

(1) 从两级 IP 地址到三级 IP 地址

- 在 ARPANET 的早期，IP 地址的设计确实不够合理。
 - IP 地址空间的利用率有时很低。
 - 给每一个物理网络分配一个网络号会使路由表变得太大因而使网络性能变坏。
 - 两级的 IP 地址不够灵活。

(2) 三级的 IP 地址

- 从 1985 年起在 IP 地址中又增加了一个“子网号字段”，使两级的 IP 地址变成三级的 IP 地址。
- 这种做法叫作划分子网(subnetting)。划分子网已成为因特网的正式标准协议。

划分子网的基本思路

- 划分子网纯属一个单位内部的事情。单位对外仍然表现为没有划分子网的网络。
- 从主机号借用若干个位作为子网号 subnet-id，而主机号 host-id 也就相应减少了若干个位。

IP 地址 ::= {<网络号>, <子网号>, <主机号>}

- 凡是从其他网络发送给本单位某个主机的 IP 数据报，仍然是根据 IP 数据报的目的网络号 net-id，先找到连接在本单位网络上的路由器。
- 然后此路由器在收到 IP 数据报后，再按目的网络号 net-id 和子网号 subnet-id 找到目的子网。
- 最后就将 IP 数据报直接交付目的主机。

子网掩码

- 当没有划分子网时，IP 地址是两级结构。
- 划分子网后 IP 地址就变成了三级结构。
- 划分子网只是把 IP 地址的主机号 host-id 这部分进行再划分，而不改变 IP 地址原来的网络号 net-id。
- 从一个 IP 数据报的首部并无法判断源主机或目的主机所连接的网络是否进行了子网划分。
- 使用子网掩码(subnet mask)可以找出 IP 地址中的子网部分。



总结归纳

概念	含义
子网划分	对主机号部分进一步划分为“子网号 + 主机号”
子网掩码	用于区分 IP 地址中 <u>网络+子网号 和 主机号的边界</u>
子网后的结构	网络号 + 子网号 + 主机号 (三级结构)
掩码作用	用于计算出 <u>网络号和子网号位置</u>

★ 子网掩码的作用：

子网掩码的本质就是用来划分 IP 地址中哪些位属于网络号 / 子网号，哪些位属于主机号。

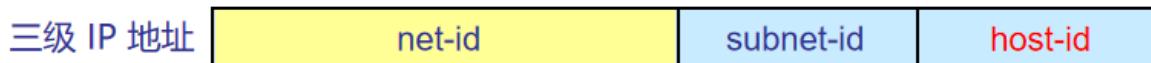
- 掩码为 **1** 的位 → 属于网络号 (包括子网号)
- 掩码为 **0** 的位 → 属于主机号

默认子网掩码

默认子网掩码

A类地址	网络地址	net-id	host-id 为全 0
	默认子网掩码 255.0.0.0	1 1 1 1 1 1 1 1 0	
B类地址	网络地址	net-id	host-id 为全 0
	默认子网掩码 255.255.0.0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
C类地址	网络地址	net-id	host-id 为全 0
	默认子网掩码 255.255.255.0	1 0 0 0 0 0 0 0 0 0	

(IP 地址) AND (子网掩码) = 网络地址



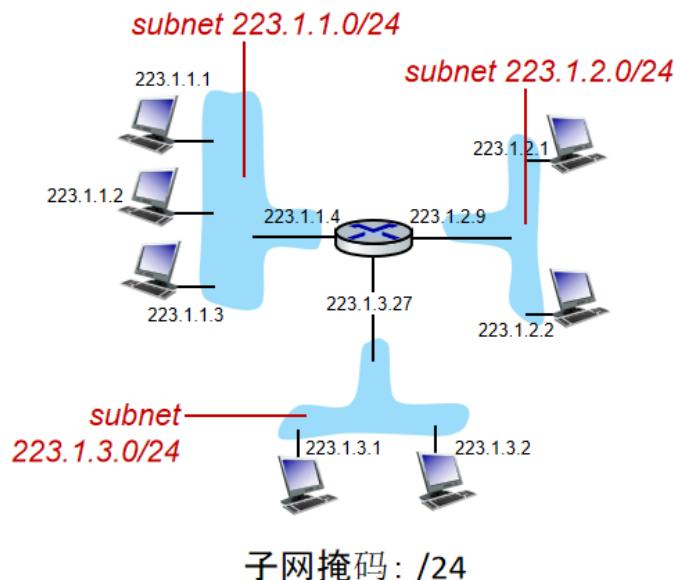
逐位进行 AND 运算



📌 什么是子网（从 IP 地址的角度看）？

子网 (subnet) 是一个具有相同网络前缀的一组设备所组成的网络。

- 什么是一个子网？
(从IP地址的观点来看)
 - 设备接口的IP地址具有同样的网络部分
 - 没有路由器的介入，物理上能够相互到达
- 定义子网的方法：
 - 将每个接口与其主机或路由器分离，创建隔离网络的“孤岛”
 - 每个隔离的网络称为一个子网



【例】已知 IP 地址是 141.14.72.24，子网掩码是 255.255.192.0。试求网络地址。

(a) 点分十进制表示的 IP 地址

141	.	14	.	72	.	24
-----	---	----	---	----	---	----

(b) IP 地址的第 3 字节是二进制

141	.	14	.01001000.	24
-----	---	----	------------	----

(c) 子网掩码是 255.255.192.0

11111111	11111111	1110000000	00000000
----------	----------	------------	----------

(d) IP 地址与子网掩码逐位相与

141	.	14	.01000000.	0
-----	---	----	------------	---

(e) 网络地址 (点分十进制表示)

141	.	14	.	64	.	0
-----	---	----	---	----	---	---

【例】在上例中，若子网掩码改为255.255.224.0。试求网络地址，讨论所得结果。

(a) 点分十进制表示的 IP 地址

141	.	14	.	72	.	24
-----	---	----	---	----	---	----

(b) IP 地址的第 3 字节是二进制

141	.	14	.	01001000	.	24
-----	---	----	---	----------	---	----

(c) 子网掩码是 255.255.224.0

11111111	11111111	11100000	00000000
----------	----------	----------	----------

(d) IP 地址与子网掩码逐位相与

141	.	14	.	01000000	.	0
-----	---	----	---	----------	---	---

(e) 网络地址 (点分十进制表示)

141	.	14	.	64	.	0
-----	---	----	---	----	---	---

不同的子网掩码得出相同的网络地址。
 但不同的掩码的效果是不同的。

子网划分

4.3.3 划分子网的IPv4地址

【举例】已知某个网络的地址为218.75.230.0，使用子网掩码255.255.255.128对其进行子网划分，请给出划分细节。

【解析】

	网络号	主机号
C类网络地址	218.75.230	0

子网掩码

255.255.255.10000000

24个连续的比特1
对应网络号部分

一个比特1表示从主机号中
借用一个比特作为子网号



划分出的子网数量 $2^I = 2$

每个子网可分配的地址数量 $2^{(8-I)} - 2 = 126$

(减2是要去掉主机号为“全0”的网络地址和“全1”的广播地址)

4.3.3 划分子网的IPv4地址

【举例】已知某个网络的地址为218.75.230.0，使用子网掩码255.255.255.128对其进行子网划分，请给出划分细节。

【解析】

网络号	主机号	
218.75.230.0	0	该网络的网络地址
218.75.230.1	1	该网络可分配的最小地址
C类网218.75.230.0 包含的全部IP地址 (共256) 个		
218.75.230.254	254	该网络可分配的最大地址
218.75.230.255	255	该网络的广播地址



4.3.3 划分子网的IPv4地址

【举例】已知某个网络的地址为218.75.230.0，使用子网掩码255.255.255.128对其进行子网划分，请给出划分细节。

【解析】

网络号	子网号	主机号	
218.75.230.0	0	0 0 0 0 0 0 0 0	子网0的网络地址 218.75.230.0 可分配最小地址 218.75.230.1
218.75.230.0	0	0 0 0 0 0 0 0 1	
C类网218.75.230.0 包含的全部IP地址 (共256) 个			
218.75.230.0	1	1 1 1 1 1 1 1 0	可分配最大地址 218.75.230.126
218.75.230.0	1	1 1 1 1 1 1 1 1	子网0的广播地址 218.75.230.127
218.75.230.1	0	0 0 0 0 0 0 0 0	子网1的网络地址 218.75.230.128
218.75.230.1	0	0 0 0 0 0 0 0 1	可分配最小地址 218.75.230.129
218.75.230.1	1	1 1 1 1 1 1 1 0	可分配最大地址 218.75.230.254
218.75.230.1	1	1 1 1 1 1 1 1 1	子网1的广播地址 218.75.230.255
接下来,			

4.3.3 划分子网的IPv4地址

【习题】已知某个网络的地址为218.75.230.0，使用子网掩码255.255.255.192对其进行子网划分，请给出划分细节。

【解析】

1. 根据所给网络地址可知其为C类网络地址，网络号占3个字节，主机号占1个字节；
2. 根据所给子网掩码可知从1字节主机号中借用2位作为子网号；

C类网218.75.230.0
包含的全部IP地址
(共256个)

划分出的子网数量 $2^2 = 4$

每个子网可分配的地址数量 $2^{(8-2)} - 2 = 62$

希望同学们都能正确完成。

网络号	子网号	主机号	
218.75.230.	0 0	0 0 0 0 0 0 0 0	子网0的网络地址: 218.75.230.0
218.75.230.	0 0	0 0 0 0 0 0 0 1	可分配最小地址: 218.75.230.1
218.75.230.	0 0	1 1 1 1 1 1 1 0	可分配最大地址: 218.75.230.62
218.75.230.	0 0	1 1 1 1 1 1 1 1	子网0的广播地址: 218.75.230.63
218.75.230.	0 1	0 0 0 0 0 0 0 0	子网1的网络地址: 218.75.230.64
218.75.230.	0 1	0 0 0 0 0 0 0 1	可分配最小地址: 218.75.230.65
218.75.230.	0 1	1 1 1 1 1 1 1 0	可分配最大地址: 218.75.230.126
218.75.230.	0 1	1 1 1 1 1 1 1 1	子网1的广播地址: 218.75.230.127
218.75.230.	1 0	0 0 0 0 0 0 0 0	子网2的网络地址: 218.75.230.128
218.75.230.	1 0	0 0 0 0 0 0 0 1	可分配最小地址: 218.75.230.129
218.75.230.	1 0	1 1 1 1 1 1 1 0	可分配最大地址: 218.75.230.190
218.75.230.	1 0	1 1 1 1 1 1 1 1	子网2的广播地址: 218.75.230.191
218.75.230.	1 1	0 0 0 0 0 0 0 0	子网3的网络地址: 218.75.230.192
218.75.230.	1 1	0 0 0 0 0 0 0 1	可分配最小地址: 218.75.230.193
218.75.230.	1 1	1 1 1 1 1 1 1 0	可分配最大地址: 218.75.230.254
218.75.230.	1 1	1 1 1 1 1 1 1 1	子网3的广播地址: 218.75.230.255

4.3.3 划分子网的IPv4地址

【2012年 题39】某主机的IP地址为180.80.77.55，子网掩码为255.255.252.0，如该主机向其所在子网发送广播分组，则目的地址可以是 D

- A. 180.80.76.0 B. 180.80.76.255 C. 180.80.77.255 D. 180.80.79.255

【解析】

B类网络地址 网络号 子网号 主机号
180.80.01001101.00110111

主机所在子网的网络地址180.80.76.0

180.80.01001100.00000000

主机所在子网的广播地址180.80.79.255

180.80.01001111.11111111

例子：子网划分

例：现有一个C类网 202.114.1.1 - 202.114.1.254，

- (1) 请写出怎样将这个C类网划分为2个、6个、14个子网；
- (2) 假设这些IP用于某公司。现公司任一部门，最多有30台机器，问应该怎样划分子网？如果有49台机器又将怎样划分？
对(1)、(2)，请写出子网所需要的位数、子网掩码和子网号。

解答要点：

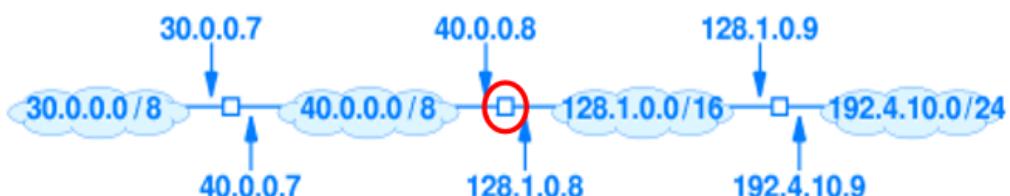
- (1) 2个子网只需借 1 bit (基于CIDR RFC1878规定)，6个子网需借 3 bit，14个子网需借 4 bit；
- (2) $30 < 32$ ，故每个子网的主机号只需5位，因为 $2^5 = 32$ ；同理49台机器的话，就需要6位，因为 $2^6 = 64$ 。

注意：对某个子网来说，主机号全0的地址不能用，它被用做表示该子网的子网号；主机号全1的也不能用，它用于本子网的广播。因此每个子网所能容纳的主机数是 $2^N - 2$ ，N是主机号位数。

基于子网的路由

使用子网掩码的分组转发

- 不划分子网时，路由表只有两项：目的网络地址和下一跳地址，例如：

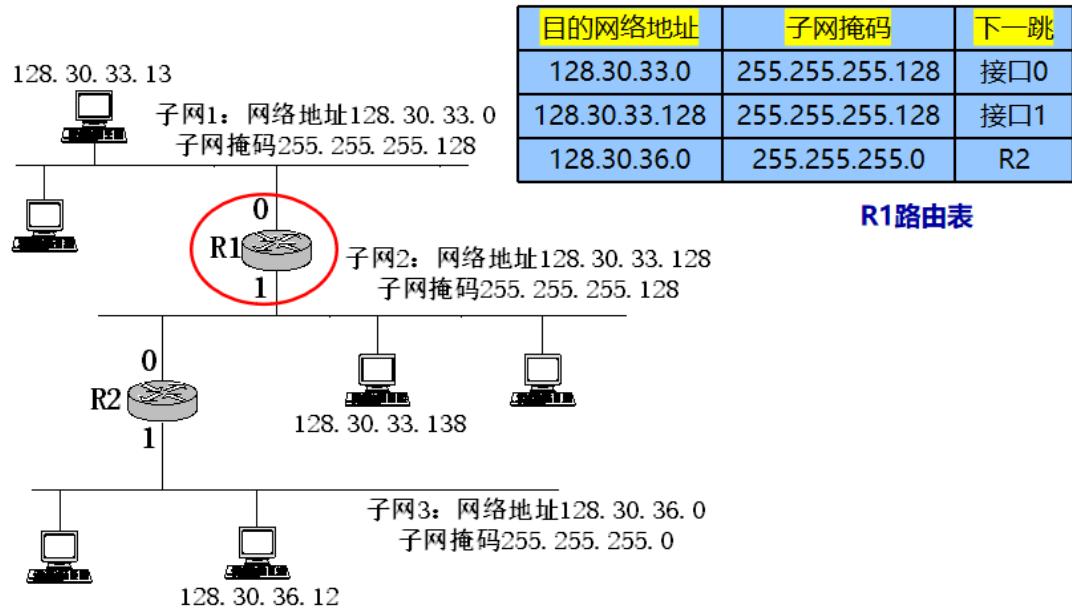


中间路由器的路由表

目的网络地址	下一跳地址
30.0.0.0	40.0.0.7
40.0.0.0	直接交付
128.1.0.0	直接交付
192.4.10.0	128.1.0.9

使用子网掩码的分组转发

使用子网划分后，路由表中将包括三项：
目的网络地址、子网掩码和下一跳地址，例如：



✓ 总结对比表

项目	不使用子网掩码	使用子网掩码 (CIDR)
路由表结构	目的网络号 + 下一跳	目的网络号 + 子网掩码 + 下一跳
匹配粒度	按 A/B/C 类网络号匹配	精确到任意子网划分
优点	简单，早期适用	<u>精准、高效利用地址空间</u>
缺点	粗略匹配，地址浪费严重	路由表更复杂，需 <u>支持最长前缀匹配</u>

3. 无分类编址 CIDR

主要特点

CIDR 最主要的特点

- CIDR 消除了传统的 A 类、B 类和 C 类地址以及划分子网的概念，因而可以更加有效地分配 IPv4 的地址空间。
- CIDR 使用各种长度的 “**网络前缀**” (network-prefix) 来代替分类地址中的网络号和子网号。
- IP 地址从三级编址 (使用子网掩码) 又回到了两级编址。

4.3.4 无分类编址的IPv4地址

■ CIDR使用“斜线记法”，或称CIDR记法。即在IPv4地址后面加上斜线“/”，在斜线后面写上网络前缀所占的比特数量。

【举例】

128.14.35.7 / 20

网络前缀占用的比特数量：20

主机编号占用的比特数量： $32-20=12$

■ CIDR实际上是将网络前缀都相同的连续的IP地址组成一个“CIDR地址块”。

■ 我们只要知道CIDR地址块中的任何一个地址，就可以知道该地址块的全部细节：

- 地址块的最小地址
- 地址块的最大地址
- 地址块中的地址数量
- 地址块聚合某类网络（A类、B类或C类）的数量
- 地址掩码（也可继续称为子网掩码）



4.3.4 无分类编址的IPv4地址

【例1】请给出CIDR地址块128.14.35.7/20的全部细节（最小地址，最大地址，地址数量，聚合C类网数量，地址掩码）。

【解析】

128.14.35.7/20 → **128.14.00100011.00000111**

最小地址：**128.14.32.0**

128.14.00100000.00000000

最大地址：**128.14.47.255**

128.14.00101111.11111111

地址数量： $2^{(32-20)}$

聚合C类网的数量： $2^{(32-20)} \div 2^8$

地址掩码：**255.255.240.0**

11111111.11111111.11110000.00000000



CIDR 记法的其他形式

- 10.0.0.0/10 可简写为 10/10，也就是把点分十进制中低位连续的 0 省略。
- 10.0.0.0/10 隐含地指出 IP 地址 10.0.0.0 的掩码是 255.192.0.0。此掩码可表示为

掩码中有 10 个连续的 1

- 网络前缀的后面加一个星号 * 的表示方法
如 00001010 00*，在星号 * 之前是网络前缀，而星号 * 表示 IP 地址中的主机号，可以是任意值。

对比传统类地址方式（有助于你理解）

写法	解释方式	掩码
10.0.0.0 (A类)	默认按 A 类地址处理 (/8)	255.0.0.0
10.0.0.0/10	明确指明前 10 位是网络号	255.192.0.0 (自动推导)

路由聚合/构成超网

4.3.4 无分类编址的IPv4地址

■ 路由聚合（构造超网）

【举例】



路由器R2的路由表	
目的网络	下一跳
:	:
172.1.4.0/25	R1
172.1.4.128/25	R1
172.1.5.0/24	R1
172.1.6.0/24	R1
172.1.7.0/24	R1
:	:

路由聚合
找共同前缀

共同前缀
172.1.4.0/25 → 172.1.00000100.0
172.1.4.128/25 → 172.1.00000100.128
172.1.5.0/24 → 172.1.00000101.0
172.1.6.0/24 → 172.1.00000110.0
172.1.7.0/24 → 172.1.00000111.0
共22位

聚合地址块: 172.1.4.0 / 22

■ 网络前缀越长，地址块越小，路由越具体；

■ 若路由器查表转发分组时发现有多条路由可选，则选择网络前缀最长的那条，这称为**最长前缀匹配**，因为这样的路由更具体。**接下来，我们再来做几个相关的考研题。**

4.3.4 无分类编址的IPv4地址

- 划分子网在一定程度上缓解了因特网在发展中遇到的困难，但是**数量巨大的C类网**因为其**地址空间太小**并没有得到充分使用，而因特网的IP地址仍在加速消耗，整个**IPv4地址空间面临全部耗尽的威胁**。
- 为此，因特网工程任务组IETF又提出了采用**无分类编址**的方法来解决IP地址紧张的问题，同时还专门成立IPv6工作组负责研究新版本IP以彻底解决IP地址耗尽问题。
- 1993年，IETF发布了**无分类域间路由选择CIDR**(Classless Inter-Domain Routing)的RFC文档：RFC 1517~1519和1520。
 - CIDR消除了传统的A类、B类和C类地址，以及划分子网的概念；**
 - CIDR可以更加有效地分配IPv4的地址空间，并且可以在新的IPv6使用之前允许因特网的规模继续增长。**
- CIDR使用“**斜线记法**”，或称CIDR记法。即在IPv4地址后面加上斜线“/”，在**斜线后面写上网络前缀所占的比特数量**。
- CIDR实际上是将**网络前缀都相同的连续的IP地址组成一个“CIDR地址块”**。
- 我们**只要知道CIDR地址块中的任何一个地址，就可以知道该地址块的全部细节：**

- 地址块的最小地址
 - 地址块的最大地址
 - 地址块中的地址数量
 - 地址块聚合某类网络（A类、B类或C类）的数量
 - 地址掩码（也可继续称为子网掩码）
- **路由聚合**（构造超网）的方法是**找共同前缀**
- **网络前缀越长，地址块越小，路由越具体；**
- 若路由器查表转发分组时发现有多条路由可选，则选择**网络前缀最长的那条，这称为最长前缀匹配**，因为这样的路由更具体。



- **一个CIDR地址块可以表示很多地址**，这种地址的聚合常称为**路由聚合**，它使得路由表中的一个项目可以表示很多个（例如上千个）原来传统分类地址的路由。
- 路由聚合也称为**构成超网**(supernetting)。
- CIDR 虽然不使用子网了，但仍然使用“**掩码**”这一名词（但不叫子网掩码）。

假设有16个C类网络，从 201.66.32.0 到 201.66.47.0，它们可以用掩码 255.255.240.0 统一表示为网络 201.66.32.0。

定长&变长的子网掩码

4.3.5 IPv4地址的应用规划

**定长的子网掩码FLSM
(Fixed Length Subnet Mask)**

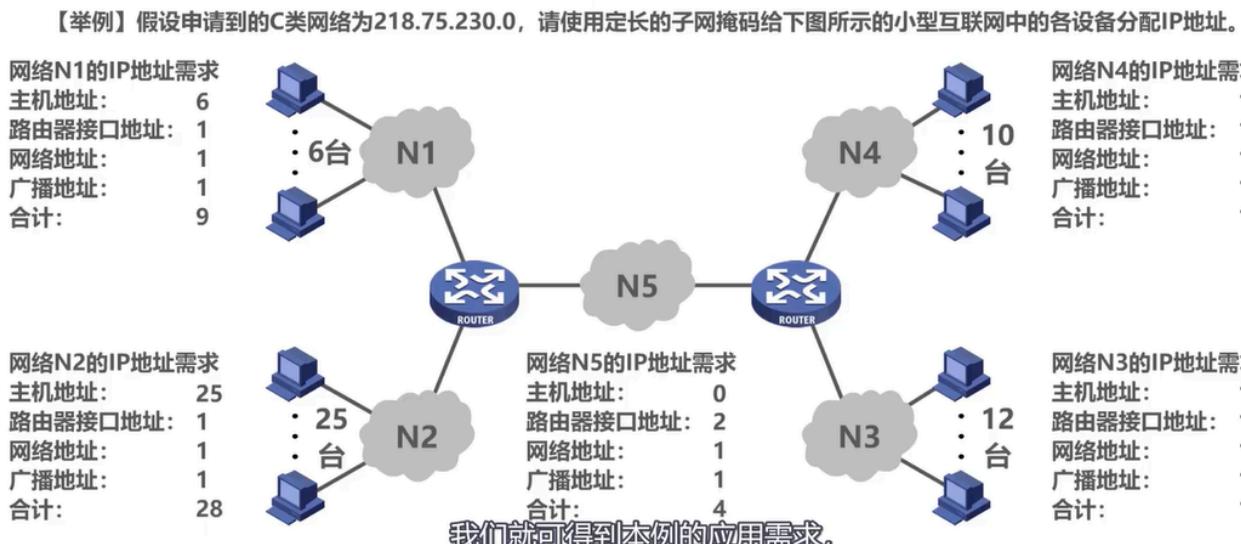
- **使用同一个子网掩码来划分子网**
- **每个子网所分配的IP地址数量相同，造成IP地址的浪费**

**变长的子网掩码VLSM
(Variable Length Subnet Mask)**

- **使用不同的子网掩码来划分子网**
- **每个子网所分配的IP地址数量可以不同，尽可能减少对IP地址的浪费**

FLSM

定长的子网掩码FLSM



2^n

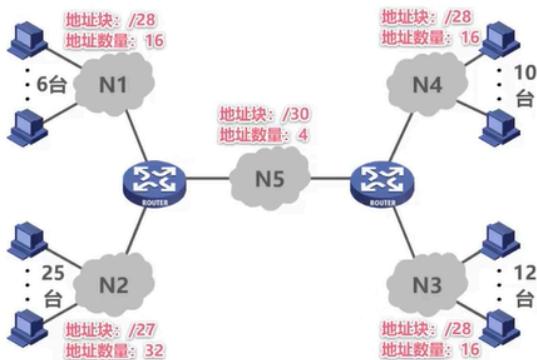
VLSM

4.3.5 IPv4地址的应用规划

定长的子网掩码FLSM

【举例】假设申请到的地址块为218.75.230.0/24，请使用变长的子网掩码给下图所示的小型互联网中的各设备分配IP地址。

应用需求：从地址块218.75.230.0/24中取出5个地址块（1个“/27”地址块，3个“/28”地址块，1个“/30”地址块），按需分配给下图所示的5个网络。



变长的子网掩码VLSM

218.75.230.0/24地址块所包含的全部地址如下所示：

- 218.75.230.0
- 218.75.230.1
- 218.75.230.2
- 218.75.230.3
- 218.75.230.4
- 218.75.230.5
- 218.75.230.6
- ⋮
- 218.75.230.249
- 218.75.230.250
- 218.75.230.251
- 218.75.230.252
- 218.75.230.253
- 218.75.230.254

在该地址块中给左图所示的网络N1~N5分配子块，分配原则是“每个子块的起点位置不能随意选取，只能选取块大小整数倍的地址作为起点”。建议先给大的子块分配。

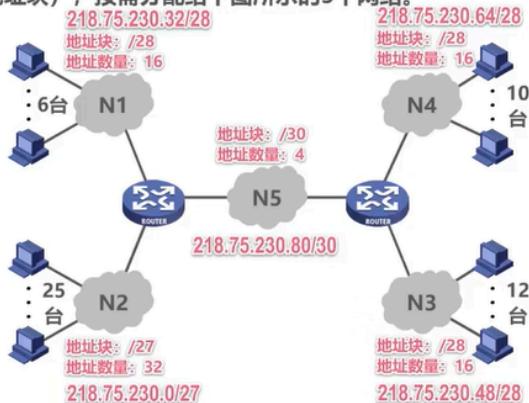
将这32个地址作为一个地址块分配给网络N2。

4.3.5 IPv4地址的应用规划

定长的子网掩码FLSM

【举例】假设申请到的地址块为218.75.230.0/24，请使用变长的子网掩码给下图所示的小型互联网中的各设备分配IP地址。

应用需求：从地址块218.75.230.0/24中取出5个地址块
(1个“/27”地址块，3个“/28”地址块，1个“/30”
地址块)，按需分配给下图所示的5个网络。



变长的子网掩码VLSM

218.75.230.0/24地址块所包含的全部地址如下所示：

218.75.230.0	网络N2的网络地址
218.75.230.1	网络N2的广播地址
218.75.230.31	网络N1的网络地址
218.75.230.32	网络N1可分配地址
218.75.230.47	网络N1的广播地址
218.75.230.48	网络N3的网络地址
218.75.230.63	网络N3可分配地址
218.75.230.64	网络N3的广播地址
218.75.230.79	网络N4的网络地址
218.75.230.80	网络N4可分配地址
218.75.230.79	网络N4的广播地址
218.75.230.80	网络N5的网络地址
218.75.230.83	网络N5可分配地址
218.75.230.84	网络N5的广播地址
218.75.230.255	剩余待分配

子网划分及CIDR练习

现有一公司已获得网络号为**202.1.1.0/24**，如果该公司有3个部门：

- (1) 如果第1个部门有60台计算机，第2个部门有20台计算机，第3个部门有16台计算机，问使用**分类IP**的子网划分方法如何分配地址？
- (2) 如果第1个部门有120台计算机，第2个部门有60台计算机，第3个部门有60台计算机，使用**分类IP**的子网划分方法可以分配地址吗？使用**CIDR方法**如何分配地址？



IP地址: 怎样获取?

问: 主机如何得到IP地址?

- 手工指定 (保存在系统配置中)

- Windows: 控制面板->网络
- UNIX/LINUX: 在/etc/rc.config中, 可使用ifconfig命令配置

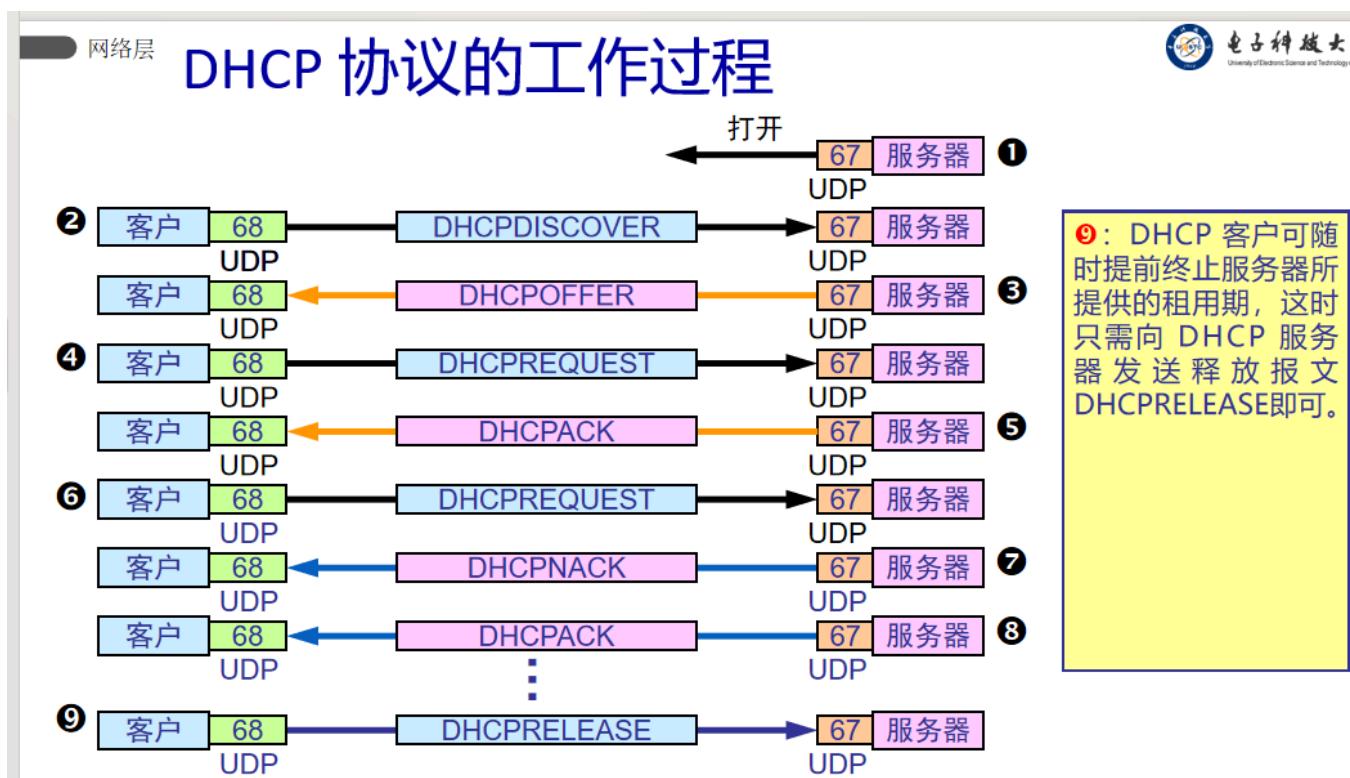
- DHCP: Dynamic Host Configuration Protocol

- 自动从一个DHCP服务器得到IP地址
- 方便灵活

DHCP协议 (动态主机配置协议)

DHCP (Dynamic Host Configuration Protocol) 是一种**应用层协议**, 用于**动态地为主机分配 IP 地址及相关网络配置**, 实现即插即用 (plug-and-play)。

工作过程





二、DHCP 工作流程（四次握手） ——配合图 1 和 图 2

👉 四步过程：

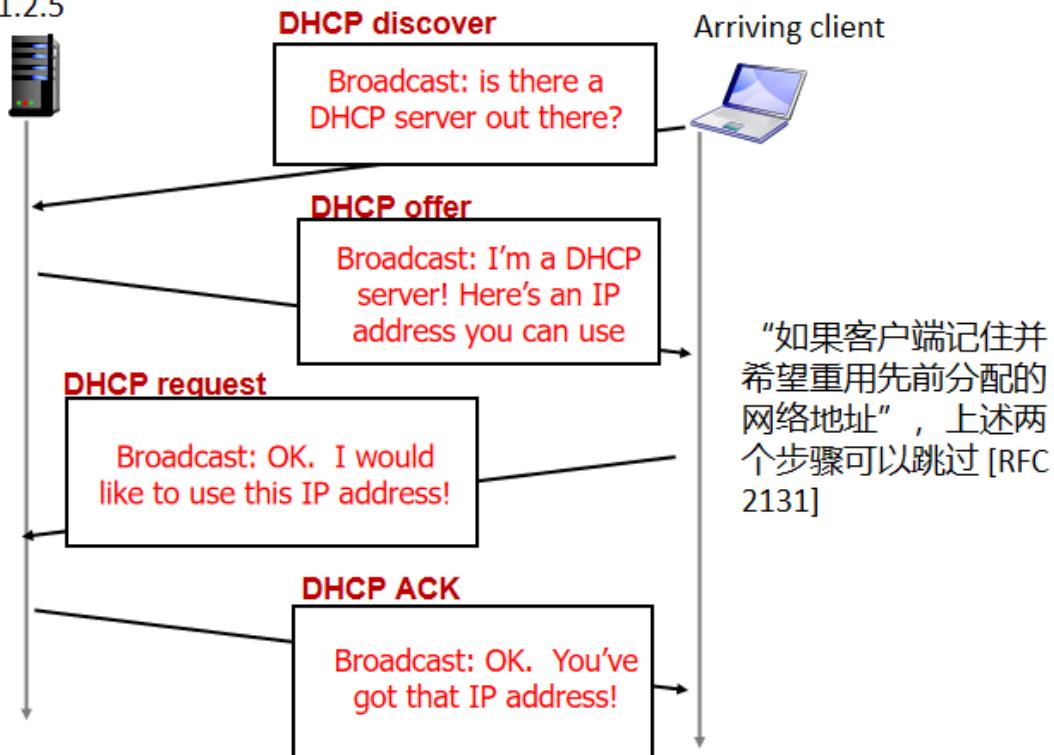
步骤	名称	含义
①	DHCP Discover	客户端广播询问有没有 DHCP 服务器
②	DHCP Offer	服务器响应，提供一个可用 IP 地址等信息
③	DHCP Request	客户端向指定服务器申请这个 IP
④	DHCP ACK	服务器确认租用，配置成功

| 所有交互基于 UDP：

- 客户端使用端口 68
- 服务器使用端口 67

DHCP 客户端-服务器场景

DHCP server: 223.1.2.5



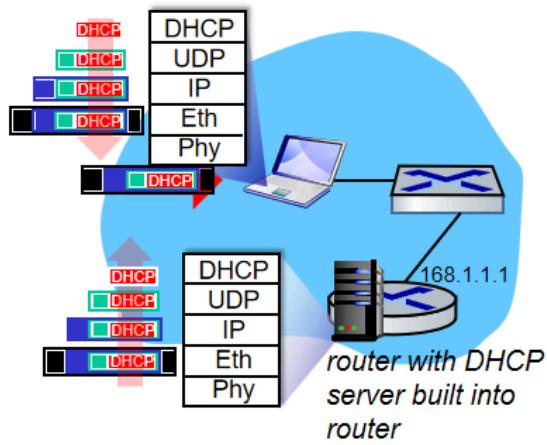
每个广播步骤解释：

1. **Discover**: 客户端刚接入网络，还没有 IP，用广播发出“我是谁”请求。
2. **Offer**: 服务器广播回应“我有地址给你：如 223.1.2.100”
3. **Request**: 客户端广播请求这个地址
4. **ACK**: 服务器确认并分配租期等信息

DHCP: 不仅获得IP地址

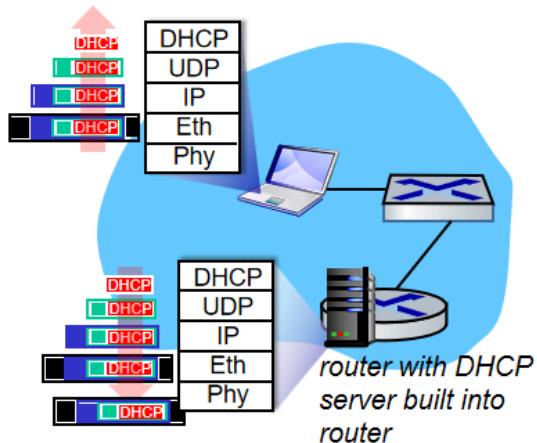
- DHCP分配的不仅仅是IP地址，还可分配：
 - 客户的第一跳路由器的地址（网关）
 - DNS服务器的IP地址或域名
 - 子网掩码
- DHCP是应用层协议

DHCP: example



- 连接笔记本将使用IP地址、第一跳路由器的地址、DNS服务器的地址：使用DHCP
- DHCP REQUEST 封装在 UDP 报文段、IP 数据包 以及 802.1 以太网帧中
- 以太网帧在 LAN 上广播（目的：FFFFFFFFFF）on LAN，运行 DHCP 服务器的路由器接收到该帧
- 以太网解析出 IP 地址，UDP 解析出 DHCP

DHCP: example



- DHCP 服务器形成包含客户 IP、客户机第一跳路由器的 IP、DNS 服务器的名称和 IP 的 DHCP ACK
- 封装的DHCP服务器帧转发到客户，在客户端解析出 DHCP
- 客户现在就知道了其IP地址、DNS服务器名称和IP地址，第一跳路由器的IP地址

- DHCP 消息是嵌套在 UDP → IP → 以太网帧中的
- 没有 IP 地址时，广播地址是 255.255.255.255 (MAC为FF:FF:FF:FF:FF:FF)
- 如果 DHCP Server 不在本子网，**路由器中继服务**负责转发请求

在前面的图 2 右上角 (步骤9) 提到：

- 客户端使用完 IP 后，可以发送 **DHCP RELEASE** 消息给服务器，释放地址，供他人使用。
- DHCP 服务器收到后，会将该地址标记为“未使用”。

客户端无 IP ——发送广播 DHCP DISCOVER



服务器回应可用地址 ——DHCP OFFER



客户端请求使用某 IP ——DHCP REQUEST



服务器确认并分配 ——DHCP ACK



(以后可能发送 RELEASE 释放地址)

组织机构如何获取IP地址？

Q: 怎样获取IP地址中的网络号部分？

A: 从ISP的地址空间中划分一块给申请者

例如：

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

ISP 拥有一个 /20 地址块后，可以通过改变子网掩码长度（CIDR）*将其划分为多个 /23 地址块，每个分配给一个组织，从而*合理分配地址，节约空间，避免浪费。

▪ ISP获得地址块的方法——从ICANN获取

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

- 分配IP地址
- 管理DNS
- 分配域名，解决纠纷

NAT: 网络地址转换

NAT 的动机和基本原理

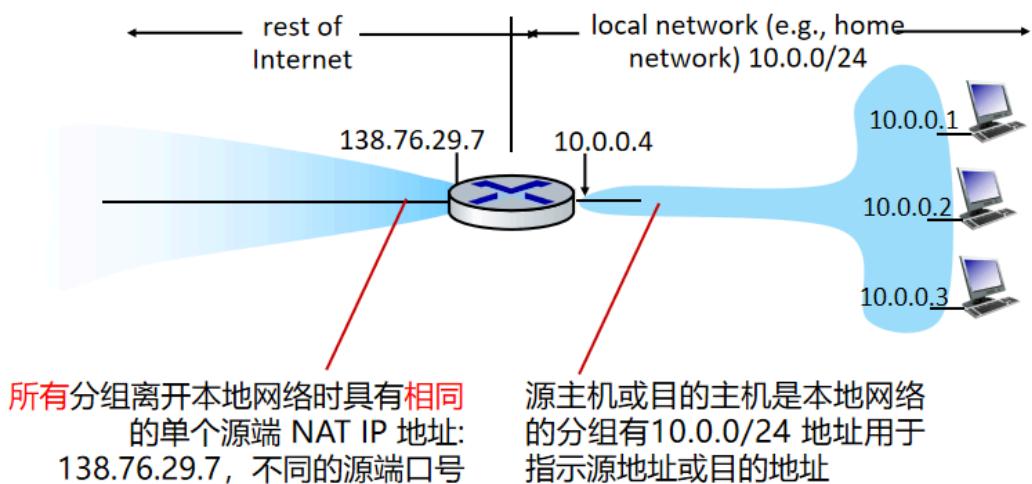
📌 动机：

- **公网 IPv4 地址紧张**: 一个本地网络只需要一个公网 IP 地址（由 ISP 提供），就能供整个局域网内多个设备访问外部网络。
- **内部地址复用**: 所有内部设备可以使用私有地址空间（例如 10.0.0.0/8、192.168.0.0/16 等）。
- **提高安全性**: 外部网络无法直接访问本地设备，提高了网络隔离和安全性。

▪ 动机: 对外部网络来讲，本地网络只用一个IP地址

- 不需要从 ISP 分配一系列地址——只要一个IP地址用于所有设备
- 在本地网络，改变设备的IP地址不用通知外部世界
- 可以变更 ISP，不用改变本地网络的设备的地址
- 本地网络内部设备不能被外部世界明确寻址，或是不可见（增加了安全性）

NAT: 就外部世界而言，本地网络中的所有设备仅共享一个IPv4地址



- 本地网络中的所有设备在“专用”IP地址空间(10 / 8、172.16 / 12、192.168 / 16)中具有32位地址，这些地址只能在本地网络中使用
- 好处：
 - 供应商ISP只需为所有设备提供一个IP地址
 - 可以在不通知外界的情况下更改本地网络中主机的地址
 - 可以更改ISP而无需更改本地网络中设备的地址
 - 安全性：本地网络中的设备不可直接寻址，外界可以看到

实现

- 实现：NAT路由器必须(透明地)：
 - 外出的分组：替换每个外出的分组的(源IP地址，端口号)为(NAT IP地址，新端口号)
 - 远程客户/服务器用(NAT IP地址，新端口号)作为目的地来响应。
 - (在NAT转换表中)记录每个(源IP地址，端口号)到(NAT IP地址，新端口号)转换配对
 - 进来的分组：对每个进来的分组，用保存在NAT表中的对应的(源IP地址，端口号)替换分组中的目的域(NAT IP地址，新端口号)

核心思想：

内部设备发送数据时，NAT 路由器会将源地址转换为自己的公网 IP 地址，并记录原地址和端口号。

- 回包时，根据转换表（NAT 表）找到原内部地址并转发。

口 NAT 转换过程四步（见第五张图）：

- 主机 `10.0.0.1:3345` 向公网主机 `128.119.40.186:80` 发送请求；
- NAT 路由器将源地址改为 `138.76.29.7:5001`，并记录转换表；
- 响应发回 NAT 路由器的 `138.76.29.7:5001`；
- 路由器查表还原目标地址为 `10.0.0.1:3345`，转发给原主机。

三、NAT 的分类和表项管理

表项结构（图 4）：

- 每个连接都会在 NAT 表中记录一条映射：

SCSS

(LAN IP, LAN 端口) <-> (公网 NAT IP, NAT 端口)

两类转换：

- 出站分组：**内部 → 外部，修改源地址；
- 入站分组：**外部 → 内部，修改目标地址。

优点

四、NAT 的优点（图 1 & 3）

- 地址节省：**多个设备共享一个公网地址；
- 网络独立性：**更换 ISP 或内部地址无需相互影响；
- 安全性：**外部无法直接访问局域网设备（除非端口映射）；
- 提高灵活性：**只需对 NAT 路由器配置，内部网络可自由规划。

🚫 限制：

- 端口数量有限：16-bit 端口最多支持约 6 万个并发连接；
- 违反端到端原则：
 - 路由器需处理第 4 层以上内容（原本只处理第 3 层 IP）；
 - 对于某些 P2P 应用（如 Skype、BT）可能影响功能；
 - 应用程序需要感知 NAT 存在，增加复杂性；
- IPv6 是更彻底的解决方案：拥有几乎无限的地址空间，不再需要 NAT。

ICMP 因特网控制报文协议

- **ICMP: Internet Control Message Protocol**
- 用于主机路由器之间彼此交流网络层信息
 - 差错报告：不可到达的主机，网络、端口、协议
 - 请求/应答（用于 ping, traceroute）
- 位于IP之上
 - 因为ICMP消息是装载在IP分组里的

<u>类型</u>	<u>代码</u>	<u>描述</u>
0	0	ping应答 (ping)
3	0	目的网络不可到达
3	1	目的主机不可到达
3	2	目的协议不可到达
3	3	目的端口不可达到
3	6	不知道的目的网络
3	7	不知道的目的主机
4	0	源端抑制 (拥塞控制 – 不用)
8	0	ping请求 (ping)
9	0	路由器公告
10	0	路由器发现
11	0	TTL 过期
12	0	IP首部损坏

一、Traceroute 的基本原理

Traceroute (或 Windows 中的 tracert 命令) 用来**发现从源主机到目标主机路径中的所有路由器。**

Traceroute 的工作依赖两个关键机制：

- **TTL (Time To Live) 机制：**控制报文最多能经过多少跳 (即多少个路由器) 。
- **ICMP 报文反馈机制：**当 TTL 用尽时，路由器返回 ICMP 报文告知源主机。

Traceroute 和 ICMP

工作流程详解

1. 发送 UDP 报文，逐步增加 TTL 值：

- 发送第一个 UDP 报文，`TTL=1`；
- 第二个 `TTL=2`；
- 依此类推.....

2. 当 TTL 用尽时，路由器的行为：

- 路由器会丢弃该数据报；
- 返回一个 ICMP "超时" 报文 (**type 11, code 0**)；
- 报文中包含了该路由器的 IP 地址（和主机名）；

3. 源主机接收到 ICMP 报文：

- 获取该路由器 IP；
- 计算从发送到收到 ICMP 的 RTT (Round-Trip Time)；
- 每个 TTL 值会重复发送三次，统计平均 RTT；

4. 何时停止发送？

停止发送的依据：

- UDP 报文最终到达目的端
- 目的端返回回应应答的 ICMP 报文 (**type 3, code 3**)
- 源端停止发送

5.
 - 当数据报最终抵达目标主机；
 - 目标主机收到无法识别的 UDP 报文（通常使用较高的端口号，目标未监听）；
 - 于是返回 ICMP 报文 "端口不可达" (**type 3, code 3**)；
 - Traceroute 根据这个 ICMP 报文判断“到达目的地”，停止追踪。

ICMP 类型	Code	含义
type 11	0	Time Exceeded (TTL 用尽)
type 3	3	Destination Unreachable (端口不可达)

- 源端发送一系列的 UDP 分组给目的端

- 第一个分组 TTL = 1
- 第二个分组 TTL = 2, 等等

- 当第n个分组到达第n个路由器时

- 路由器丢弃该分组
- 并给源端发送一个 ICMP 报文
(type 11, code 0)
- 这个报文包含了路由器的名称和 IP 地址

- 当源端收到 ICMP 报文时，计算传输往返时间 RTT
- 对每个 TTL 作三次

停止发送的依据：

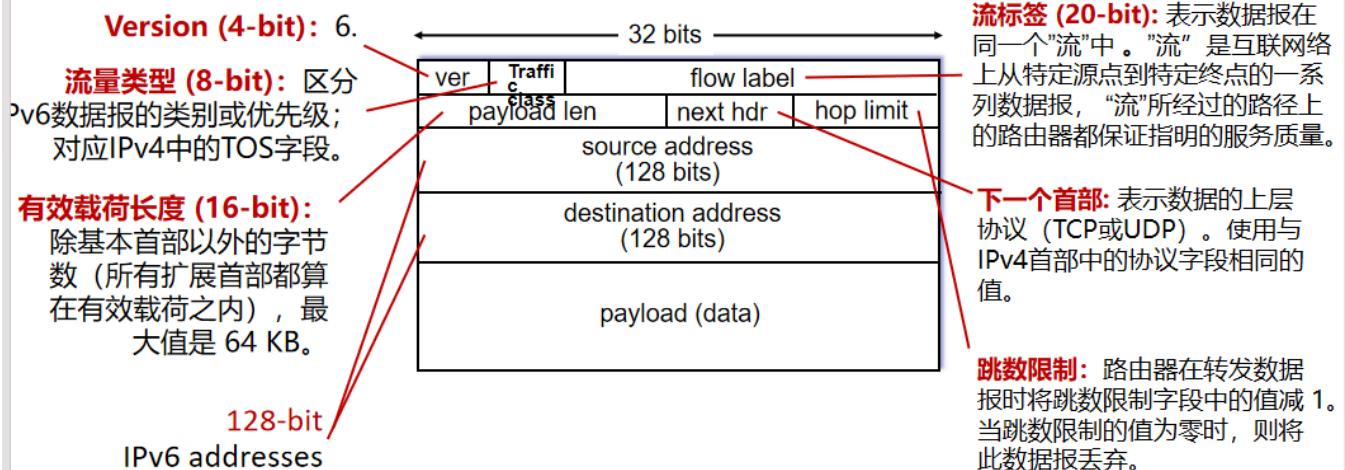
- UDP 报文最终到达目的端
- 目的端返回回应应答的 ICMP 报文
(type 3, code 3)
- 源端停止发送

IPv6

- 其他动机：

- 更大的地址空间。IPv4 32 位 -> IPv6 128 位。
- 扩展的地址层次结构。
- 灵活的首部格式。IPv6 定义了许多可选的扩展首部。
- 改进的选项。IPv6 允许数据报包含有选项的控制信息，其选项放在有效载荷中。
- 允许协议继续扩充。
- 支持即插即用（即自动配置）。因此 IPv6 不需要使用 DHCP。
- 支持资源的预分配。IPv6 支持实时视像等要求，保证一定的带宽和时延的应用。
- IPv6 首部改为 8 字节对齐。首部长度必须是 8 字节的整数倍。原来的 IPv4 首部是 4 字节对齐。

IPv6首部



与IPv4比较

▪ 与 IP v4比较，IP v6：

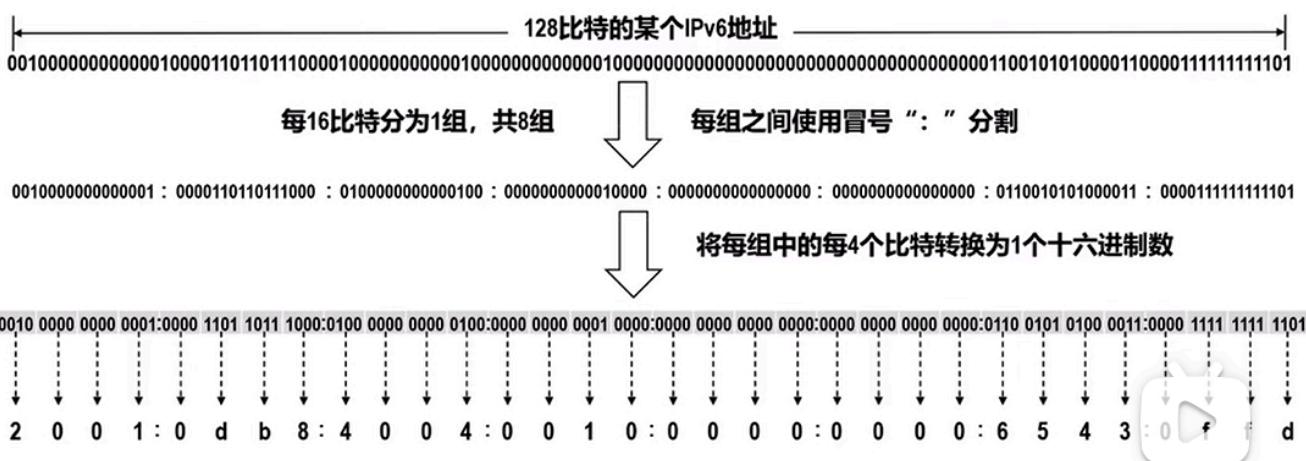
- **无首部长度:** IPv6 首部长度固定为 40 字节
- **无校验和:** 全部去掉，减少每一跳的处理时间
- **无选项** (使得首部固定为40字节) : 使用扩展首部来实现选项功能。
- 服务类型字段 TOS -> **流量类型 Traffic class**
- 总长度 -> **有效载荷长度**
- 协议字段 -> **下一个首部**
- TTL 字段 -> **跳数限制**
- **无分片/组装**

扩展首部

- IPv6 把原来 IPv4 首部中选项的功能都放在**扩展首部**中，并将扩展首部留给路径两端的源站和目的站的主机来处理。
 - 数据报途中经过的路由器都不处理这些扩展首部（只有一个首部例外，即**逐跳选项**扩展首部）。
 - 这样就大大提高了路由器的处理效率。
 - 在 RFC 2460 中定义了六种扩展首部：**逐跳选项**、路由选择、分片、鉴别、封装安全有效载荷、目的站选项

地址表示

02 | IPv6地址的表示方法



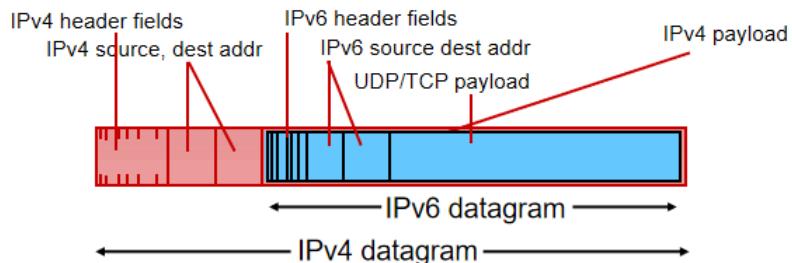
- 为了使地址再稍简洁些，IPv6 使用**冒号十六进制记法**(colon hexadecimal notation, 简写为 colon hex)。
 - 每个 16 位的值用十六进制值表示，各值之间用冒号分隔。例如：
68E6:8C64:FFFF:FFFF:0:1180:960A:FFFF
 - 在十六进制记法中，允许把数字前面的 0 省略。例如把 0000 中的前三个 0 省略，写成 1 个 0。
 - 冒号十六进制记法可以允许**零压缩** (zero compression)，即一连串连续的零可以为一对冒号所取代。
FF05:0:0:0:0:0:B3 可压缩为： **FF05::B3**
 - 注意：在任一地址中只能使用一次零压缩。

- 冒号十六进制记法可结合使用点分十进制记法的后缀，这种结合在 IPv4 向 IPv6 的转换阶段特别有用。
- 例如： **0:0:0:0:0:128.10.2.1**
再使用零压缩即可得出： **::128.10.2.1**
- CIDR 的斜线表示法仍然可用。
- 例如：60 位的前缀 12AB00000000CD3 可记为：
12AB:0000:0000:CD30:0000:0000:0000:0000/60
或 **12AB::CD30:0:0:0/60** (零压缩)
或 **12AB:0:0:CD30::/60** (零压缩)

从IPv4 过渡到 IPv6

隧道

- 并非所有路由器都可以同时升级
 - 没有“标志日”
 - 网络将如何与IPv4和IPv6混合路由器一起运行？
- 隧道：**在IPv4路由器之间作为IPv4数据报中的有效载荷携带IPv6数据报（“数据包中的数据包”）
 - 在其他情况下广泛使用的隧道（4G / 5G）



隧道：在IPv4路由器之间作为IPv4数据报中的有效载荷携带IPv6数据报（“数据包中的数据包”）在其他情况下广泛使用的隧道（4G / 5G）

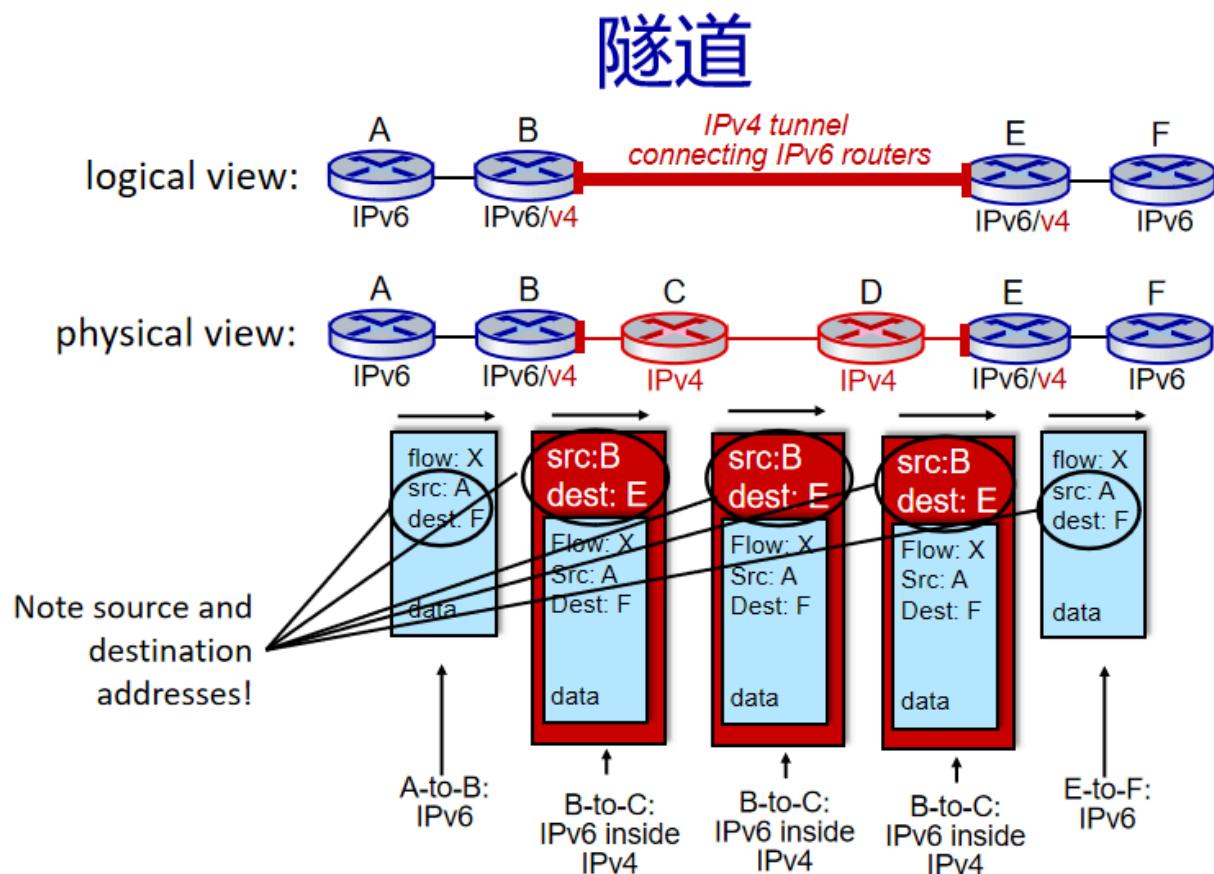
一、什么是隧道 (Tunneling)

背景：

由于 IPv4 网络还广泛存在，但现在逐渐部署 IPv6，因此需要一种机制让 IPv6 报文可以在 IPv4 网络中“穿行”——这就是“隧道”。

定义：

Tunneling 是指将一个协议（如 IPv6）的数据报封装到另一个协议（如 IPv4）中，以便通过该协议的网络进行传输。



1. Logical View (上方) :

- 从 IPv6 网络角度来看:



- 用户和上层协议感知不到中间是 IPv4 网络，逻辑上是一条 IPv6 路径。

2. Physical View (下方) :

真实传输过程更复杂，涉及双层封装:

段	实际传输内容	外部头部 (用于路由)	内部数据
A→B	原生 IPv6 报文	IPv6 src=A, dest=B	数据
B→C	IPv4 报文 (封装)	IPv4 src=B, dest=E	内层 IPv6 报文 src=A, dest=F
C→D	同上 (中转)	IPv4 src=B, dest=E	...
D→E	同上 (中转)	IPv4 src=B, dest=E	...
E→F	解封装后 IPv6	IPv6 src=A, dest=F	数据

注意:

- B 和 E 是关键的“封装/解封装点”;**
- 在 IPv4 中转路径上，IPV6 报文完全隐藏；**
- 网络只根据 IPv4 外层头部进行转发。**

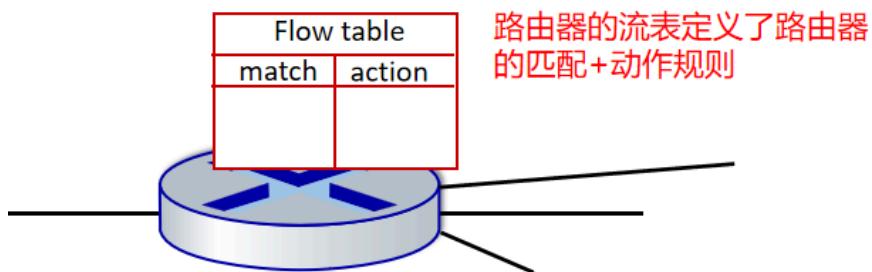
🧠 总结: Tunneling 的意义

特点	说明	操作
跨越不支持 IPv6 的网络	IPv6 报文能通过 IPv4 网络传输	无
封装机制	IPv6 报文作为 IPv4 的 payload	无
对端路由器透明	对用户和应用来说，这条路径看起来像是一条连续的 IPv6 通路	无
典型用例	IPv6 transition (IPv6 过渡) 机制，如 6to4、Teredo	无

通用转发和SDN

流表抽象

- **流 (flow)**: 由头部字段值定义 (在链路层、网络层、传输层字段中)
- **通用转发**: 简单的数据包处理规则
 - **匹配**: 数据包头字段中的模式值
 - **动作**: 对于匹配的数据包: 丢弃, 转发, 修改, 匹配的数据包或将匹配的数据包发送到控制器
 - **优先级**: 消除重叠模式
 - **计数器**: 计数 byte 和 packet 数量



第2页：OpenFlow 流表条目结构

每条 Flow Entry 包含三部分：

1. Match (匹配字段) :

- 可以是二层 (MAC) 、三层 (IP) 、四层 (TCP/UDP 端口) 等

2. Action (操作) :

- 支持以下操作:

1. 转发到指定端口 (forward)
2. 丢弃 (drop)
3. 修改头部字段 (如 IP 或 MAC)
4. 上送控制器 (封装后)

3. Stats (统计信息) :

- 每条流的 packet 数量、byte 数量，用于流量监控

总结：OpenFlow 的优势与核心机制

组件	说明
控制器	集中控制流表规则的下发 (控制平面)
交换机 (转发设备)	根据流表执行匹配与动作 (数据平面)
流表 (Flow Table)	每条规则由匹配字段 + 动作 + 统计组成
可支持功能	路由、防火墙、负载均衡、QoS 等

OpenFlow 抽象

- **匹配+动作 (match + action):** 抽象统一了不同种类的设备

Router

- **match:** 最长目的IP前缀
- **action:** 转发到链路

Firewall

- **match:** IP 地址和 TCP/UDP 端口号
- **action:** 允许或拒绝

Switch

- **match:** 目的MAC 地址
- **action:** 转发或洪泛

NAT

- **match:** IP 地址和端口
- **action:** 重写地址和端口

✓ 第4页：通用转发机制小结

核心思想：“匹配 + 动作”

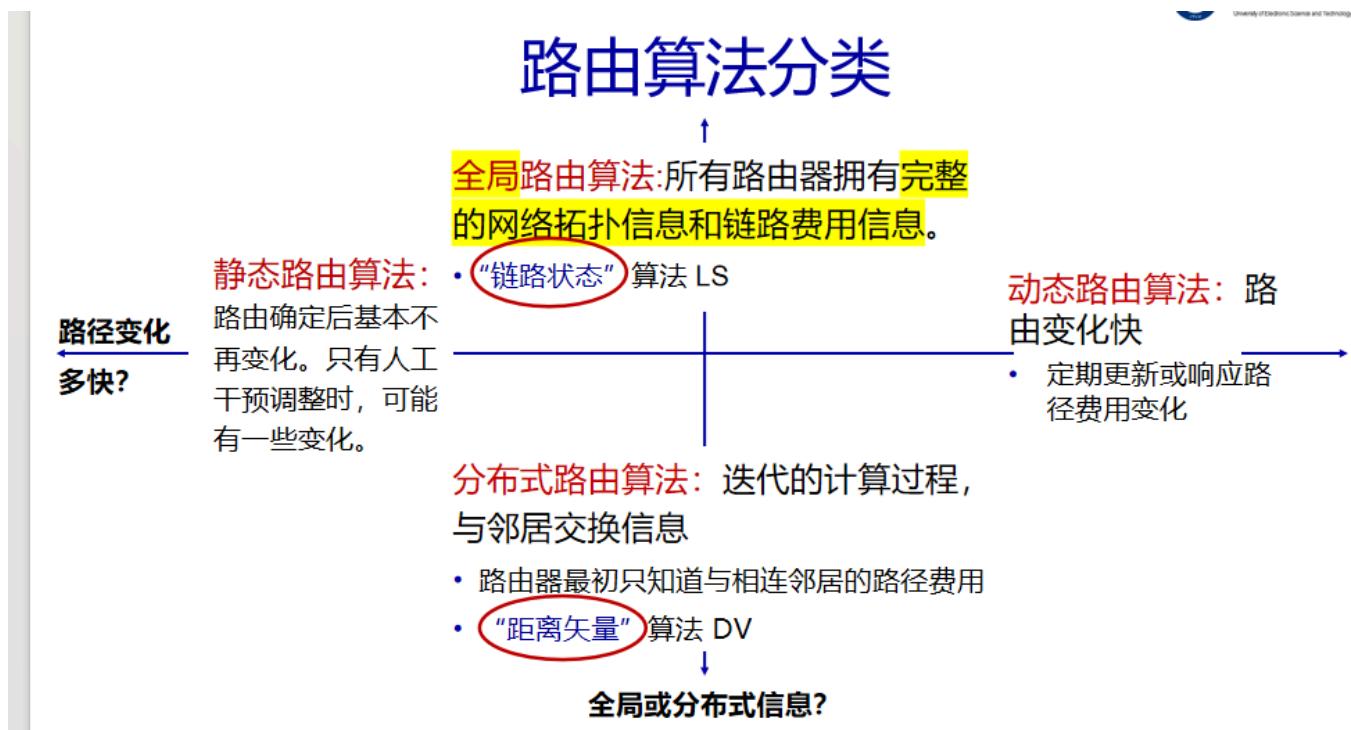
- **匹配:** 可以针对任意协议层（链路层、网络层、传输层）字段；
- **动作:** 丢弃、转发、修改、上送控制器等；
- 实现了将“网络行为”变为**编程行为 (programmable behavior)**。

通用抽象优势：

优点	说明
可编程	控制器编写规则、动态下发
灵活	可按需定制数据路径
可拓展	结合 P4 等语言可扩展协议字段匹配
解耦控制	网络设备只管转发，控制逻辑交由控制器

选路算法

分类



LS链路状态路由算法

一、LS 算法是啥？全称是什么？

LS 算法 (Link-State Algorithm)

即 **链路状态路由算法**，它是 Dijkstra 算法的基础。

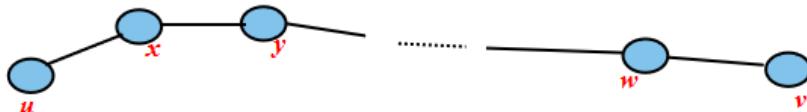
它是一类 **分布式路由算法**，由每个路由器独立运行，但它们共享同一个全网拓扑视图，通过 Dijkstra 算法计算最短路径。

Dijkstra

- **基本思想:** 以源节点为起点，每次找出一个到源节点的费用最低的节点，直到把所有的目的节点都找到为止。

▪ 术语定义

- $c(x, y)$: 表示从节点x到y的链路费用;
 $= \infty$ 如果不是直接邻居
- $D(v)$: 表示从源节点到目的节点v的当前路径的费用;
- $p(v)$: 表示从源节点到目的节点v的路径上的前驱节点(例如w是v的前驱节点);
- N' : 表示已经找到最低费用路径的节点集合。



路由震荡?

🧠 一、基本背景：什么是“路由振荡”？

定义：

当链路代价 (cost) 依赖于流量 (traffic load) 时，节点在运行 Dijkstra 算法计算路径后，导致网络中的流量重新分布，进而改变链路代价，进而再次改变路由选择，如此循环，就可能导致路由持续来回变化而无法稳定——这就是“路由振荡”。

距离向量路由算法DV算法

距离向量路由算法是一种迭代的、异步的和分布式的算法。

- 分布式：每个节点都从其直接相连邻居接收信息，进行计算，再将计算结果分发给邻居。
- 迭代：计算过程一直持续到邻居之间无更多信息交换为止。
- 异步：不要求所有节点相互之间步伐一致地操作。
- 自我终结：算法能自行停止。

基于 *Bellman-Ford* (BF) 方程 (动态规划):

Bellman-Ford equation

定义 $D_x(y)$: 节点 x 到节点 y 的最低费用路径的费用。
则:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

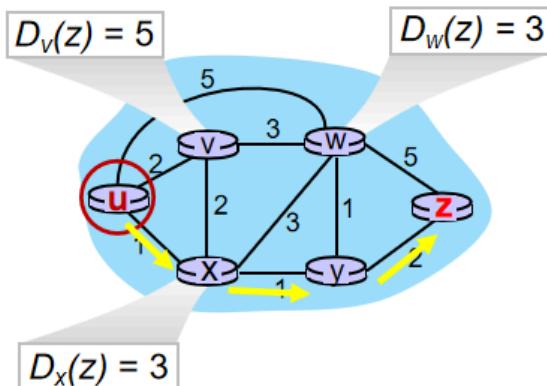
从 x 的所有邻居 v 中选取的最小费用

节点 v 到节点 y 的最低费用路径的费用

从 x 到 v 的直接链路费用

Bellman-Ford 例子

假设 u 的邻居节点 x, v, w 知道目的地 z :



Bellman-Ford 方程告诉我们:

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

得到节点 u 的转发表中到目的
节点 z 的下一跳是节点 x

关键思想:

- 每个节点都会将自己的距离矢量估计值发送给邻居
- 当 x 从任何邻居接收到新的DV估计值时, 它将使用B-F公式更新其自己的DV:

$$D_x(y) \leftarrow \min_v \{ c_{x,v} + D_v(y) \} \text{ for each node } y \in N$$

- 当距离向量不再变化时, 估计值 $D_x(y)$ 收敛到实际的最小费用 $d_x(y)$

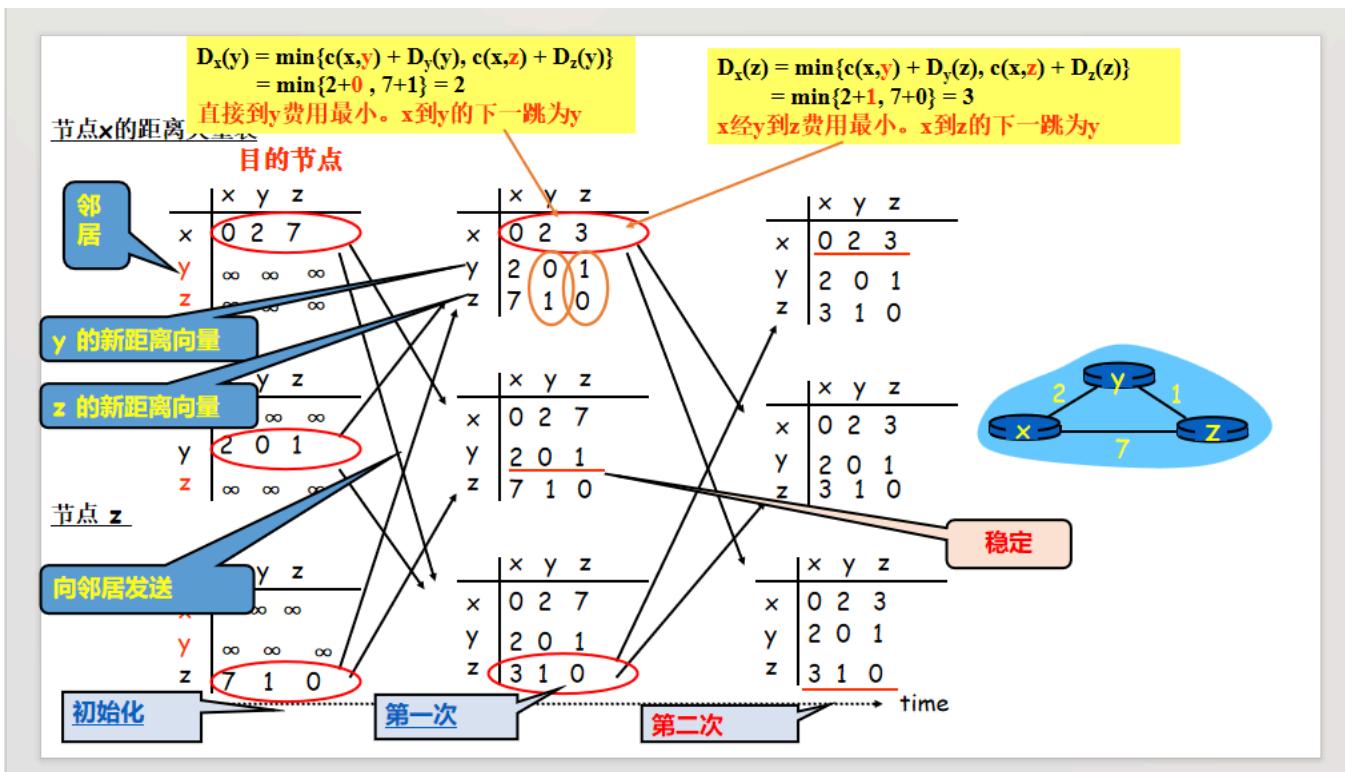
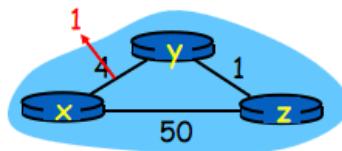
考虑y与z到目的节点x的距离表变化

- t0: y 检测到x的链路费用从4变为1，更新其距离向量，并通知其邻居z；
- t1: z收到来自y的更新报文，并更新自己的距离表，此时到节点x的最低费用减为2，并通知其邻居y；
- t2: y收到来自z的更新报文，并更新自己的距离表，此时到节点x的最低费用不变仍为1。不发送更新报文，算法静止。

当x与y之间费用减少，DV算法只需两次迭代到达静止状态。

节点之间链路费用减少的“好消息”
在网络中能迅速传播。

“good news travels fast”



LS算法与DV算法比较

▪ 消息复杂度：

- LS算法：知道网络每条链路的费用，需发送 $O(nE)$ 个报文；当一条链路的费用变化时，必须通知所有节点
- DV算法：迭代时，仅在两个直接相连邻居之间交换报文；当链路费用改变时，只有该链路相连的节点的最低费用路径发生改变时，才传播已改变的链路费用。

▪ 收敛速度：

- LS算法：需要 $O(nE)$ 个报文和 $O(n^2)$ 的搜寻，可能会振荡
- DV算法：收敛较慢。可能会遇到选路回环，或计数到无穷的问题。

✓ n 是节点（路由器）的集合， E 是边（链路）的集合

▪ 健壮性：当一台路由器发生故障、操作错误或受到破坏时，会发生什么情况？

- **LS算法：**路由器向其连接的一条链路广播不正确费用，路由计算基本独立（仅计算自己的转发表），有一定健壮性。
- **DV算法：**一个节点可向任意或所有目的节点发布其不正确的最低费用路径，一个节点的计算值会传递给它的邻居，并间接地传递给邻居的邻居。
一个不正确的计算值会扩散到整个网络。

层次选路

层次选路

网关路由器

- 一个区域内的路由器组成集合“**自治系统**”(AS, autonomous system)
- 同一个自治系统的路由器运行相同的路由协议——**区域内路由协议**
- 不同自治系统内的路由器可以运行不同的区域内路由协议

- 和其他自治系统内的路由器直接相连的路由器
 - 运行域间路由协议，与其他网关路由器交互
- 同自治系统内的所有其他路由器一样也运行域内路由协议

◀ ▶ ⏪ ⏩ ⏴ ⏵

域（自治系统）内路由选择

- 使用**域内路由协议**，也被称作**内部网关协议 (IGP)**
- 标准的域内路由协议：
 - **RIP**: 路由信息协议
 - **OSPF**: 开放式最短路径优先
 - **IGRP**: 内部网关路由协议 (Cisco 所有)

OSPF (Open Shortest Path First)

- “open” : 开放、公用的
- 用**链路状态算法**
 - 分发LS 分组
 - 每个节点具有拓扑图
 - 路由计算使用 Dijkstra算法
- 每个router都广播**OSPF通告**， OSPF通告里为每个邻居路由器设一个表项（记录每个邻居的链路特征和费用）。
- 通告会散布到 整个 自治系统 (通过洪泛法)
 - OSPF信息直接通过 IP传输 (不是 TCP 或 UDP)

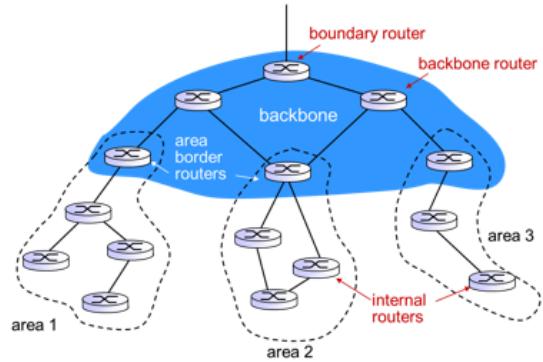
优点

▪ **OSPF 优点 (RIP所没有的)**

- **安全**: 所有OSPF 消息需要认证 (防止恶意入侵)
- 允许多个相同开销的路径 (在 RIP中只有一条路径)
- 对于每个链路，有多个消费尺度用于不同的服务类型TOS (例如在尽力转发时卫星链路代价设置为“低”，而对实时应用设置为高)
- **单播和多播综合支持**:
 - 多播 OSPF (MOSPF) 使用和 OSPF同样的链路数据库
 - 在大的区域中使用层次 OSPF

层次 OSPF

- 两级层次：本地区域、主干区域（这些区域都是在一个自治系统内）
 - 只在区域内发送链路状态通告
 - 每个节点有详细的区域拓扑；仅知道到达其他区域内网络的方向（即最短路径）
- 区域边界路由器**（同时属于本地区域和主干区域）：“汇总”了到本区域内部网络的路径，并通告给其他区域边界路由器。
- 主干路由器**：限于在主干区域内运行OSPF路由协议（本身不是区域边界路由器）
- 边界路由器**：连接到其他自治系统



Internet 域间选路: BGP

BGP（边界网关协议）是 Internet 上自治系统（AS）之间的核心路由协议，被称为“事实上的标准”。

二、AS 之间是如何互联和选路的？（图2）

每个 AS 内部运行自己的 IGP（如 OSPF），而 AS 之间靠 BGP 交换路由信息。

转发表配置说明：

- 域内转发**：使用 IGP（如 OSPF），负责本 AS 内部的最短路径转发
- 域间转发**：使用 BGP，决定数据从本 AS 发往另一个 AS 的哪条路径最优

图中：

- AS1 和 AS2、AS3 互联
- BGP 选择的是到达其他 AS 的最佳路径

四、示例：如何在 router 1d 上设置转发表？（图4）

设：

- AS1 运行 BGP
- 通过 BGP 学到：网络 X 可通过 AS3 的边界路由器 (1c) 访问
- BGP 将此信息传递给 AS1 所有内部路由器 (如 1d)

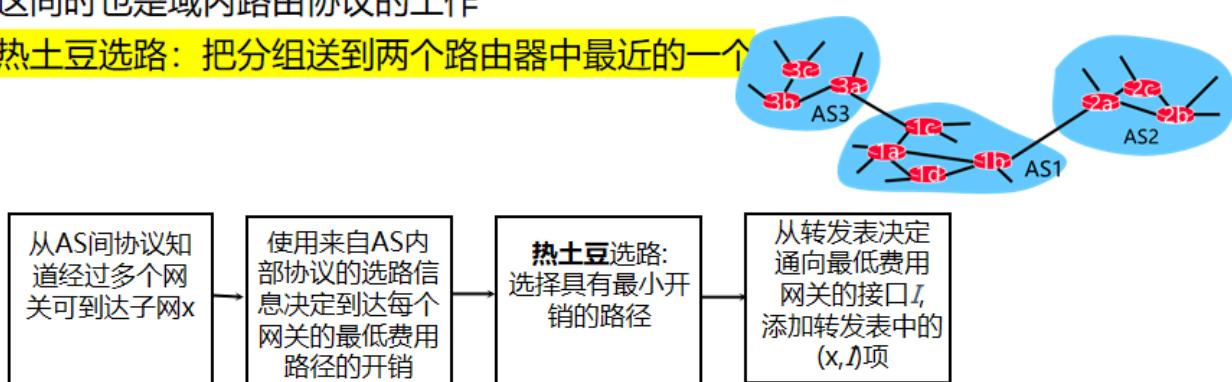
Router 1d 的行为如下：

1. 根据 IGP (如 OSPF) 得出从 1d 到 1c 的最短路径
2. 在自己的转发表里加入一项：`(x, 接口 I)`，表示去 x 要走接口 I (到达 1c 的路径)

Internet 域间选路: BGP

● 示例：在多个自治系统中选择

- 现在假设 AS1 通过域间选路协议知道子网 x 从 AS3 和 AS2 都可以到达
- 为了配置转发表，路由器 1d 必须决定通过哪个网关将分组转发到目的子网 x
- 这同时也是域内路由协议的工作
- **热土豆选路：**把分组送到两个路由器中最近的一个



一、什么是 BGP 会话 (BGP Session) ?

BGP 是一个**基于 TCP 的路由协议**。在实际运行中，**两个路由器通过 TCP 连接建立一个 BGP 会话**，用于交换路由信息。

特点：

- BGP 使用 **TCP 端口 179**
- 会话是**半永久的**：一旦建立，只要没有错误或手动重置，会一直保持连接
- **BGP 会话与物理链路无关**，即使两台路由器不直接相连，也可以建立 BGP 会话（如中间经过其他路由器转发）

当路由器得知一个新的前缀时，就在它的转发表中为该前缀创建一个项。

路径属性 和 BGP 路由

路径属性 和 BGP 路由

- 当通告前缀时，通告包含了BGP属性.
- **前缀 + 属性 = “路由”**
- 两个重要的属性：
 - **AS-PATH**: 包含了前缀的通告已经通告过的那些AS，如 AS 67 AS 17
 - **NEXT-HOP**: 指出到达下一个AS的具体AS间边界路由器（可能存在多条从当前AS到达下一个AS的链路）
 - 当网关路由器接收到路由通告时，使用**输入策略**来决定接收/舍弃该通告。

BGP 路由选择

BGP 路由选择

- 路由器可能知道到相同前缀的多条路由，路由器必须从中选择。
- **排除规则**（应用排除规则直到有一条留下）
 - **本地偏好值属性**: 具有最高偏好值的路由被选择
 - **最短 AS-PATH 的路由**
 - **最靠近 NEXT-HOP 路由器的路由**: **热土豆路由**
 - 其他标准

BGP 报文

BGP 报文

- BGP 报文交换使用 **TCP**
- BGP 报文：
 - **OPEN**: 建立到对方的TCP连接，并对发送者进行认证
 - **UPDATE**: 通告新路径(或者撤销旧路径)
 - **KEEPALIVE**: 在没有UPDATES时保持连结活跃；也对OPEN请求作出应答
 - **NOTIFICATION**: 报告前面报文的错误；也用于关闭连结

为什么AS内选路和AS间选路采用不同的协议？

为什么AS内选路和AS间选路采用不同的协议？

- 策略：

- **AS间**：管理员想控制本AS内产生的通信流怎样选路，以及什么通信流穿过自己的网络
- **AS内**：单个管理者，因此不需要策略

- 规模：

- 层次路由节省了转发表的大小空间，减少了路由更新的流量

- 性能：

- AS内：集中在性能上
- AS间：策略可能比性能更加重要

SDN

什么是SDN

- **软件定义网络 (SDN, Software Defined Network)** 源自美国斯坦福大学CLeanState研究组提出的一种新型网络创新架构，可通过软件编程的形式定义和控制网络，具有控制平面和转发平面分离及开放性可编程的特点。
- SDN的核心理念是，希望应用软件可以参与对网络的控制管理，满足上层业务需求，通过自动化业务部署，简化网络运维。
- SDN并不是一个具体的技术，它是一种网络设计理念，规划了网络的各个组成部分（软件、硬件、转发面和控制面）及相互之间的互动关系。

SDN体系结构的4个关键特征

■ 网络层

SDN体系结构的4个关键特征

4. 可编程的网络

routing
access control

...

load balance

3. 网络控制功能

(SDN控制器或网络操作系统、网络应用控制程序)

control plane

data plane

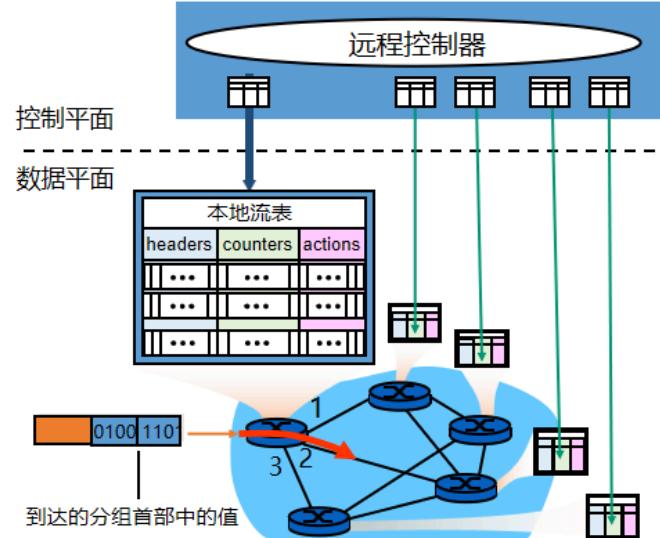
2. 控制平面、数据平面分离

1: 基于流的转发 (比如 OpenFlow)

SDN：通用转发

- SDN的核心思想是建立一个通用转发体系

——每个交换设备包含一个流表(flow table)。流表由一个逻辑上中心化的控制器(远程控制器)来计算和分发。

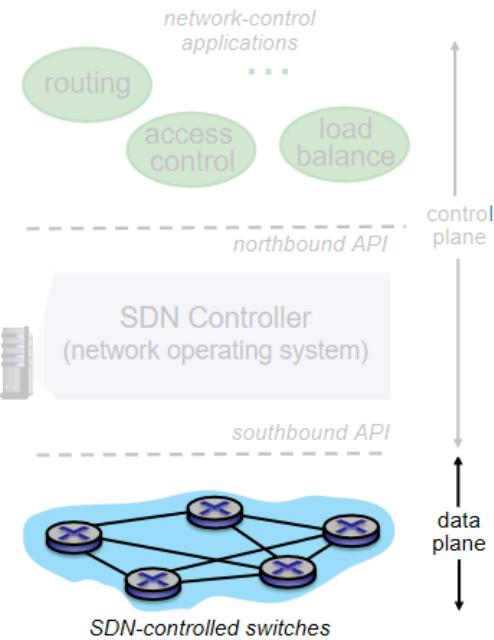


SDN—数据平面交换机

数据平面交换机 Data-plane

switches:

- 快速、简单的商用交换机在硬件中实现了通用的数据平面转发功能 (第4.4节)
- 流量 (转发) 表在控制器的监管下计算、安装
- 基于表的交换控制 (例如OpenFlow) API
 - 定义了哪些是可控制的, 哪些不是可控制的
- 与控制器通信的协议 (例如, OpenFlow)

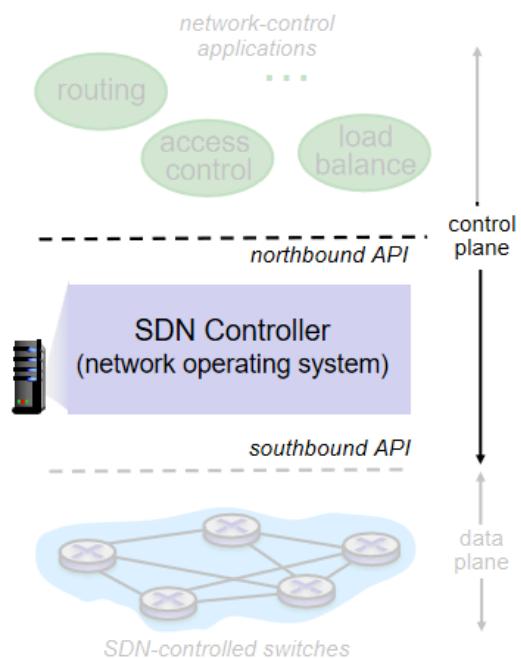


SDN—控制器

SDN—控制器

SDN controller (network OS):

- 维护网络状态信息
- 通过北向API与“上方”的网络控制应用程序进行交互
- 通过南向API与“下方”的网络交换机交互
- 采用分布式系统, 保证性能、可伸缩性、容错性和鲁棒性



网络层：总结

- 我们已经学习了：
 - 网络层服务
 - 路由原理: 链路状态和距离矢量
 - 层次路由
 - IP
 - Internet 选择协议 RIP, OSPF, BGP
 - 路由器的内部结构
 - IPv6
- 网络层提供的服务和功能
 - 主机通信
 - 虚电路和数据报
 - 转发
 - 选路
- 路由器工作原理
- 网际协议IP
 - IP报文、IP分片和重组
 - IP编址和IP子网
 - IP地址分类和无分类编址CIDR
 - NAT网络地址转换
 - IPv6协议及特点、IPv4和IPv6互通

- 选路算法
 - 链路状态选路算法
 - 距离向量算法
 - 层次选路
- 因特网中的选路协议
 - 内部网关协议：RIP、OSPF、IGRP
 - 外部网关协议：BGP
- SDN
 - 概念，用途，架构
 - OpenFlow

五、数据链路层

6.1 链路层概述

6.3 多路访问链路和协议

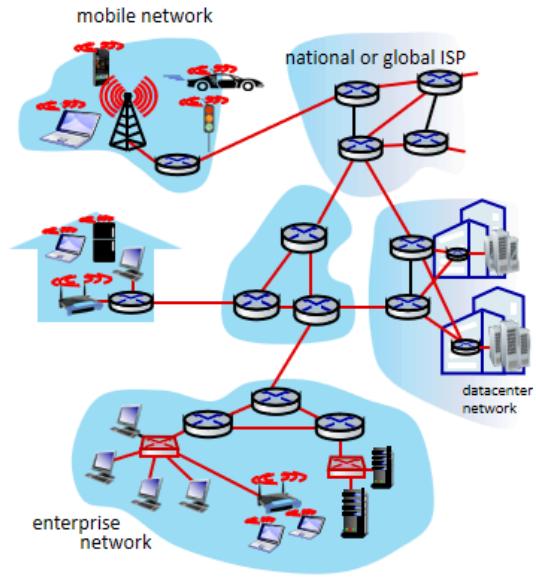
6.2 差错检测和纠错

6.4 交换局域网

链路层的术语

- 主机和路由器: **节点(nodes)**
- 沿着通信路径连接相邻节点的通信信道: **链路(links)**
 - 有线链路(wired links)
 - 无线链路(wireless links)
- 第二层的分组: **数据帧(frame)**, 它是封装了的数据报

数据链路层的职责是将数据报从一个节点传送到与该节点**直接**有物理链路相连的另一个节点。



链路层提供的服务

- **封装成帧, 链路接入(framing, link access):**
 - 封装数据报为数据帧, 增加头部, 尾部信息
 - 如果是共享链路, 首先需要接入到链路
 - 在数据帧头部中, 用MAC地址来标识源目的MAC地址
 - 不同于IP地址
- **在相邻节点之间可靠传输数据帧**
 - 我们在第3章已经学习了如何在运输层实现数据的可靠传输
 - 在比特错误率很低的链路(光纤、双绞线)很少使用
 - 无线链路: 高比特错误率
 - 问题: 为什么要在链路层和端到端都实现可靠传输?

- **流量控制(flow control):**
 - 用于控制发送节点向直接相连的接收节点发送数据帧的频率
- **差错检测(error detection):**
 - 差错可能由信号衰减、噪声引入
 - 接收方检测是否出现错误：
 - 通知发送方重传或丢弃数据帧
- **错误纠正(error correction):**
 - 接收方标识和纠正比特错误，而不需要请求重传
- **半双工和全双工(half-duplex and full-duplex):**
 - 在半双工模式，链路的两个节点都可以发送数据，但是不能同时发送

为什么要在链路层和端到端都实现可靠传输？

三、为什么两者都要有？

✓ 原因1：分层职责不同

- 链路层只对一个跳负责；它不知道全局路径，也无法保证多跳的可靠性。
- 传输层负责的是端到端整个通信的完整性与顺序等，它掌握连接的整体状态。

✓ 原因2：提高效率，分担负担

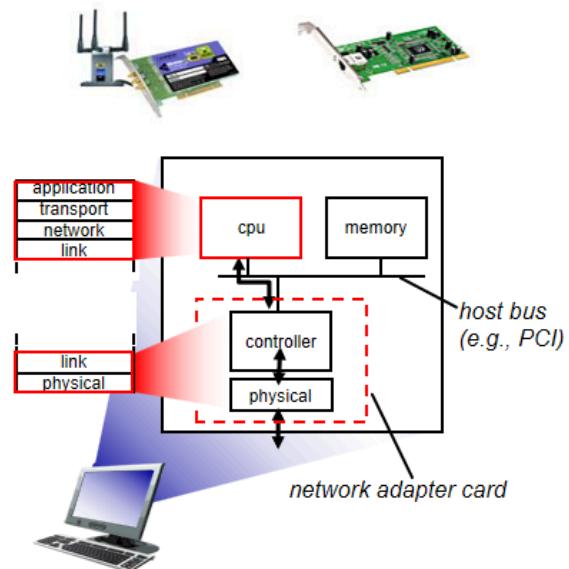
- 如果每一层都依赖 TCP 端到端重传，会导致重传成本高（整段路径再走一遍）。
- 链路层重传能就近修复错误，减少端到端的重传压力，提高性能。

✓ 原因3：并非所有链路都需要可靠性

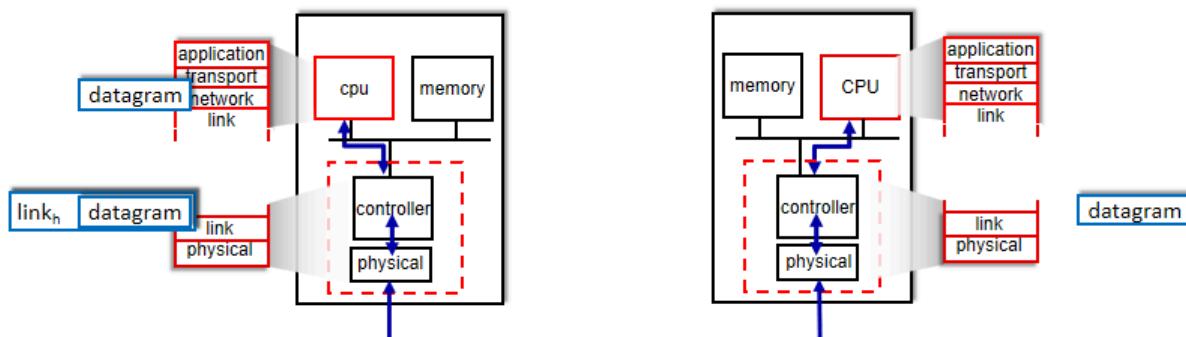
- 有些链路如有线以太网极其可靠，不实现链路层可靠机制可以简化设计、提高速度。
- 有些链路如无线信道误码率高，需要链路层就地修复。

6.1.2 链路层实现的位置

- 在主机和网络设备(路由器)上实现
- 在主机上，链路层的主体部分是在**网络适配器**上实现的(称为网卡)
 - 以太网卡，802.11卡；以太网芯片组
 - 实现链路层和物理层的功能
- 硬件、软件、固件的组合



接口通信



发送方：

- 封装数据报为数据帧
- 增加差错检测比特，**可靠数据传输**，**流量控制等机制**。

接收方：

- 执行**检查错误**、**可靠数据传输**、**流量控制等机制**
- 抽取数据报，将其递交给上层

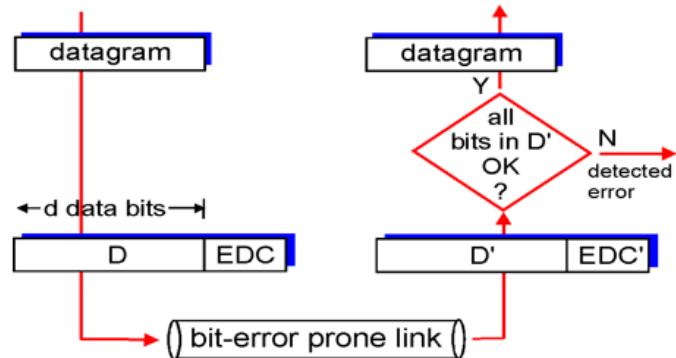
差错检测和纠错

• 比特级差错检测和纠错

- 对一个节点发送到一个相邻节点的帧，检测是否出现比特差错，并纠正。
- 相关技术很多。
- 差错检测和纠错的过程

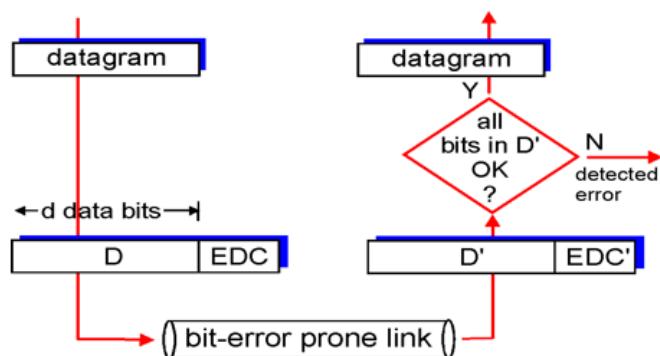
• 差错检测并非100%可靠

- 协议可能丢失一些错误
- 差错校验位越多，检测和纠正功能越好



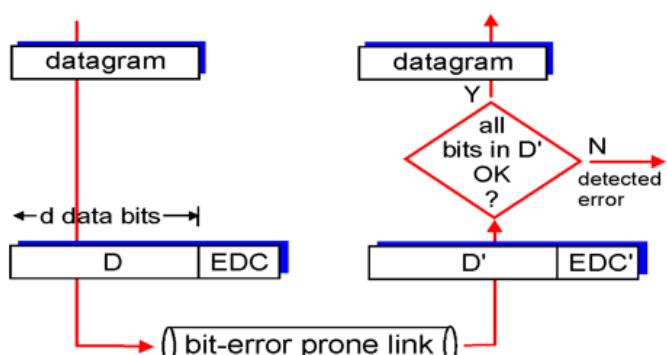
• 发送节点

- 将数据D附加若干差错检测和纠错位EDC，一起发送到链路。
- 数据D包括网络层传来的数据报，以及链路级寻址信息、序列号和其他字段。
- 保护范围包括数据D的所有字段。



• 接收节点

- 接收比特序列D'和EDC'。
- 如果发生传输比特错误 ($0 \rightarrow 1$, $1 \rightarrow 0$)，D'和EDC'可能与发送的D和EDC不同。
- 接收方根据D'和EDC'，判断D'是否和初始的D相同 (D的传输是否正确)。
- 正确：解封取出数据报，交给网络层；
- 出错：差错处理。



- 差错检测和纠正技术不能保证接收方检测到所有的比特差错，即可能出现未检测到的比特差错，而接收方并未发现。
- 选择一个合适的差错检测方案使未检测到的情况发生的概率很小即可。
- 差错检测和纠错技术越好，越复杂，开销更大。

三种主要差错检测技术

· **奇偶校验**：最基本的方法。

· **Internet校验和**：常用于运输层。

· **循环冗余检测**：常用于链路层。

比特奇偶校验

• 发送方：

- 在要发送的信息D（d位）后面附加一个奇偶校验位
- 使“1”的个数是奇数（奇校验）或偶数（偶校验）
- 一起传输发送（d+1位）。



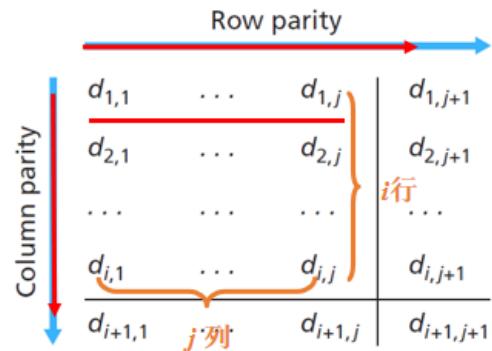
• 接收方：

- 检测收到的信息（d+1位）中“1”的个数。
- 偶校验：发现奇数个“1”，至少有一个比特发生差错（奇数个比特差错）。
- 奇校验：发现偶数个“1”，至少有一个比特发生差错。
- 可以查出任意奇数个错误，但不能发现偶数个错误。
- 若比特差错概率很小，差错独立发生，一比特奇偶校验可满足要求。
- 若差错集中一起“突发”（突发差错），一帧中未检测到的差错的概率达到50%。

二维奇偶校验

• 基本思想：

- 将要传信息D (d比特) 划分为 i 行 j 列 (i 个组, 每组 j 位)；
- 对每行和每列分别计算奇偶值；
- 结果的 $i+j+1$ 个奇偶比特构成了帧的差错检测比特。



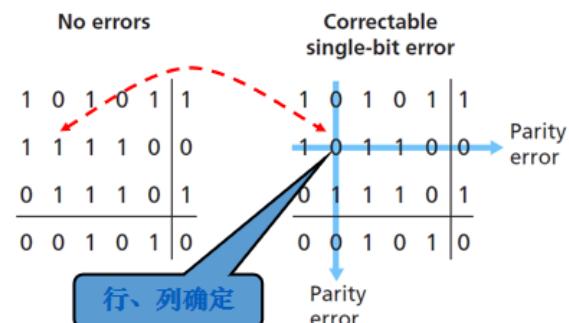
二维奇偶校验的例子

要发送的数据比特10101 11110 01110,

划分3组, 每组5个比特。进行行、列偶校验

特点：

- 可以检测并纠正单个比特差错 (数据或校验位中)。
- 能够检测(但不能纠正)分组中任意两个比特的差错。



接收方可以检测并纠正差错的能力称为前向差错纠正FEC。

- 可与ARQ技术一起应用, 接收方立即纠正差错, 减少发送方重发的次数。
- 降低分组传输的往返传播时延, 适用于实时网络应用。

19

步骤：右下角的校验位怎么得来？

计算规则：它保证所有校验位（行+列）之和也满足偶校验

在二维奇偶校验方法中不可能纠正两位错误。

发送方:

- 将数据的每两个字节当作一个16位的整数，可分成若干整数；
- 将所有16位的整数求和；
- 对得到的和逐位取反，作为检查和，放在报文段首部，一起发送。

接收方:

对接收到的信息(包括检查和)按与发送方相同的方法求和。

- 全“1”：收到的数据无差错；
- 其中有“0”：收到的数据出现差错。

或者核对计算的检查和是否等于检查和字段的值。



Internet校验和的特点

- 分组开销小：检查和位数比较少；
- 差错检测能力弱：
- 适用于运输层（差错检测用软件实现，检查和方法简单、快速）。
- 链路层的差错检测由适配器中专用的硬件实现，采用更强的CRC方法。

循环冗余检测

计算机网络中广泛采用

$$10111 \rightarrow x^4 + x^2 + x + 1$$

- 循环冗余检测CRC (cyclic redundancy check) 编码：**

- 即多项式编码，把要发送的比特串看作为系数是0或1的一个多项式，对比特串的操作看作为多项式运算。

- 基本思想：**

- 设发送节点要把数据 D (d 比特) 发送给接收节点。
- 发送方和接收方先共同选定一个生成多项式 G ($r+1$ 比特)，最高有效位(最左边)是1。

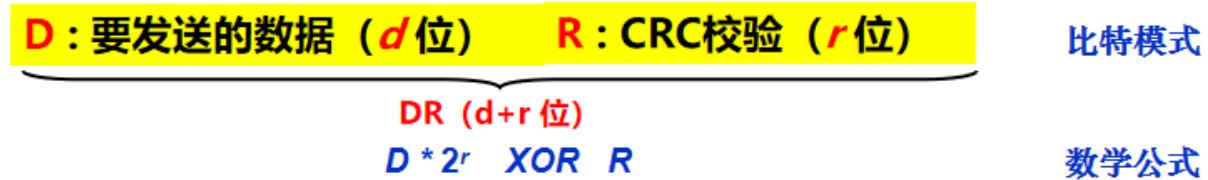
循环冗余检测的基本思想

发送方：

- 计算出一个 r 位附加比特 R , 添加到 D 的后面产生 DR ($d+r$ 比特)
- DR 能被生成多项式 G 模2运算整除, 一起发送。

• 接收方：

- 用生成多项式 G 去除接收到的 DR ($d+r$ 比特)
 - 余数非0: 传输发生差错;
 - 余数为0: 传输正确, 去掉尾部 r 位, 得所需数据 D 。



模2运算

什么是模2运算

- 加法不进位, 减法不借位, 即操作数按位异或 (XOR)

例

$$1011 \text{ XOR } 0101 = 1110 ; \quad 1011 - 0101 = 1110$$

$$1001 \text{ XOR } 1101 = 0100 ; \quad 1001 - 1101 = 0100$$

- 乘法和除法与二进制运算类似, 其中加法或减法没有进位或借位。
- 乘以 2^r , 即比特模式左移 r 个位置。

$$\begin{aligned} D \times 2^r \text{ XOR } R &= D00\dots00 \text{ XOR } R \\ &= DR \text{ (d+r 比特)} \end{aligned}$$

“相同为0, 不同为1”

计算R (CRC比特)

- DR能被G模2运算整除：即

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

- 等式两边都用R异或，得到

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

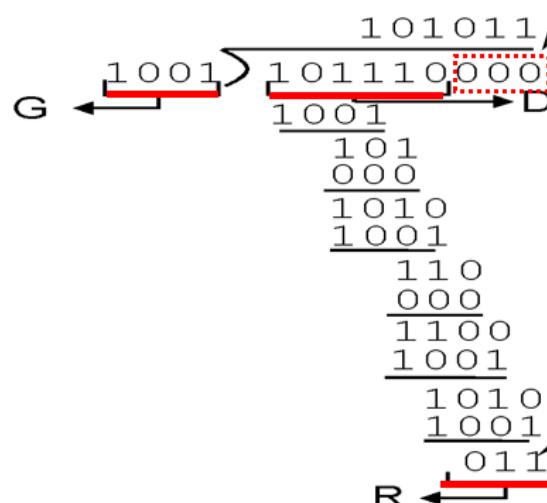
- 即用G来除 $D \times 2^r$ ，余数值刚好为R。

R的计算：将数据D后面添加 r 个0，除以给定的生成多项式G，所得余数即为R (r 位)。

$$R = \text{remainder}[\frac{D \cdot 2^r}{G}]$$

CRC编码的例子

设 (数据) $D = 101110, d = 6, G$ (生成多项式) $= 1001, r = 3$



实际传输的数据形式是：
101110 011

CRC练习1

假设：

➤通信双方协商的生成多项式为：

$$G = X^4 + X^2 + X + 1$$

➤发送方要发送的数据为：

$$D = 11001100$$

问题答案：

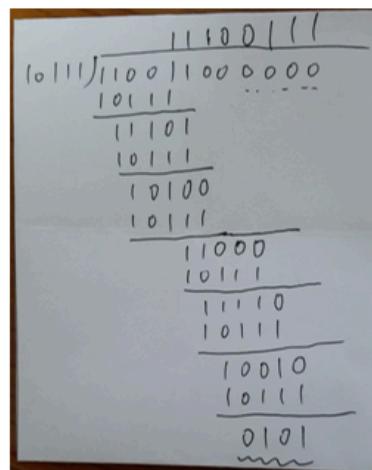
➤CRC校验信息需要多少位？—— 4位

➤发送方最终发送的数据是多少位？—— 12位

➤发送方最终发送的数据内容是什么？—— 11001100 0101

➤如果传输过程中出现一位错误能否检测？出现六位错误能否检测？

—— 一位错误可检测到，六位错误一般情况下不可以（见教材P301）



循环冗余码CRC的特点

- 生成多项式G的选择：常见的有8、12、16和32比特生成多项式G。
- 国际标准已经定义了8-、16-、32-位生成多项式G；8-位CRC用于ATM信元首部的保护；32-CRC用于大量链路层IEEE协议。其他检错方法不常用，故不作专门介绍
 - CRC8生成多项式为 $G(x) = x^8 + x^5 + x^4 + 1$
 - CRC16生成多项式为 $G(x) = x^{16} + x^{12} + x^5 + 1$
 - CRC-32生成多项式为 $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{11} + x^{10} + x^{16} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

- CRC特点：

能检测小于 $r+1$ 位的突发差错（所有连续的 r 比特或者更少的差错可以检测到）、任何奇数个比特差错。

差错检测方法比较

- 奇偶校验能力最弱，CRC校验能力最强。
- 奇偶校验通常用于简单的串口通信
- Internet校验和通常用于网络层及其之上的层次，要求简单快速的软件实现方式
- CRC通常应用于链路层，一般由适配器硬件实现

多路访问链路和协议

- 两种网络链路：
 - 点对点链路：链路两端各一个节点。一个发送和一个接收。如点对点协议PPP。
 - 广播链路：多个节点连接到一个共享的广播信道。

广播：任何一个节点传输一帧时，信号在信道上广播，其他节点都可以收到一个拷贝。常用于局域网LAN中，如早期的以太网和无线局域网。

本节主要学习**广播链路的信道共享技术**。

多路访问协议

- 目的：**协调多个节点在共享广播信道上的传输。**
 - 避免多个节点同时使用信道，发生冲突（碰撞），产生互相干扰。
- 冲突（collide）：**两个以上的节点同时传输帧，使接收方收不到正确的帧（所有冲突的帧都受损丢失）。**
 - 造成广播信道时间的浪费。
 - 多路访问协议可用于许多不同的网络环境，如有线和无线局域网、卫星网等。

多路访问协议类型（三类）

- **信道划分协议**

- 把信道划分为小“片”（时隙）
- 给节点分配专用的小“片”

- **随机访问协议**

- 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，要发送的节点越多轮流时间越长

1.1 时分多路访问

6.3.1 信道划分协议



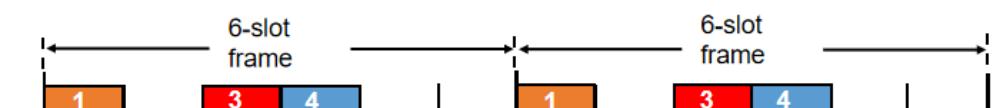
主要有TDMA、FDMA、CDMA三种。

设信道支持 N 个节点，传输速率是 R b/s。

- **时分多路访问TDMA (time division multiple access):**

将时间划分为**时间帧**，每个时间帧再划分为 N 个**时隙**（长度保证发送一个分组），分别分配给 N 个节点。**每个节点只在固定分配的时隙中传输。**

例：6个站点的LAN，时隙1、3、4有分组，时隙2、5、6空闲



特点

TDMA的特点：

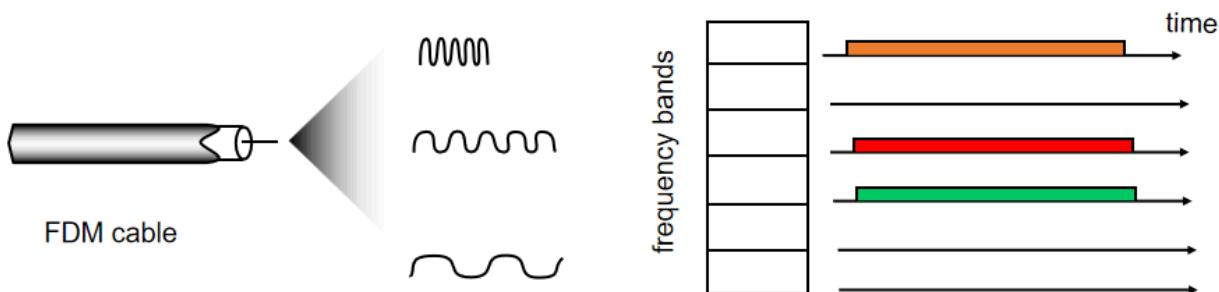
- 避免冲突、公平：每个节点专用速率 R/N b/s。
- 节点速率有限： R/N b/s；
- 效率不高：节点必须等待它的传输时隙。

1.2 频分多路访问FDMA

• 频分多路访问FDMA (frequency division multiple access)：

将总信道带宽 R b/s 划分为 N 个较小信道（频段，带宽为 R/N ），分别分配给 N 个节点。例：6个站点的LAN，频带1、3、4 有分组，频带2、5、6 空闲
特点：

- 避免冲突、公平： N 个节点公平划分带宽；
- 节点带宽有限、效率不高：节点带宽为 R/N 。



1.3 码分多路访问CDMA

• 码分多路访问CDMA (frequency division multiple access)：

- 每个节点分配一个唯一的编码
- 每个节点用它唯一的编码来对它发送的数据进行编码
- 允许多个节点“共存”，信号可叠加，即可以同时传输数据而无冲突
(如果编码是“正交化”的)

6.3.2 随机访问协议

基本思想：

- 发送节点以信道全部速率 (R b/s) 发送；
- 发生冲突时，冲突的每个节点分别等待一个随机时间，再重发，直到帧(分组)发送成功
- 节点间没有协调者

典型随机访问协议：

- ALOHA协议(纯ALOHA, 时隙ALOHA)
- 载波监听多路访问CSMA协议
- 带冲突检测的载波监听多路访问CSMA/CD
- 带冲突避免的载波监听多路访问CSMA/CA

ALOHA

CSMA (载波侦听多路访问)

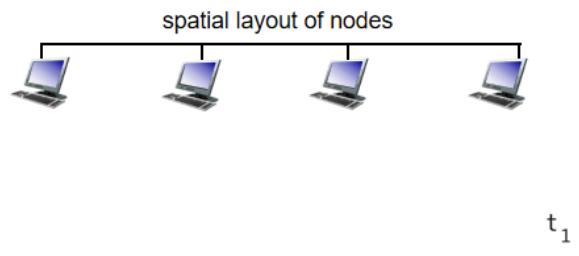
CSMA (载波侦听多路访问)

- **载波侦听 CS:** 某个节点在发送之前，先监听信道。
 - **信道忙:** 有其他节点正往信道发送帧，该节点随机等待（回退）一段时间，然后再侦听信道。
 - **信道空:** 该节点开始传输整个数据帧。
- **冲突检测 CD:** 边发送边监听，即节点在传输同时侦听信道。
 - 如果检测到有其他节点正在传输帧，发生冲突，立即停止传输，并用某种方法来决定何时再重新传输。
- **人类类比:** 自己说话之前，先听一下有没有其他人正在说话，不要打断他人说话！
- **CSMA 的特点:**
 - 发前监听，可减少冲突。
 - 由于传播时延的存在，仍有可能出现冲突，并造成信道浪费。

- 时间 t_0 : 节点B侦听到信道空, 开始传输帧, 沿着媒体传播比特。

- 时间 t_1 ($t_1 > t_0$) : 节点D有帧要发送。B的传输信号未到D, D检测到信道空, 开始传输。很快, B的传输开始在D节点干扰D的传输 (冲突)

端到端信道传播时延: 信号从一个节点到另一个节点所花费的传播时间。传播时延越长, 节点不能侦听到另一个节点已经开始传输的可能性越大。



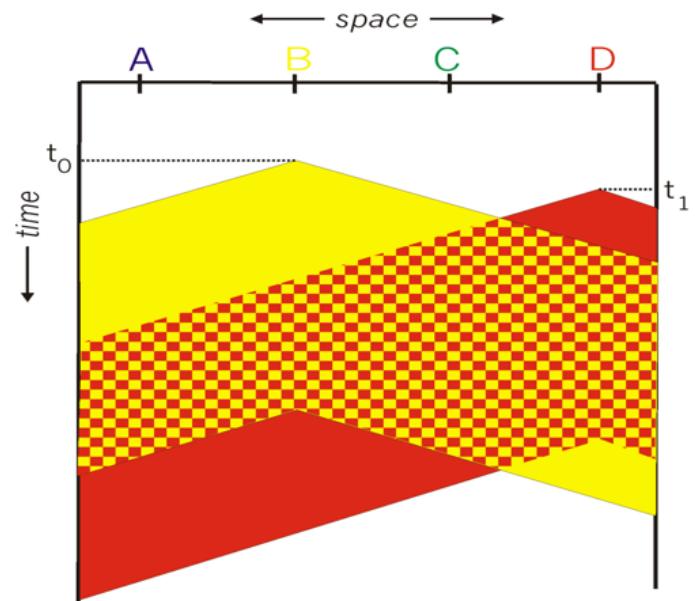
带来问题：信道浪费



- 节点没有进行冲突检测, 即使发生了冲突, 节点仍继续传输它们的帧。但该帧已经被破坏、是无用的帧, 信道传输时间被浪费。

注意:

距离与传播时延对碰撞概率的影响。



带冲突检测的CSMA(CSMA/CD)

增加“载波监听”和“冲突检测”两个规则。

• 基本原理: 传送前侦听

- 信道忙: 延迟传送
- 信道闲: 传送整个帧
- 发送同时进行冲突检测: 一旦检测到冲突就立即停止传输, 尽快重发。
- 目的: 缩短无效传送时间, 提高信道的利用率。

CSMA/CD: 以太网的多路访问协议

相应标准为IEEE802.3。

• 机制:

- ✓ **未使用时隙:** 适配器可以在**任何时刻开始**传输；
- ✓ **使用载波侦听:** 当适配器侦听到有其他的适配器在传输，就不会传输帧；
- ✓ **使用冲突检测:** 当检测到其他适配器也在传输帧，就中止传输；
- ✓ **重传:** 冲突后，等待一个随机时间（**小于**传输一帧的时间），再重传。

□ 说明:

- ✓ 侦听和冲突检测，由以太网适配器通过**测量传输前和传输过程中的电压等级**来进行。
- ✓ 各适配器**独立运行CSMA/CD协议**，不需和其他适配器进行协调。

CSMA/CD协议工作流程

(1) 封装成帧:

发送适配器从父节点获得一个网络层数据报，封装成以太网帧，放到缓冲区中；

(2) 适配器侦听信道:

- ✓ **空闲:** 即在96比特时间内（帧间最小间隔9.6μs），没有信号从信道进入，开始传输该帧；
- ✓ **忙:** 等待，直至侦听不到信号(加上96 比特时间)，开始传输该帧。

(3) 无冲突成功传输:

整个帧传输期间未检测到其他适配器的信号，该帧传输成功。

(4) 有冲突停止传输:

传输时检测到其他适配器的信号，就停止传输帧，并**传输一个48 比特的拥塞信号**。

(5) 等待随机时间再侦听:

传输拥塞信号后，适配器进入**指数回退**阶段，等待一段时间，并返回到第2步。

CSMA/CD 协议要求:

发送数据的帧必须在信号往返传播时间内尚未发送完毕，否则发送端无法检测到冲突。

也就是说：

- **最短帧的发送时间 ≥ 信号的往返传播时间**

假定1km长的CSMA/CD网络的数据传输速率为1 Gb/s。设信号在网络上的传播速率为 200,000 km/s。求能够使用此协议的最短帧长。

对于1km电缆，单程传播时间为 $1/200000 = 5 \mu\text{s}$ ，往返传播时间为 10 μs ，为了能够按照CSMA/CD工作，最小帧的发射时间不能小于 10 μs 。

以Gb/s速率工作，10 μs 可以发送的比特数等于 $10 * 10^{-6} / 10^{-9} = 10000$ ，因此，最短帧是10000 bit 或1250 字节长

最短帧长*最短帧发送时间

以太网CSMA/CD的运行机制



以太网CSMA/CD的运行机制

1. 适配器从网络层得到分组，创建帧
2. 如果适配器侦听到信道空闲，开始传送帧。如果信道忙，它会等到信道空闲才传送帧
3. 如果适配器传送整个帧时，都没有检测到其它传输，则完成该帧的传送
4. 如果适配器在发送中检测到其它传送，就放弃传送，并发送一个拥塞信号
5. 放弃传送后，适配器进入**指数回退阶段**，即该帧经过 n 次冲突后，适配器在**{0,1,2,...,2^m-1}** 中随机选取一个 k 值，其中 $m=\min(n,10)$ ，然后等待 $k*512$ 比特时间后，回到第2步

二进制指数回退算法

以太网CSMA/CD的运行机制讨论

拥塞信号

- 48比特，确保所有传送者知道冲突发生

比特时间

- 对于10 Mbps Ethernet 为0.1微秒，当 $k=1023$ ，等待时间大约50毫秒

二进制指数回退算法

- 目标：适配器依据当前负载情况重传，重负载时等待时间变长
- 第一次冲突：在 {0,1} 中选 k 值；延迟 $k \times 512$ 比特时间传送
- 第二次冲突：在 {0,1,2,3} 中选 k 值...
- **10次以后，在 {0,1,2,3,4,...,1023} 中选 k 值。**
- **k 是等概率选择**

CSMA/CD效率

当许多活动节点有大量的帧要发送时，帧在信道中无冲突传输的时间所占份额。

- ✓ 当只有一个节点有一个帧要发送时，该节点能够以最大速率（全速）传输。
- ✓ 如果很多节点都有帧要发送，信道的有效传输速率可能小的多。

近似公式：
$$\text{效率} = \frac{1}{1 + 5 t_{prop} / t_{trans}}$$

✓ t_{prop} ：信号在任意两个适配器之间传播所需的最大时间

✓ t_{trans} ：传输一个最大长度的以太网帧的时间。

最大长度的帧是1518字节，最小长度的帧是64字节（不包括前同步码）。

对10Mb/s的以太网，约为1.2ms。 $1518 \times 8 / 10M$

✓ 当 t_{prop} 接近0时：效率接近1。即如果传播时延是0，冲突节点将立即中止而不会浪费信道。

✓ 当 t_{trans} 变得很大时：效率也接近于1。

即若一个帧夺取了信道，将占有信道很长时间，信道在大多数时间都会有效地工作。

3. 轮流协议

轮询协议 (Polling)

1. 轮询协议 (Polling)

机制：

- 网络中设置一个**主节点 (master)**；
- 主节点轮流“**询问**”各个从节点是否有数据发送；
- 被轮到的从节点如果有数据，就发送；
- 没有数据，则跳过进入下一个节点。

特点：

优点	缺点
简单、避免冲突	主节点负担重， 单点故障
有顺序、易控制	增加轮询开销、效率可能低

类比：

像老师点名：“1号你要说话吗？2号你呢？3号呢？”

机制：

- 在网络中维护一个**“令牌 (Token) ”**，是一种特殊的控制帧；
- 节点收到令牌后才有权发送数据；
- 发送完数据后，把令牌传给下一个节点；
- 没有数据要发，也必须把令牌传下去。

特点：

优点	缺点
避免冲突、无需主控	若令牌丢失，需要机制恢复
公平、适用于大流量	有延时和管理开销

类比：

像传话筒：只有拿到话筒的同学才能发言，发完必须传给下一个人。

多路访问控制协议的总结

- **信道划分：**时分，频分，码分

- **随机接入：**

- ALOHA, S-ALOHA, CSMA, CSMA/CD
- 载波侦听：在某些技术中容易实现(有线)，在有些技术中比较困难(无线)
- CSMA/CD used in Ethernet
- CSMA/CA used in 802.11

- **轮流**

- 来自中心站的轮询
- 令牌传递

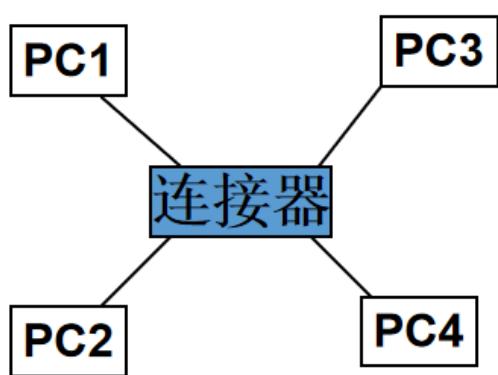
交换局域网

局域网概述

- 主要特点：网络为一个组织所拥有，且地理范围和站点数目均有限
- 局域网按**拓扑结构**进行分类：星形网、环形网、总线网、树形网和网状网

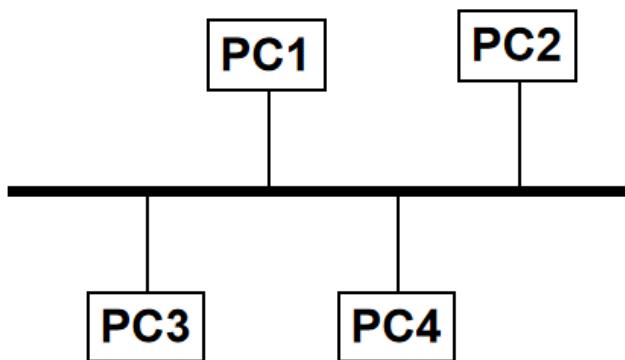
常见的网络拓扑结构

- 星型结构



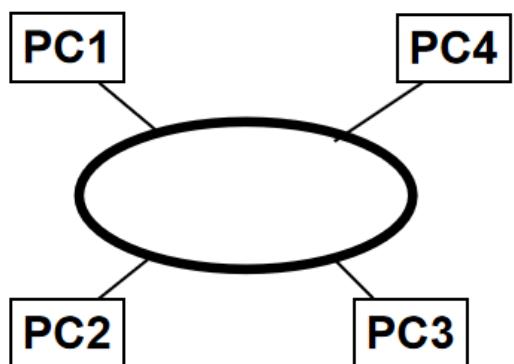
- 辐射状连接
- 中央结点集中式通信控制
- **优点:** 结构简单，访问协议简单，单个节点的故障不会影响到整个网络。
- **缺点:** 对中央结点的可靠性要求很高，一有故障，全网瘫痪。

- 总线结构



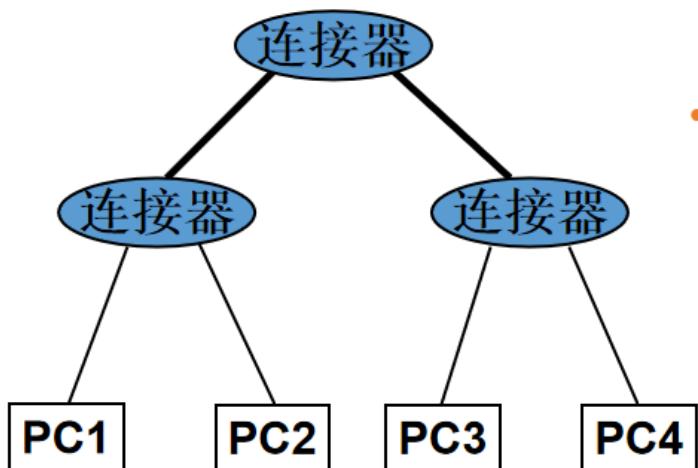
- 所有的站点都连接在同一根传输线，即“总线”上
- **优点:** 结构简单，易于扩充
- **缺点:** 故障检测比较困难

- 环型结构



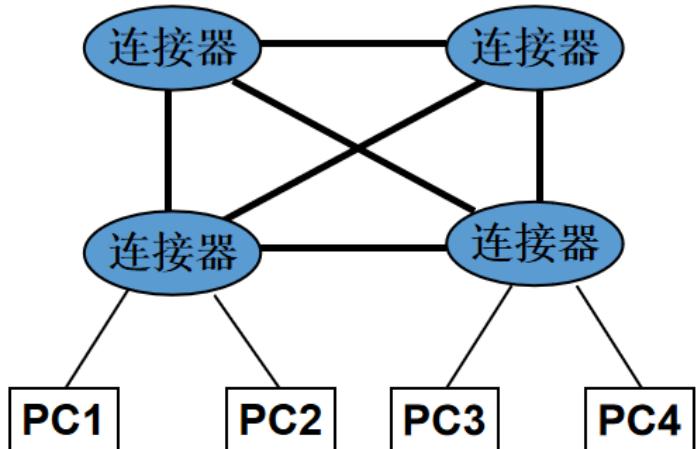
- 站与站点之间首尾相接，形成一个环，数据只能沿单方向传输
- **优点：**这种结构适合于光纤介质。实时性较强
- **缺点：**如果处理不当，站点的故障会引起全网故障

- 树型结构



- 它是从星型拓扑演变而来的，形状像一棵倒挂的树
- **特点：**与星型拓扑大致相似。它与星型结构相比降低了通信线路成本，增加了网络复杂性

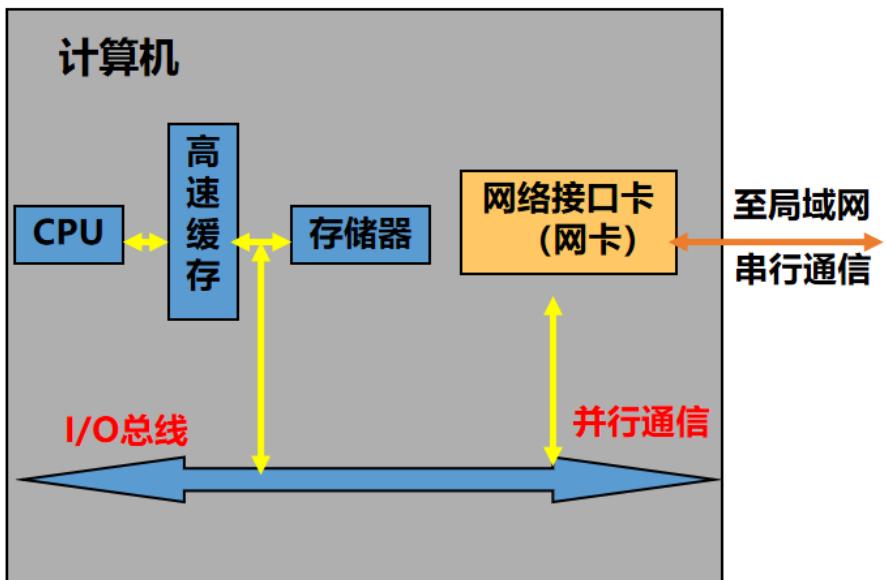
- 网状结构



- 网状网络的每一个站点都与其它站点一一直接互连
- **优点：**连接方法主要是利用冗余的连接，实现站与站之间的高速传输和高容错性能，以提高网络的速度和可靠性
- **缺点：**关系复杂，建网难，维护难

计算机与局域网的连接

- 计算机与局域网通过**网络接口板**进行连接，网络接口板又称**通信适配器**（Adapter）或**网络接口卡NIC**（Network Interface Card），通常我们称为“**网卡**”。

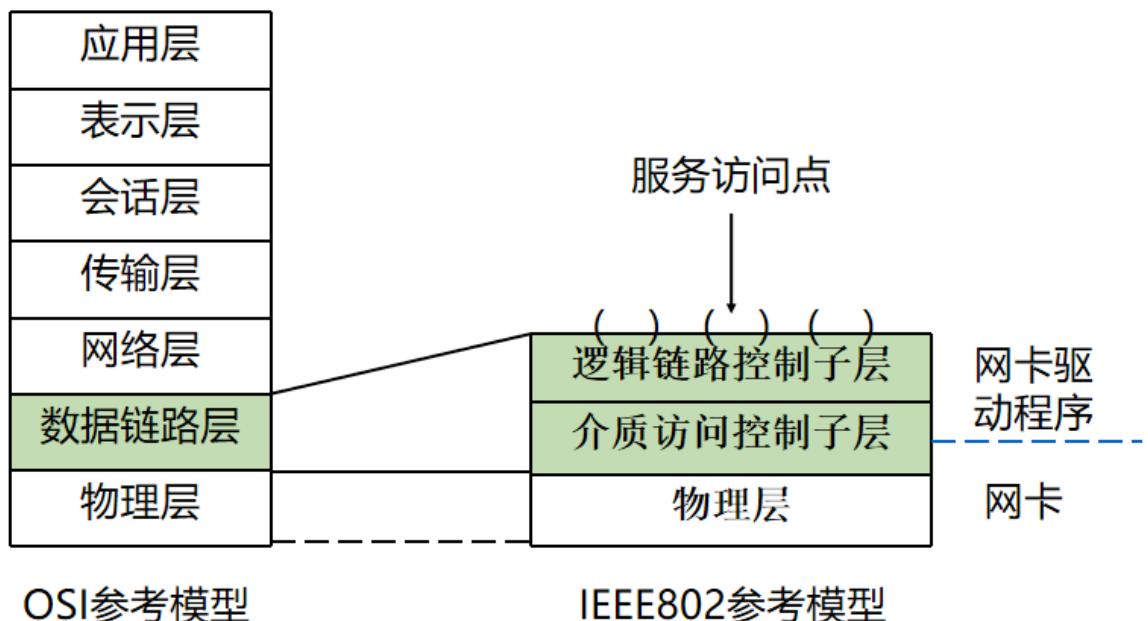


局域网体系结构

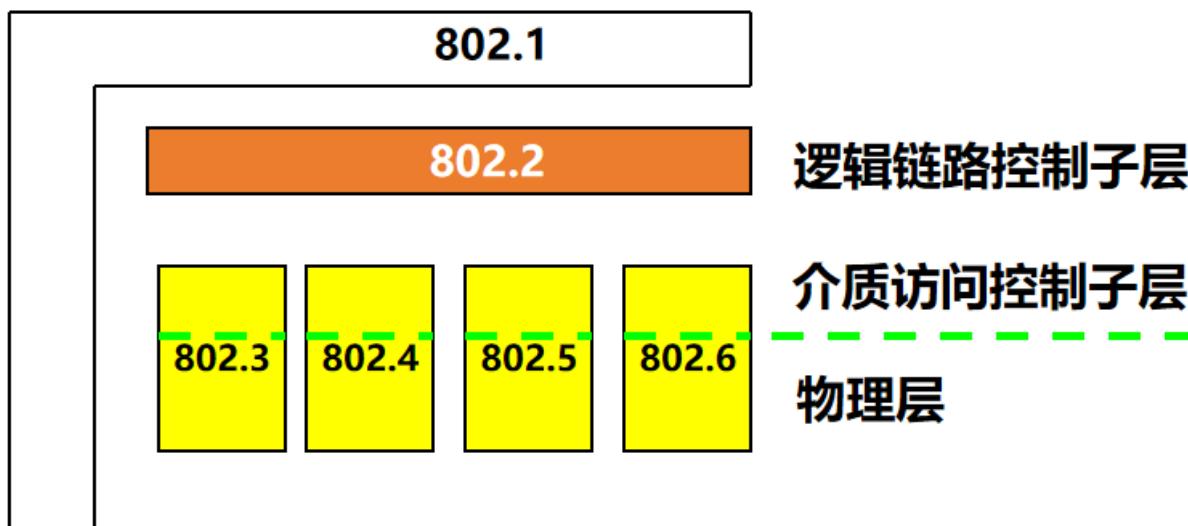
局域网体系结构



IEEE802与OSI参考模型对应关系



几个基础性IEEE802标准的关系



链路层寻址和ARP

每个节点有网络层地址和链路层地址。

- 网络层地址: 节点在网络中分配的一个唯一地址 (IP地址)。用于把分组送到目的IP网络。长度为32比特 (IPv4)。
- 链路层地址: 又叫做MAC地址或物理地址、局域网地址。
 - 用于把数据帧从一个节点传送到另一个节点(同一网络中)。
- MAC地址 (LAN地址、物理地址) :
 - 节点“网卡”本身所带的地址 (唯一)。
 - MAC地址长度通常为6字节(48比特), 共 2^{48} 个。 1A-2F-BB-76-09-AD
 - 6字节地址用16进制表示, 每个字节表示为一对16进制数
 - 网卡的MAC地址是永久的 (生产时固化在其ROM里)

局域网中每个网卡都有唯一的局域网地址

MAC地址分配

- 由专门机构IEEE管理物理地址空间
 - 负责分配六个字节中的**前三个字节**（高24位，**地址块**）
- MAC地址是平面结构**
 - 带有同一网卡的节点，在任何网络中都有同样的MAC地址。**
- IP地址具有层次结构**
 - 当节点移动到不同网络时，节点的IP地址发生改变。

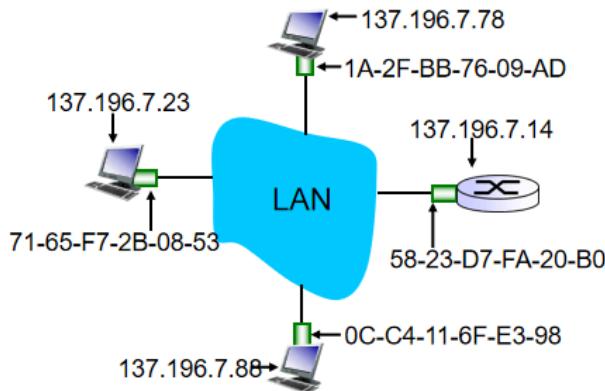


MAC地址识别

- 广播信道的局域网中，一个节点发送的帧，在信道上广播传输，其他节点都可能收到该帧。
- 大多数情况，一个节点只向某个特定的节点发送。
- 由“网卡”负责MAC地址的封装和识别。
- 发送适配器：**将目的MAC地址封装到帧中，并发送。**所有其他适配器都会收到这个帧。**
- 接收适配器：**检查帧的目的MAC地址是否与自己MAC地址相匹配：
 - 匹配：**接收该帧，取出数据报，并传递给上层。
 - 不匹配：**丢弃该帧。
- 广播帧：**发送给所有节点的帧 **全1地址：FF-FF-FF-FF-FF-FF**

节点的3种不同地址表示

- **应用层的主机名、网络层的IP地址和链路层的MAC地址**
- 实际在链路上传输时，根据MAC地址，确定相应的节点



地址之间如何进行转换：

主机名 → IP地址 → MAC地址

◆ **DNS域名系统**: 将**主机名**解析到**IP地址**。

DNS为在因特网中任何地方的主机解析主机名。

◆ **ARP地址解析协议**: 将**IP地址**解析到**MAC地址**。

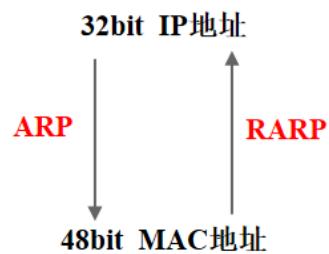
ARP只为在同一个LAN上的节点解析IP地址。

ARP(Address Resolution Protocol): 地址解析协议

问题：如何根据一个主机的IP地址，查找其MAC地址

ARP表: 局域网上的每个节点(主机、路由器)都有这个表：

- 为某些局域网节点进行 IP <-> MAC地址映射：
< IP address; MAC address; TTL >
- TTL (存活时间): 地址映射将被删除的时间 (通常为20分钟)

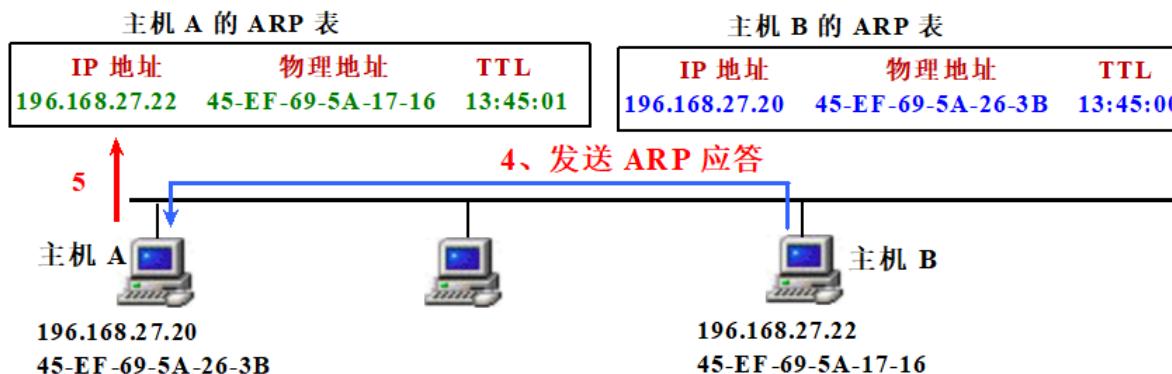


ARP和RARP在TCP/IP协议族中的位置

ARP解析实例

位于同一子网内的主机地址解析。

5、更新 ARP 表

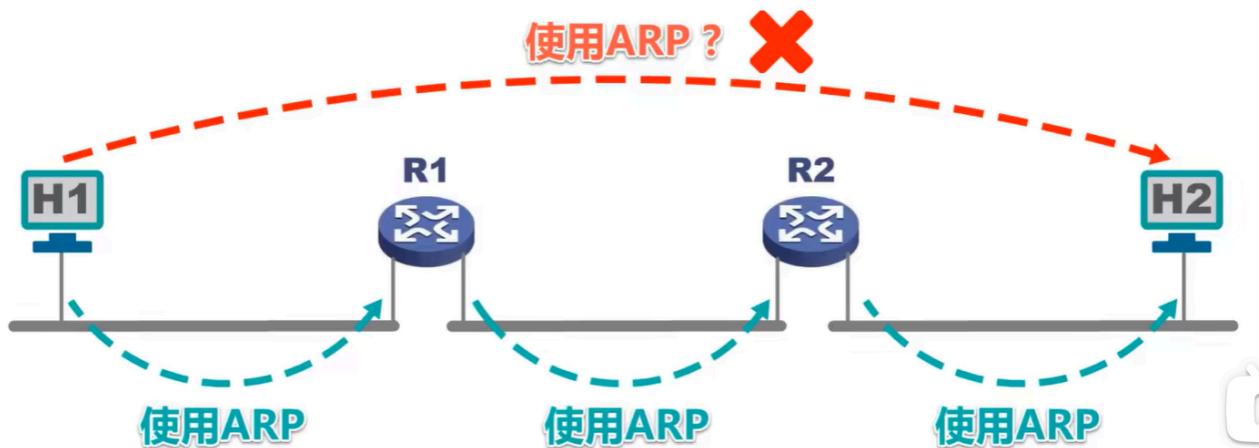


位于同一子网的主机进行通信时的 ARP 应答处理

ARP: 两个主机位于同一个局域网

- 主机A希望发送数据报给主机B
 - B的MAC地址不在A的ARP映射表中
- 主机A 广播 ARP查询分组, 其中包含**B的 IP地址**
 - 目的MAC地址 = FF-FF-FF-FF-FF-FF
 - 局域网中所有节点收到ARP查询分组
- 主机B收到ARP查询分组, 返回B的MAC地址给主机A
 - 包含有B的MAC地址的帧发送给主机A(单播)
- 主机A在它的ARP表中缓存 **IP-to-MAC 地址对**, 直到信息
 - 软状态: 信息超时会被删除, 除非有新的更新消息
- ARP是即插即用的:
 - **节点创建ARP表不需要网络管理员的干预**

3.7 MAC地址、IP地址以及ARP协议

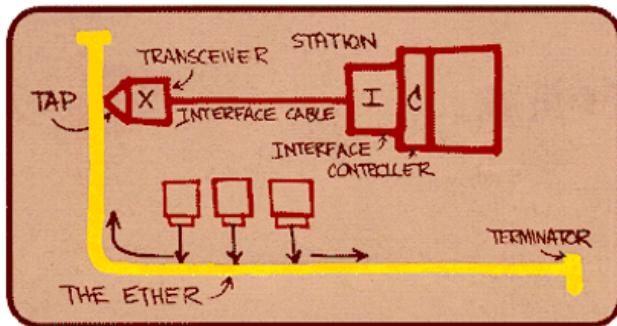


以太网(Ethernet)

到目前为止，以太网是最为著名的有线局域网技术

以太网成功的原因：

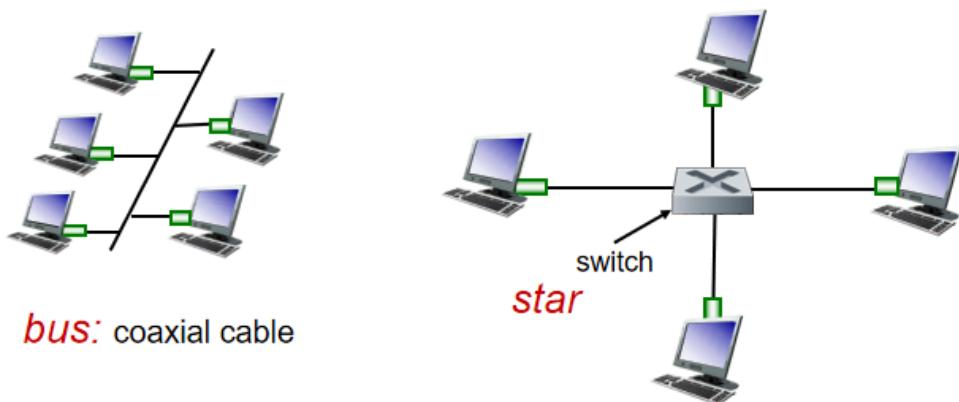
- 是第一个广泛使用的局域网技术；
- 简单、便宜；
- 版本不断更新，数据速率更高、成本更低。



Metcalfe's Ethernet sketch

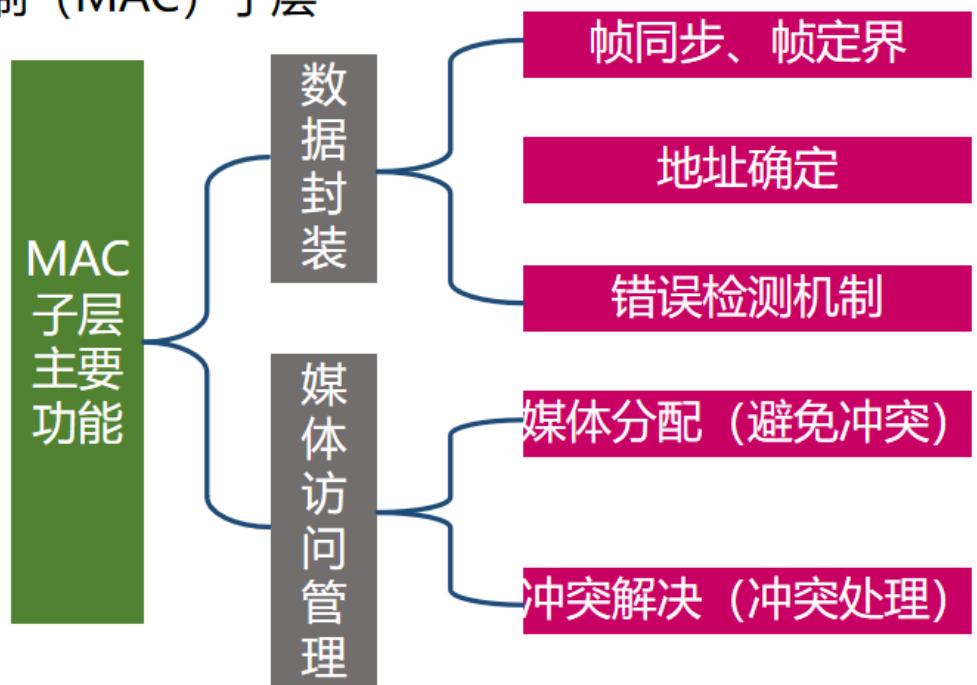
以太网的物理拓扑结构

- 总线(bus): 一直流行到90年代中期
 - 所有节点都属于相同的冲突域
- 星形(star): 目前流行
 - 中心是交换机
 - 每个端口运行一个独立的以太网协议(节点相互之间不发生碰撞)



以太网链路层控制技术

- 媒体访问控制 (MAC) 子层



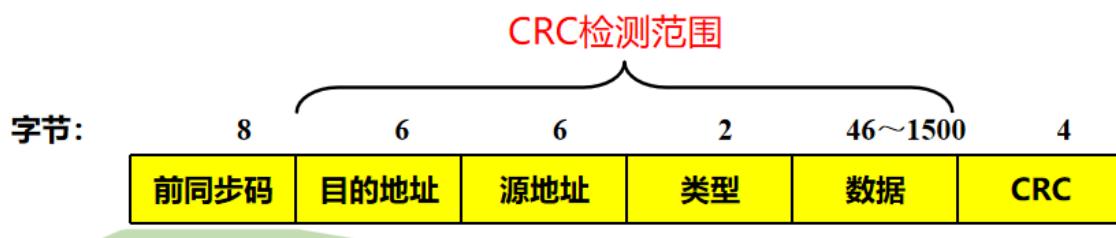
以太网的帧结构

发送方适配器将IP数据报（或其他网络层协议分组）封装成以太网帧，并传递到物理层。

- ✓ 常用的以太网 MAC 帧格式有两种标准：

- DIX Ethernet V2 标准
 - IEEE 的 802.3 标准

- ✓ 最常用的 MAC 帧是以太网 V2 的格式。



- **发送方**: 发送适配器将IP数据报封装成以太网帧，并传递到物理层。
 - **接收方**: 接收适配器从物理层收到该帧，取出IP数据报，并传递给网络层。

前同步码(8字节)

- 前7字节是“10101010”，最后一个字节是“10101011”。
- 使接收方和发送方的**时钟同步**，接收方一旦收到连续的8字节前同步码，可确定有帧传过来。
- 前同步码是“无效信号”，接收方收到后删除，不向上层传送。
- CRC的校验范围不包括前同步码。

前同步码	目的地址	源地址	类型	数据	CRC
------	------	-----	----	----	-----

源、目的MAC地址(各6字节)

- 例，同一以太网LAN中两台主机通信。
- 主机A向主机B发送一个IP数据报。
- ✓ 主机A适配器的MAC地址：XX-XX-XX-XX-XX-XX
- ✓ 主机B适配器的MAC地址：YY-YY-YY-YY-YY-YY
- 适配器B只接收目的地址与其MAC地址匹配或广播地址的帧，并将数据字段的内容传递给网络层。否则，丢弃该帧。

前同步码	目的地址	源地址	类型	数据	CRC
------	------	-----	----	----	-----

类型字段(2字节)

以太网可以“多路复用”（支持）多种网络层协议。通过“类型”字段区分。

- 发送方填入网络层协议“类型”编号；
- 接收适配器根据“类型”字段，将数据字段传递给相应的网络层协议。

前同步码	目的地址	源地址	类型	数据	CRC
------	------	-----	----	----	-----

数据字段(46~1500字节)



携带网络层传来的IP数据报

- 以太网的最大传输单元MTU是1500字节：
 - 若IP数据报超过1500字节，必须将该数据报分段。
- 最小长度是46字节：
 - 如果IP数据报小于46字节，必须填充为46字节。接收方网络层去除填充内容。

前同步码	目的地址	源地址	类型	数据	CRC
------	------	-----	----	----	-----

循环冗余检测CRC(4字节)

检测数据帧中是否出现比特差错（翻转）。

- ✓ **发送主机计算CRC：**范围包括目的地址、源地址、类型、数据字段的比特，结果放入帧CRC字段。
- ✓ **接收主机进行CRC校验：**接收主机对收到的帧进行同样计算，并校验结果是否和CRC字段的内容相等。若计算结果不等于CRC字段的值(CRC校验失败)，该帧有差错。

前同步码	目的地址	源地址	类型	数据	CRC
------	------	-----	----	----	-----

以太网: 不可靠的无连接服务

以太网: 不可靠的无连接服务



以太网向网络层提供的服务。

- **无连接服务：**通信时，发送方适配器不需要先和接收方适配器“握手”。
- **不可靠的服务：**接收到的帧可能包含比特差错。
 - 收到正确帧，不发确认帧；
 - 收到出错帧，丢弃该帧，不发否定帧。
 - 发送适配器不会重发出错帧。
 - 丢弃数据的恢复是通过终端传输层的可靠数据传输机制来实现的
 - **以太网并不知道是传输新数据，还是重传数据。**
- 以太网的MAC协议：使用无时隙的CSMA/CD协议（二进制指数回退）
——见前述6.3.2节

链路层交换机

- 链路层设备

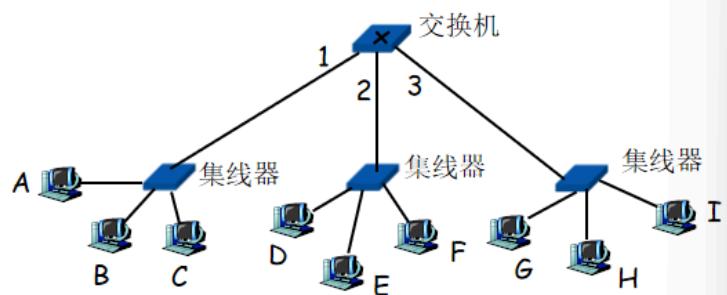
- 存储转发数据帧
- 检查达到的数据帧的MAC地址，有选择的转发数据帧到一个或多个输出链路，当数据帧被转发到一个共享网段时，使用CSMA/CD来访问共享链路

- 透明

- 主机不关心是否存在交换机

- 即插即用和自学习

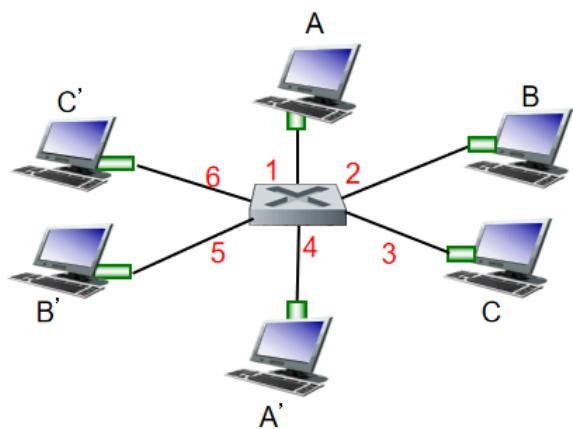
- 交换机不需要手工配置



支持多节点同时传输

交换机：支持多节点同时传输

- 每个主机由单独的链路直接连到交换机端口
- 交换机可以缓存数据帧
- 以太网协议在每个输入链路使用，无碰撞，全双工
 - 每条链路自身是一个碰撞域
- 交换机：A-to-A' 和B-to-B' 可以同时传输，而不会发生碰撞



switch with six interfaces
(1,2,3,4,5,6)

交换机转发表

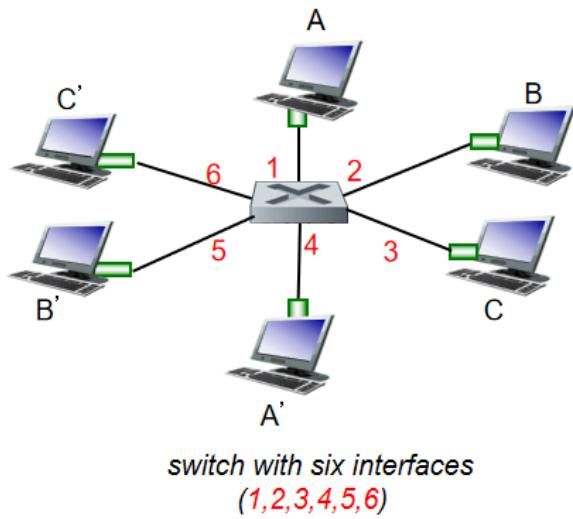
问题: 交换机是怎么知道 A' 可通过端口 4 达到, B' 可通过端口 5 到达呢?

回答: 每个交换机有一个交换机转发表, 其中每个条目:

(主机的MAC地址, 到达主机的端口, 时戳)
类似于路由表

问题: 转发表中的条目是怎么建立的呢?
是否类似于路由协议呢?

回答: 通过**自学习**



自学习

- 交换机会学习通过哪些端口可以到达哪些主机
 - 当收到数据帧时, 交换机“学习”发送主机的位置: 进入的局域网网段(到达端口)
 - 在转发表中记录发送主机/位置对



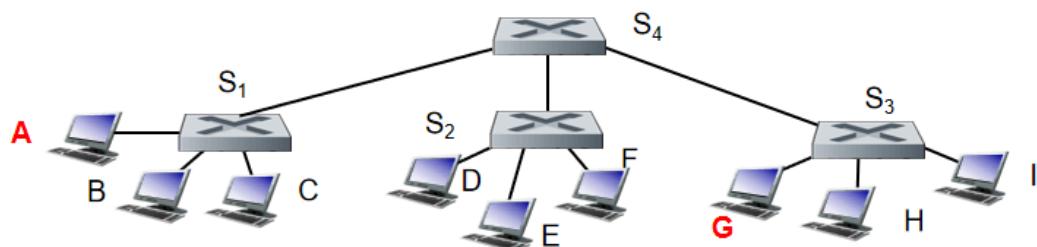
交换机：数据帧的过滤/转发

当交换机收到数据帧：

1. 记录到达链路和发送主机的**MAC地址**
2. 使用数据帧的**目的MAC地址**, 在**转发表中检索**
3. **如果在转发表条目中找到对应的MAC地址**
4. 执行{
 **如果 目的MAC地址对应的端口与数据帧的达到端口相同
 则 丢弃该数据帧
 否则 转发该数据帧到条目指定的端口**
}
5. }
6. **否则, 向除到达端口之外的所有端口转发(flood)**

交换机互连

- 交换机可以互连在一起

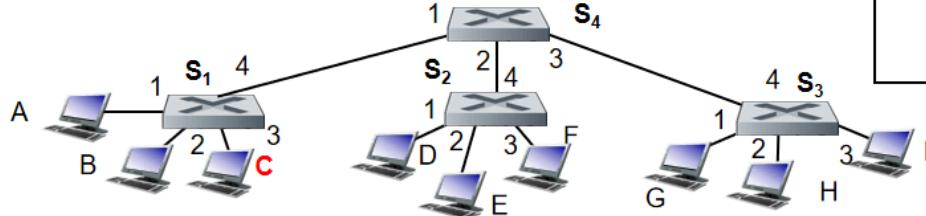


问题：A发送数据帧给G——S1是怎么知道要把数据帧先转发到S4和S3的？

回答：泛洪和自学习

多个交换机自学习的例子

假设主机 C 发送数据帧到主机 I，主机 I 响应给主机 C



MAC addr	interface	TTL
C	1	60
I	3	60

问题: 请大家画出 S_1, S_2, S_3, S_4 交换机转发表和分组转发

S_1

MAC addr	interface	TTL
C	3	60
I	4	60

S_2

MAC addr	interface	TTL
C	4	60

S_3

MAC addr	interface	TTL
C	4	60
I	3	60

- 在大多数以太网交换机中，MAC地址表项的默认老化时间为60秒
- 也就是说：
 - 如果 60 秒内没有再次收到来自该 MAC 地址的帧，该 MAC 地址就会从转发表中被移除（过期）

交换机的交换特点

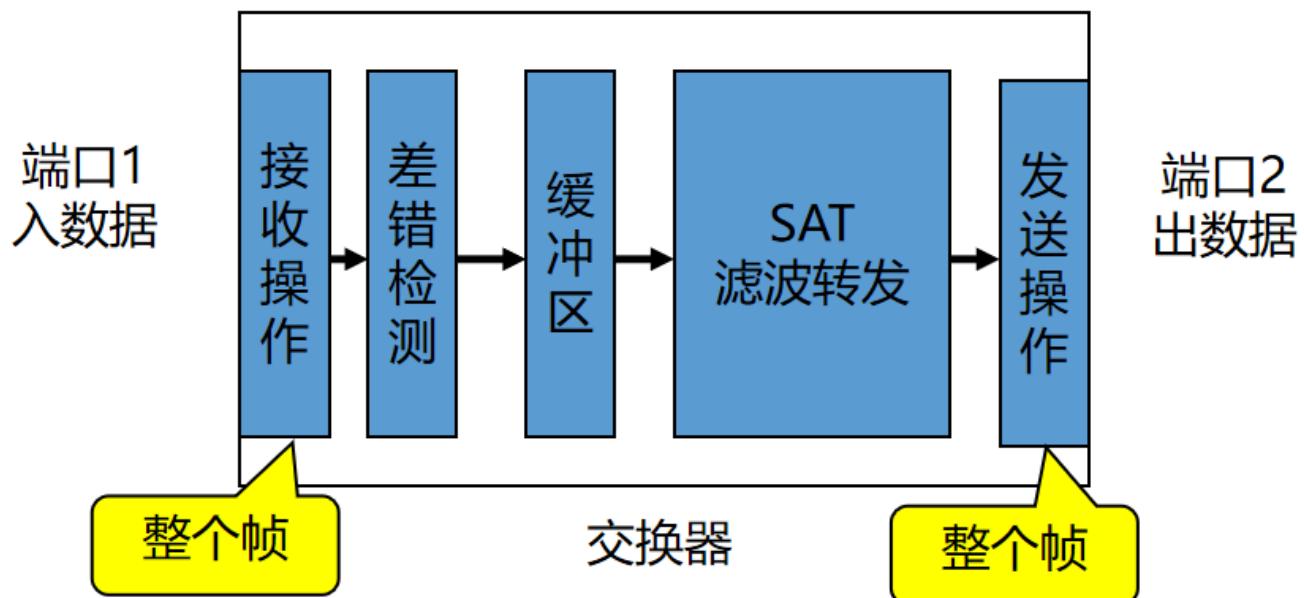
- 识别目的MAC地址，根据交换表进行端口选择
- 识别源MAC地址更新交换表

在识别目的MAC地址和源MAC地址的过程中
是否需要接收并缓存完整的帧呢？

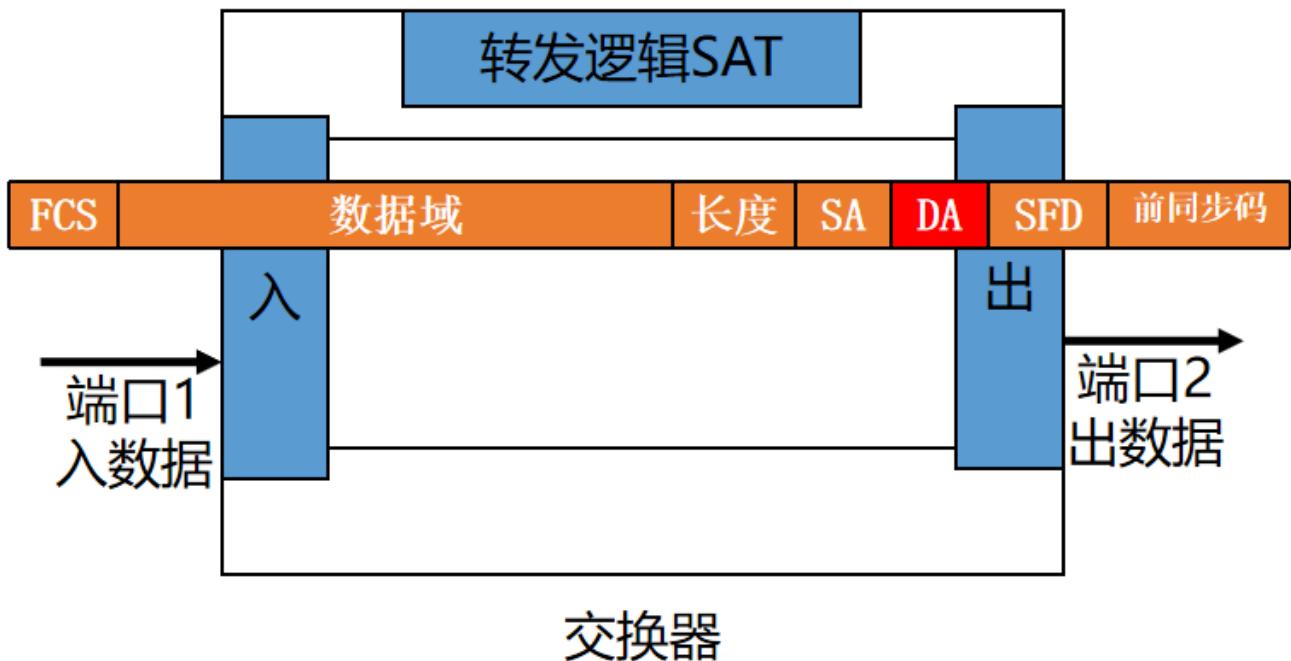
● 交换机的交换方式：

- 存储转发（缓存整个帧后再转发）
- 快速分组又称直通交换（识别出目的地址直接转发）

交换方式1.存储转发



快速分组交换方式



◆ 2) 直通交换 (Cut-through)

✿ 工作原理:

- 交换机只要接收到帧的前**14个字节** (目的MAC+源MAC+类型字段) 后，就可以立即开始转发
- **不等待完整帧，不做 CRC 检查**

✓ 特点:

优点

延迟小，速度快

缺点

容易转发损坏帧 (不做错误检测)

交换机的交换方式讨论

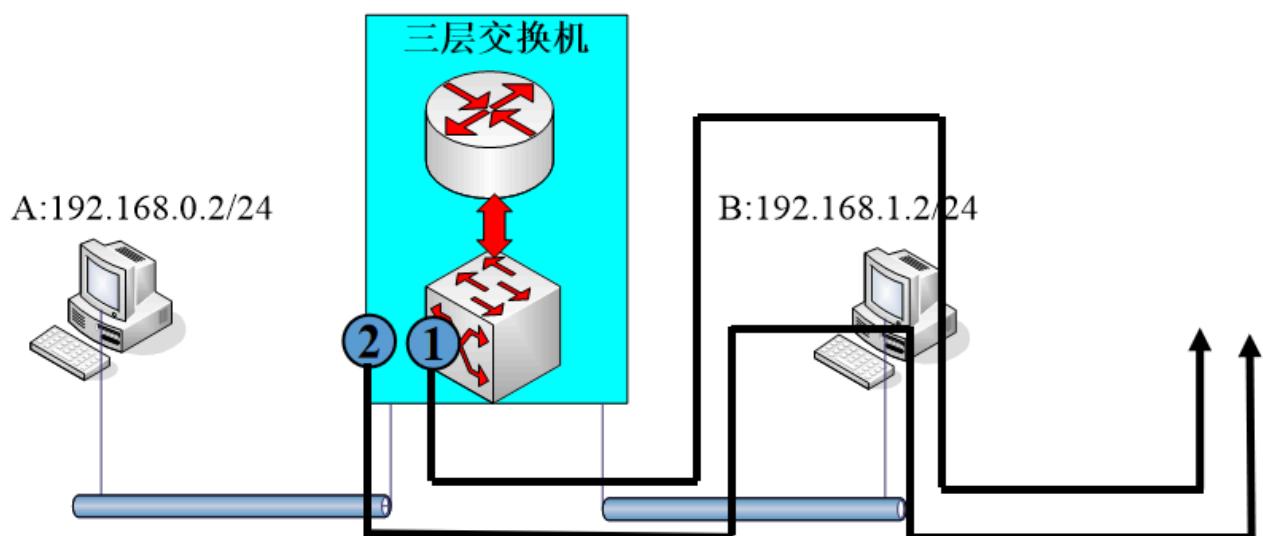
- **存储转发:** 具有差错检测功能，转发时延较大，适用于出错率高的链路。
- **快速分组又称直通交换:** 不具有差错检测功能，转发时延较小，适用于时延要求高，出错率低的链路。

三层交换机

- 三层交换是相对于传统的交换概念而提出的
- 传统的交换技术是在OSI网络参考模型中的第二层（即数据链路层）进行操作的，通常称做“二层交换机”。
- 三层交换技术能够在网络模型中的第三层实现数据包的高速转发。
- 简单地说，三层交换技术就是二层交换技术+**三层转发技术**，三层交换机就是“**二层交换机+基于硬件的路由器**”

交换机制

三层交换机的交换机制



- 一次路由，多次交换

① 第一次“路由”：跨网段通信（IP 决策）

- 主机 A 想给 B 发送数据 → 发现 B 不在本地子网内 → 发给网关（即三层交换机的接口）
- 三层交换机进行一次三层转发（IP 路由）：
 - 查路由表，发现目标 IP 属于 192.168.1.0/24 网段 → 应从端口 2 发出
 - 查找目标 MAC 地址（可能触发 ARP）

这一步就是所谓的：**一次路由**

② 后续“交换”：数据转发优化（MAC 决策）

- 一旦第一次 IP 路由成功
- 三层交换机会在其硬件层缓存转发路径（建立类似 MAC 转发项）
- 后续同样的流量就可以直接使用**“快速二层交换”**完成，不再查 IP 层路由

这就是图中所说的：**多次交换**

三层交换机的工作原理

- 发送站点A在开始发送时，把自己的IP地址与B站的IP地址比较，判断B站是否与自己在同一子网内。
- 若目的站B与发送站A在同一子网内，则进行**二层的转发**。
- 若两个站点不在同一子网内，则发送站A要向“**缺省网关**”发出ARP请求，请求获得B的MAC地址。
- 如果三层交换机知道B的MAC地址，则向A回复B的MAC地址。否则三层交换机根据路由信息向B站广播一个ARP请求，B站得到此ARP请求后向三层交换机回复其MAC地址，三层交换机将B站的MAC地址保存到二层交换引擎的MAC地址表中，并回复给发送站A。
- A直接用B的MAC地址封装数据帧，三层交换机接收到数据后直接进行**二层交换**。

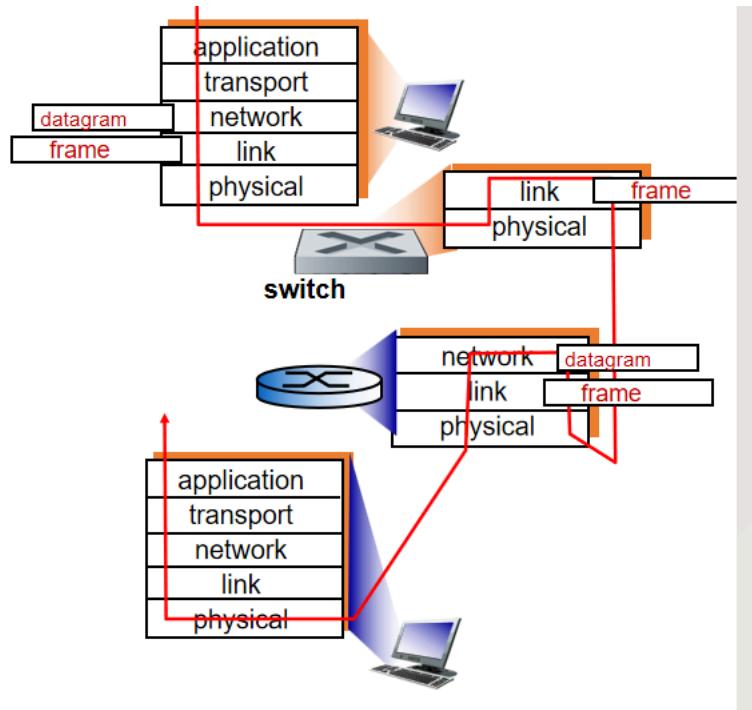
交换机 vs. 路由器

两者都是存储转发设备：

- 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)

两者都有转发表：

- 路由器：使用路由算法计算转发表，基于IP地址转发
- 交换机：通过泛洪、自学习来学习转发表，基于MAC地址转发



交换机的优缺点



交换机的优缺点

- ✓ 即插即用：不需网络管理员干预；
- ✓ 较高的分组过滤和转发率：二层数据处理；
- ✓ 拓扑结构限制为一棵生成树： 防止广播帧循环。

- ✗ 较大的ARP表产生可观的ARP流量和处理量；
- ✗ 可能产生“广播风暴”：

某台主机失控，并传输无穷的以太网广播帧流，交换机将转发所有这些帧，导致整个以太网的崩溃。

路由器的优缺点

- ✓ 网络寻址是**层次的**；
 - ✓ 若网络中**存在冗余路径**，分组不会在路由器中循环。
 - ✓ **无生成树限制**，使用路由器构建因特网可以采用大量丰富的拓扑结构。
 - ✓ 可以使用源和目的之间的**最佳路径**。
 - ✓ 为第二层的广播风暴提供防火墙保护。
-
- ✓ 非即插即用：路由器及主机都需配置IP地址。
 - ✓ 每个分组的处理时间比交换机长。

小网络（几百台主机）：采用交换机。
大网络（几千台主机）：使用路由器。

	集线器	路由器	交换机
流量隔离	No	Yes	Yes
即插即用	Yes	No	Yes
优化选路	No	Yes	No
直通	Yes	No	Yes
隔离广播	No	Yes	No

VLAN

Virtual Local Area Network (虚拟局域网)

“利用支持 VLAN 的交换机，可以在一个实际的物理局域网内，定义多个虚拟的局域网”

这句话的意思是：

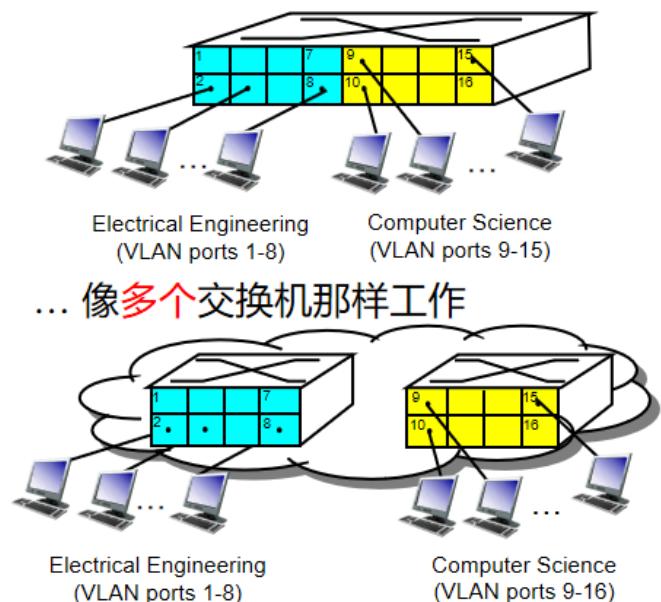
- 即使所有主机都连在同一个交换机上
- 你依然可以通过配置 VLAN，让它们**“看不到彼此”，像在不同交换机上一样**

VLANs

Virtual Local Area Network (VLAN)

利用支持VLAN的交换机，可以在一个实际的物理局域网内，定义多个虚拟的局域网

基于端口的VLAN: 利用交换机内置的管理软件，将端口分组，使得一个单独的交换机



◆ 右上角图解 (基于端口划分 VLAN)

说明了一种最常用的 VLAN 实现方式：

基于端口的 VLAN (Port-based VLAN)

- 通过在交换机的配置中指定：
 - 端口1~8 属于 VLAN 10 (电气工程)
 - 端口9~16 属于 VLAN 20 (计算机科学)

交换机会把这两个端口集合**分别视为两个广播域**

◆ 右下角图示 (逻辑效果)

图中画了两个逻辑交换机：

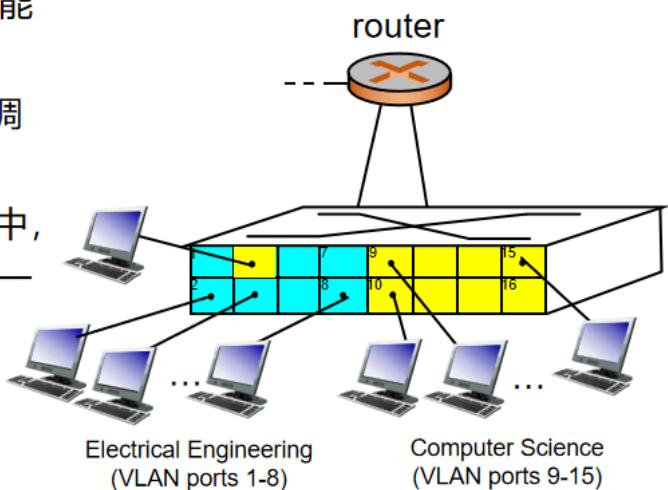
虽然所有主机**物理连接的是同一台交换机**，但因为 VLAN 隔离的存在，
通信行为就像连接在**两台独立交换机上**一样互不干扰

基于端口的 VLAN

● **流量隔离**: 从1-8号端口进/出的帧, 只能访问1-8号端口

● **动态成员**: 端口可以在VLAN之间动态调整

● **VLAN间转发**: 通过路由完成 (在实际中, 厂商会将路由功能和交换功能都整合在一台设备中)



2. 动态成员 (图中红点②)

“端口可以在 VLAN 之间动态调整”

这意味着管理员可以:

- 在软件中将某个端口**重新划入另一个 VLAN**
- 不需要拔线、物理改动, 只需在交换机配置中变更即可

好处: 灵活、便于管理, 尤其适用于用户经常迁移办公场所的场景

3. VLAN 间转发 (图中红点③)

“不同 VLAN 的主机通信，需要通过路由器进行”

这叫做 **VLAN 间路由 (Inter-VLAN Routing)**

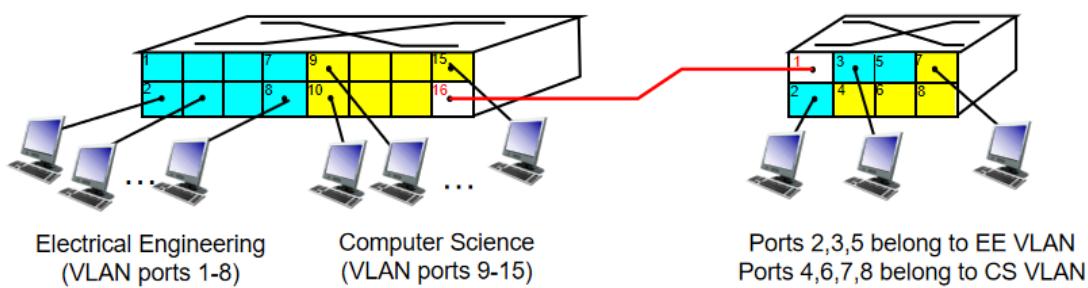
常见方式有两种：

方法	原理
外接路由器	图中所示：将多个 VLAN 的子接口连接到路由器，路由器来进行不同 VLAN 之间的通信
三层交换机内部完成	高端交换机支持三层功能， 内置路由器功能 ，可在交换芯片中实现高速转发

重点：交换机只能处理同 VLAN 的通信，跨 VLAN 必须路由器参与。

跨越多个交换机的 VLAN

跨越多个交换机的 VLAN



- **Trunk Port (干线端口)** : 每台交换机上的一个特殊端口被配置为干线端口，干线端口属于所有VLAN，发送到任何VLAN的帧经过干线链路转发到其他交换机。
干线端口承载定义在多个物理交换机之上的VLAN间的流量
- 某一个VLAN内的流量帧，如果要跨域物理的交换机，需使用802.1q格式（带有VLAN ID 信息）
- 802.1q协议的作用：对干线端口之间传输的帧，添加/移除额外的头部字段

链路虚拟化：网络作为链路层

多协议标签交换

Multiprotocol label switching (MPLS)



五、MPLS 转发的流程（简化版）

1. 首跳路由器查 IP 地址 → 打上 MPLS 标签
2. 中间路由器 不查 IP，而是查 Label 转发表 → 交换标签或转发
3. 最后一个 MPLS 路由器 剥掉标签 → 交给普通 IP 层处理

整个过程是“标签驱动转发”，更快、更灵活、更适合运营商骨干网或 VPN。

MPLS capable routers

- 也就是：标签交换路由器，label-switched router
- 仅根据标签值将数据包转发到出接口（不检查 IP 地址）
 - MPLS 转发表与 IP 转发表不同
- **灵活性：**MPLS 转发决策不同于 IP
 - 使用目的地和源地址以不同方式将流路由到同一目的地（流量工程）
 - 如果链接失败，则快速重新路由：预先计算的备份路径（对 VoIP 有用）

✓ 总结归纳：

内容	MPLS 特性	操作
路径选择依据	可按源+目的地址、QoS、业务类型等灵活控制	编辑
转发机制	基于 label，查表速度快，不查 IP	编辑
路径建立方式	RSVP-TE/标签分发协议预先设定路径	编辑
标签表作用	每个路由器根据 in label 查找 → 替换 out label – 下一跳	编辑
优势	快速转发、灵活路径控制、支持流量工程、快速恢复	编辑

第六章：总结

- 链路层功能、提供的服务
- 差错检测方法
- 多路访问链路和协议
- 局域网技术
- 以太网技术(链路层和物理层的实现方式)
 - 帧格式
 - 以太网提供无连接、不可靠的服务
 - 以太网采用的CSMA/CD原理
 - 物理层曼彻斯特编码*
- 集线器和交换机
- VLAN

###