

# 1. Introduction

## What's the Internet

一个世界范围的遍及几十亿计算设备的网络

协议：控制因特网中信息的接收和发送，包括信息格式、顺序以及传输及接收中的行为

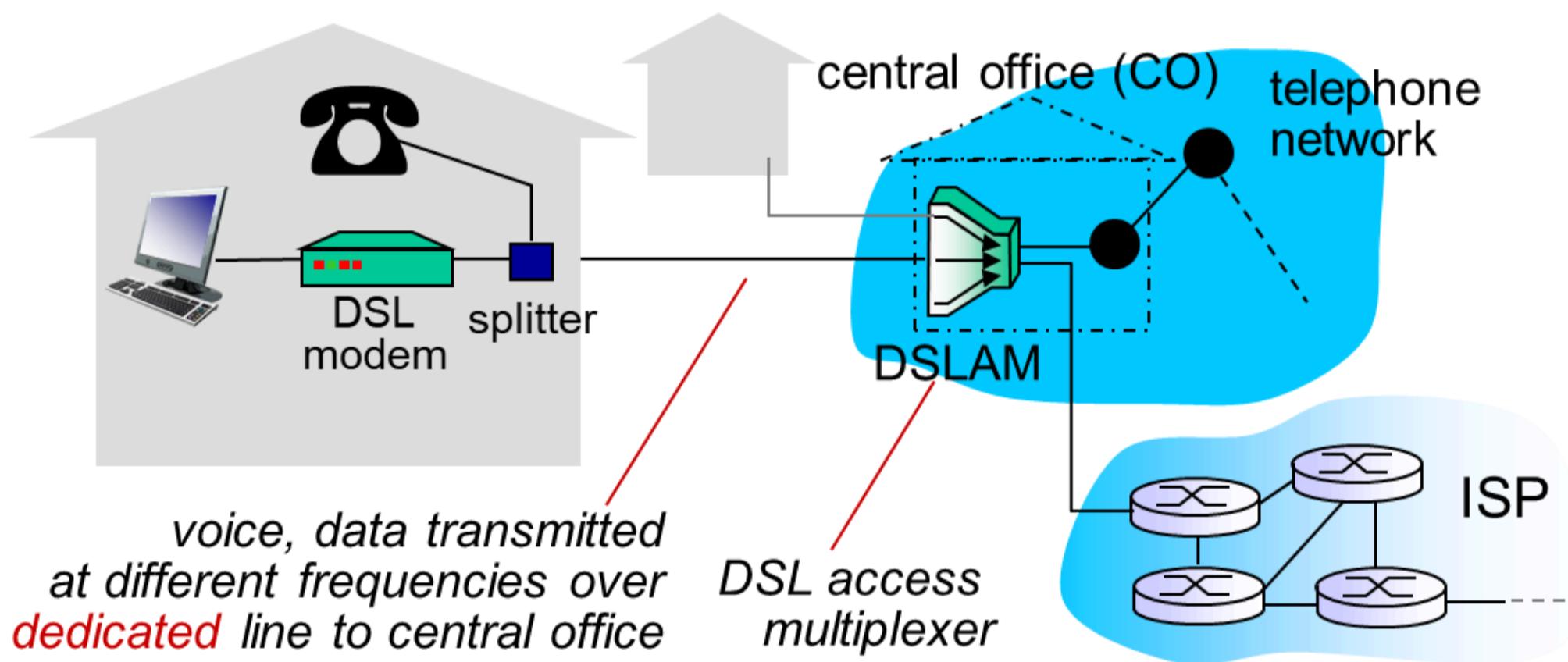
## 网络边缘 Network Edge

边缘：端主机称为主机(host)，又可以被进一步细分为客户端(client)和服务端(server)，具体地可以看为计算机、手机、汽车、卫星等

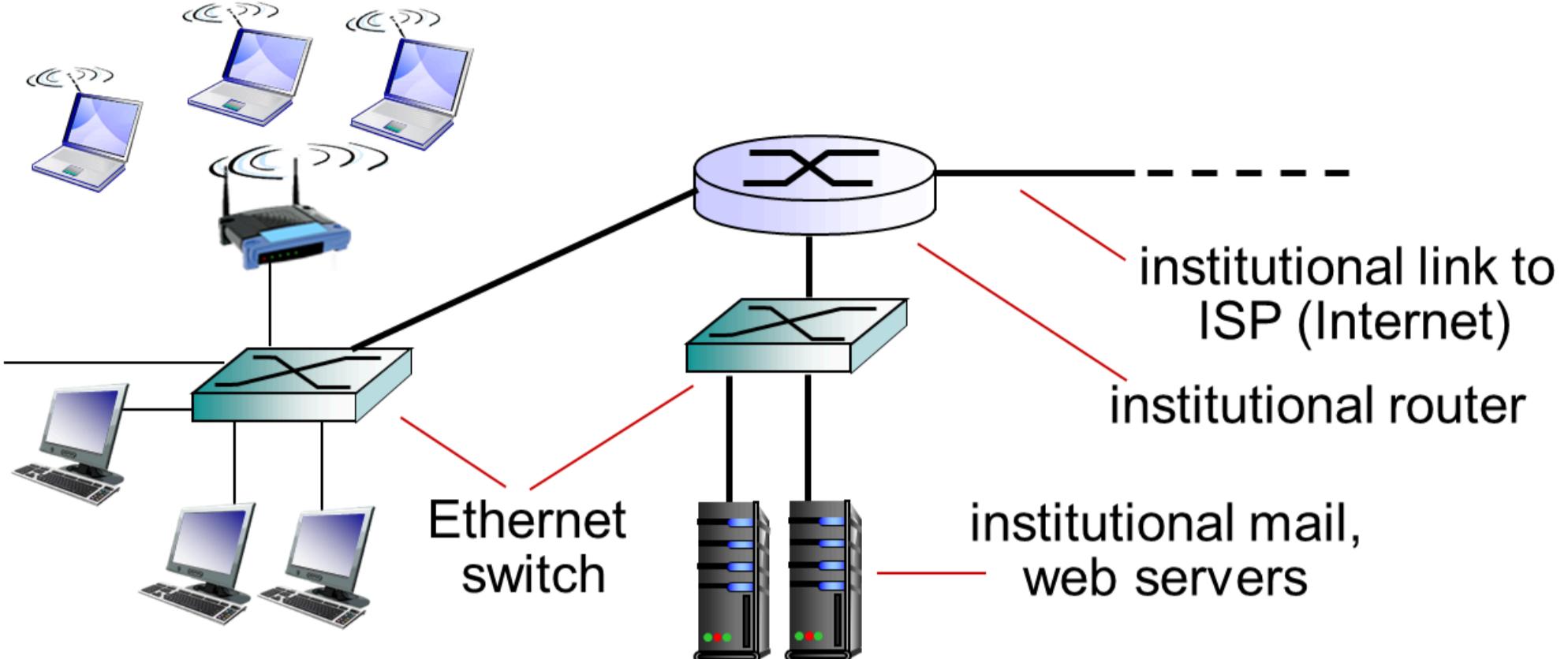
## 接入网络

接入网络(access networks)：指将端系统物理连接到其边缘路由器的网络。包含家庭接入网络、机构接入网络、移动接入网络

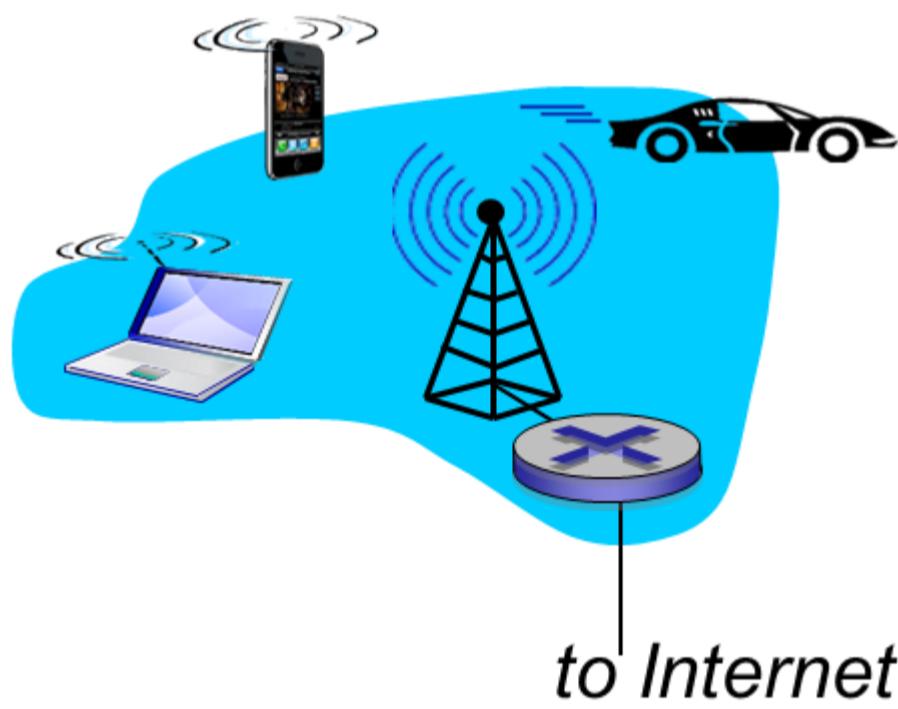
家庭接入通常使用数字用户线 (DSL, Digital Subscriber Line) 及电缆。DSL通过使用现有电话线接入电话公司的数字用户线接入复用器 (DSLAM) ，以高频音方式传输并以数字形式接收。



而在企业和高校等机构中（甚至越来越多家庭环境），往往选择使用局域网（LAN）将端系统连接到边缘路由器。



用户还可使用手机通过运营商的基站进行发送和接收数据分组，称为广域无线接入，如3G, 4G, 5G, LTE等



## 物理媒体

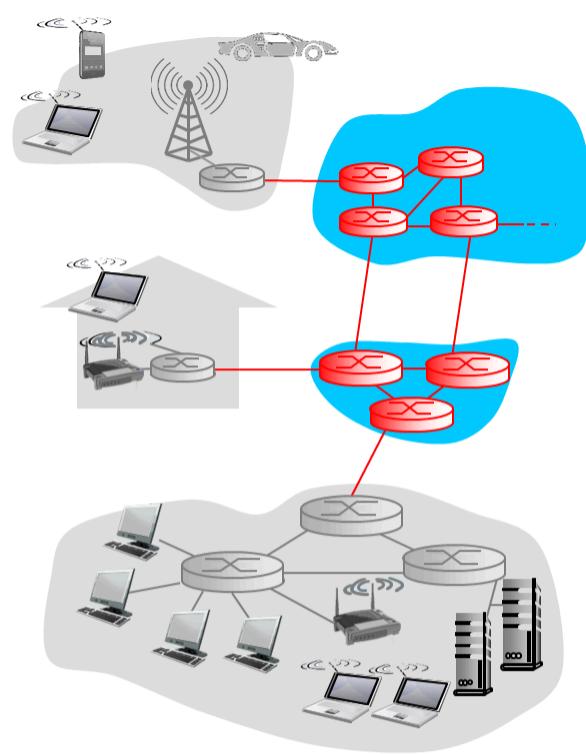
然而网络的连接离不开物理媒体的支持。发射端与接收端直接需要通过跨越物理媒体发送电磁波或光脉冲进行数据交换。

物理媒体分为：

- 导引型媒体(Guided Media): 电波沿着固体媒体前进，如光缆、双绞铜线或 同轴电缆
- 非导引型媒体(Unguided Media): 电波在空气或外层空间中传播，例如在无线局域网或数字 卫星频道中。

## 网络核心 Network Core

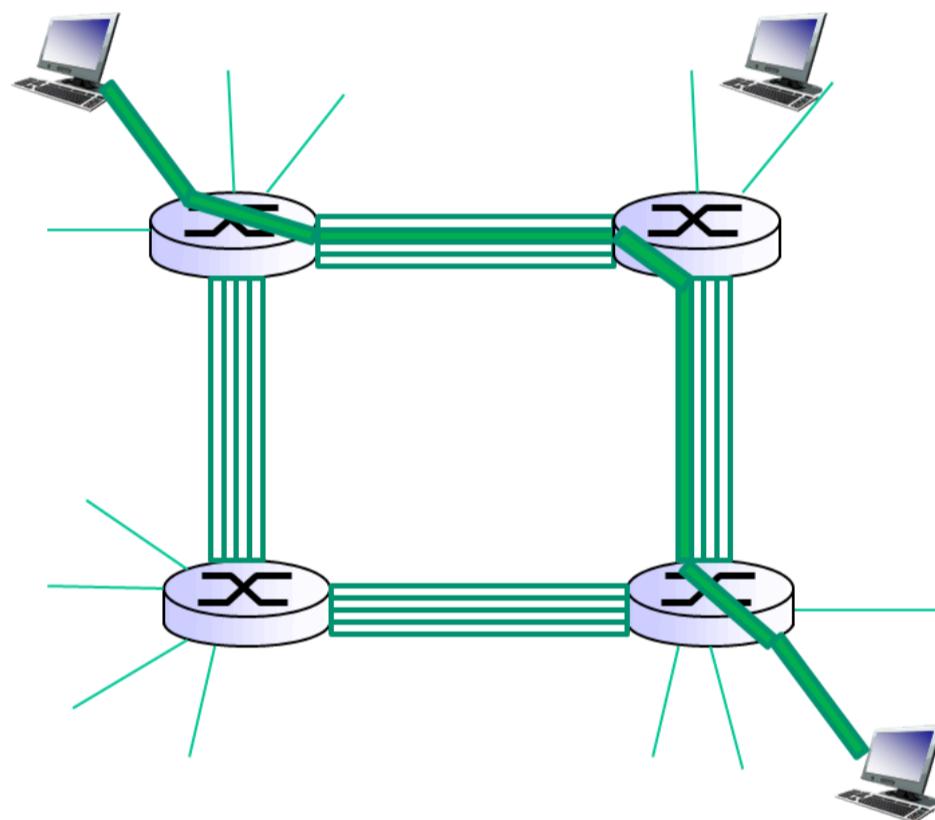
即由互联因特网端系统的 分组交换机和链路构成的网状网络



在互联网中有两种基本的数据传输方式：电路交换(circuit switching)和分组交换(packet switching)

## 电路交换 circuit switching

端与端之间通信所需的资源被预留



链路中的电路通过频分复用 (Frequency-Division Multiplexing, FDM )或时分复用(Time-Division Multiplexing, TDM)实现

对于FDM， 在连接期间链路为每条连接专用一个频段

而对于TDM， 其时域被分割为帧， 帧又被分为固定数量的时隙。在传输中分配时隙， 时隙内所有的频段都被使用。

FDM

Example:

4 users



frequency

time

TDM

frequency

time

## 分组交换 packet switching

源将长报文划分为较小的数据块，称之为分组(packet)在源和目的地之间，每个分组都通过通信链路和分组交换机(packet switch )传送。分组以等于该链路最大传输速率的速度传输通过通信链路。

通过由N条速率均为R的链路组成的路径(所以, 在源和目的地之间有N-1台路由器)，从源到目的地发送一个分组(L比特大)，传输时延为：

$$d_{\text{端到端}} = N \frac{L}{R}$$

排队时延：当到达的分组传输到某条链路，但该链路忙着传输其他分组，分组还需要存储在缓存区内，承担排队时延。

分组丢失：另外，当缓存区已满时，部分分组会被丢弃。

## 网络的网络

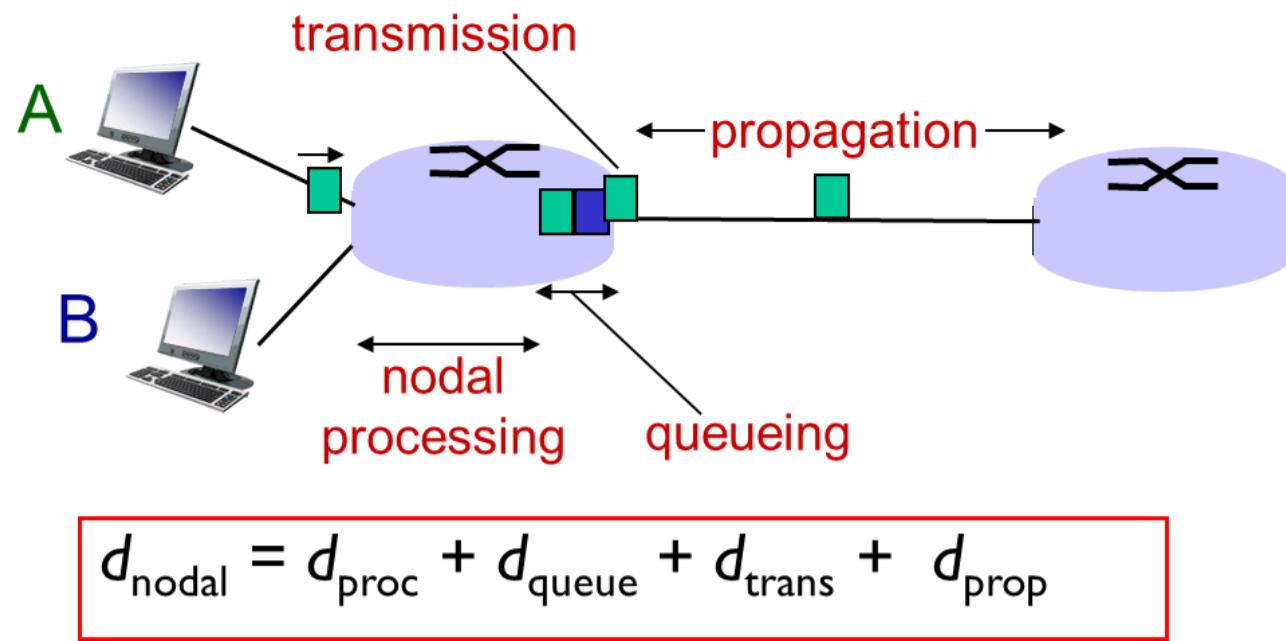
将各个ISP直接相联的开销过大，因此通常进行增设更高级网络进行互联

## 时延、丢包和吞吐量

### 时延 delay

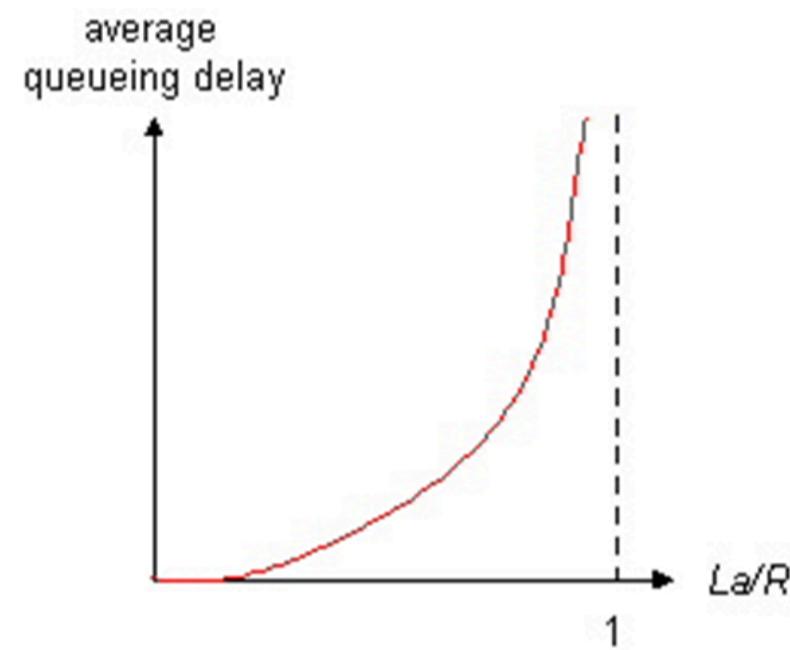
一个分组在沿途的各个节点之间会经受几种时延，为

- 处理时延: 分组到达节点的处理过程耗时
- 排队时延: 分组在链路上等待传输时，它经受排队时延，取决于排队队列长度。
- 传输时延: 整个分组内所有比特到达链路传输的耗时，受制于链路的传输速率。为  $L/R$ ，其中L为分组长度，R为链路传输速率
- 传播时延: 比特在链路上的耗时， $d/s$ 是其中d是路由器A和路由器B之间的距离，s是该链路的传播速率



## 排队时延和丢包

不同分组的排队时延并不相同，通常第一分组和最后一组经受的排队时延截然不同。设 $a$ 为分组到达队列的平均速率，每个分组由 $L$ 比特组成，链路传输速率为 $R$ ，则比率 $La/R$ 被称为流量强度，在估算排队时延时非常有用



当缓存区充满，排队时延过长时，分组将会被丢弃，也就是丢包。

## 吞吐量 throughput

瞬时吞吐量：在某个给定时刻主机接收到文件的速率

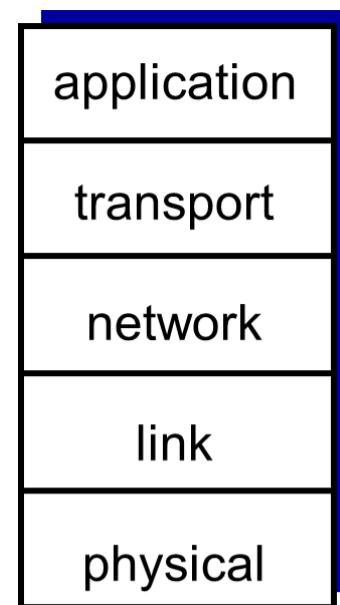
平均吞吐量：一段时间内主机接收到文件的速率

## 协议层次及服务模型

### 协议分层

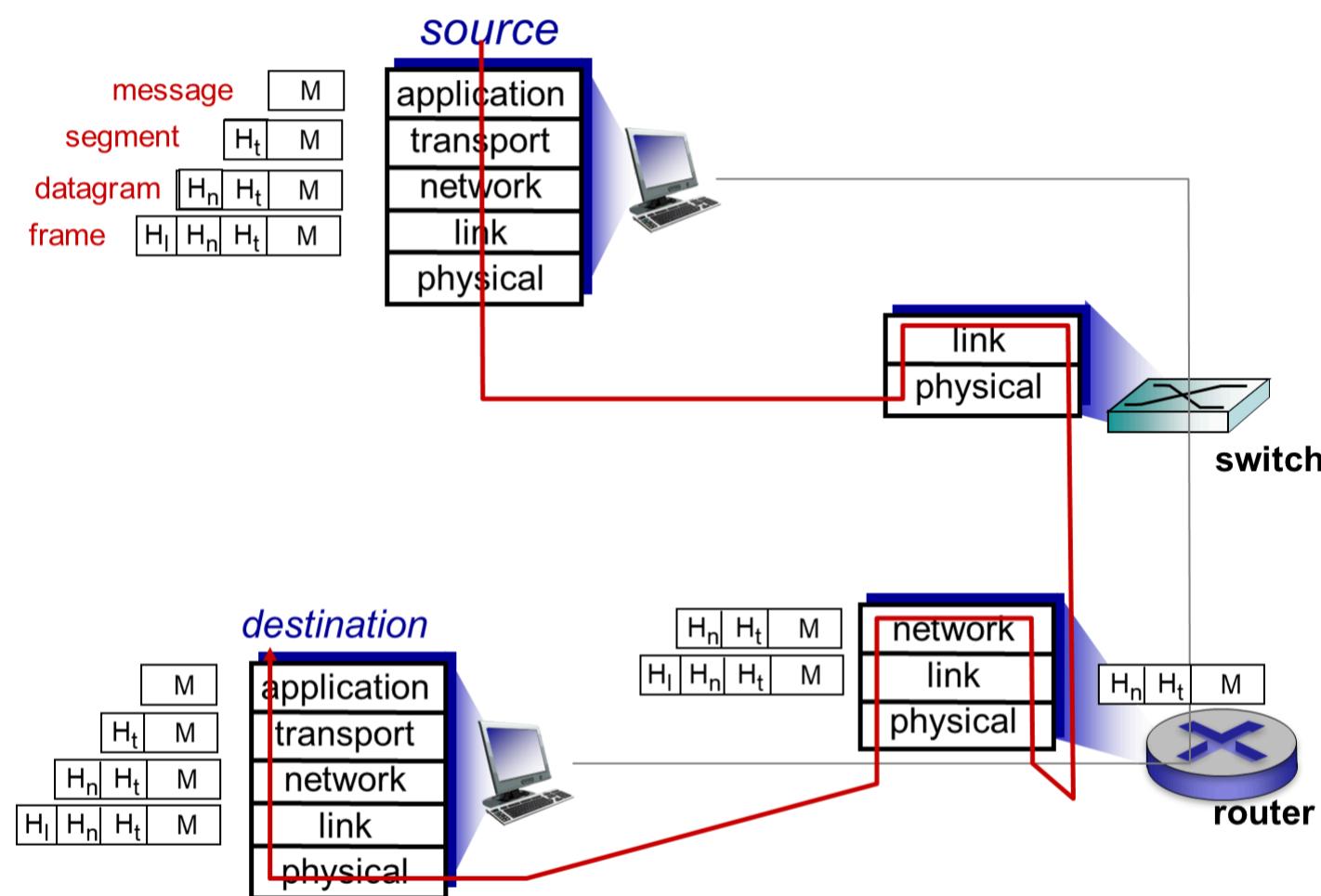
网络设计者以分层（layer）的方式组织协议以及实现这些协议的网络硬件和软件，每一层向上层提供服务，即服务模型

- 应用层是网络应用程序及它们的应用层协议存留的地方，如HTTP,SMTP,FTP.
- 运输层在应用程序端点之间传送应用层报文，TCP,UDP
- 网络层将称为数据报的网络层分组在主机之间传输，IP
- 链路层将分组（帧）从一个节点移动到路径上下一个节点
- 物理层将帧内的一个个比特在节点间传输



## 封装 encapsulation

来自上层的数据在经过当前层时附加上该层信息，以供传输到对应位置后能够正确交付的过程。



例如，运输层收取到报文并附上附加信息供接收端的传输层使用，由此传输层报文段封装了应用层报文段

## 2. Application Layer

### 应用层协议原理

#### 应用程序体系结构 Application Architecture

##### 客户-服务器体系结构 (client-server architecture)

其中有两个身份：

- Server: 始终打开，具有固定IP地址，还可以作为可扩展的数据中心，配备大量主机同时发挥服务器作用
- Client: 与服务器连接（可能间歇性），可能拥有动态IP，客户端间不会直接连接

##### P2P结构 (P2P architecture)

对于服务器几乎没有依赖，应用程序在主机之间直接通信，各个主机既提供服务又享受服务。

此外，P2P结构具有自扩展性 (self-scalability)。每个对等方都由于请求文件产生工作负载，但每个对等方通过向其他对等方分发文件也为系统增加服务能力。

# 进程通信 (Processes communicating)

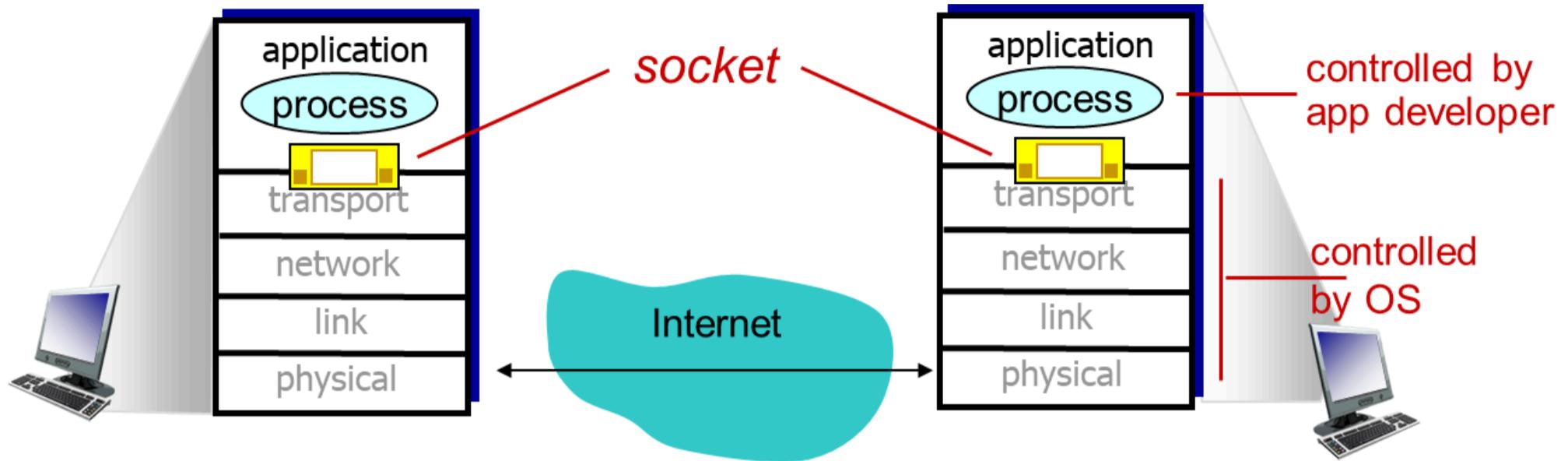
程序实际上在使用进程通信，不同主机的程序使用同号进程进行通信

## 客户与服务进程

通常来说，启动通信的进程称为客户进程，等待消息的是服务进程

## 进程与计算机网络之间的接口 (Socket)

进程之间进行报文发送必须通过网络，具体地，进程通过套接字(socket)的软件接口向网络发送报文和从网络接收报文。



socket可以视为应用程序和网络之间的API

## 进程寻址 Addressing processes

为向特定目的进程发送报文，需要一个地址。该地址包含了目的主机的地址以及指定接收进程的标识符

具体地，即为32位的IP地址和端口号组成

## 可供应用程序使用的运输服务

### 可靠数据传输

运输层可以为应用层提供无差错的可靠数据传输

### 吞吐量

某些应用要求具有最低吞吐量要求，需要运输层满足，这些应用称为带宽敏感的应用 (bandwidth-sensitive application)。而对吞吐量要求没那么绝对的称为弹性应用 (elastic application)

### 定时

运输层协议能提供定时保证

### 安全性

运输协议能提供一种或多种安全服务

## 因特网提供的运输服务

因特网为应用程序提供两个运输层协议，即UDP和TCP。

### TCP服务

TCP服务模型包括面向连接服务和可靠数据传输服务。

- 面向连接的服务：发送报文之前，TCP让客户端和服务器进行握手，即创建连接。

- 可靠的数据传送服务：通信进程能够依靠TCP,无差错、按适当顺序交付所有发送的数据。
- 拥塞控制：当发送方和接收方之间的网络出现拥塞时，TCP的拥塞控制机制会抑制发送进程（客户或服务器）。

## UDP服务

UDP是一种不提供必要服务的轻量级运输协议，它仅提供最小服务，无连接且不保证数据传输。这一过程可能出现丢失或乱序。此外，其没有拥塞控制机制。

## Web和HTTP

### HTTP概况

Web的应用层协议是超文本传输协议 (HyperText Transfer Protocol, HTTP) ,它是Web 的核心。

HTTP协议中存在两个程序：客户程序和服务程序。客户程序和服务器程序运行在不同的端系统中，通过交换 HTTP报文进行会话。客户程序发出请求，服务器程序返回响应。

HTTP使用TCP作为它的支撑运输协议，HTTP客户首先发起一个与服务器的TCP连接。一旦连接建立，该浏览器和服务器进程就可以通过套接字接口访问TCP。

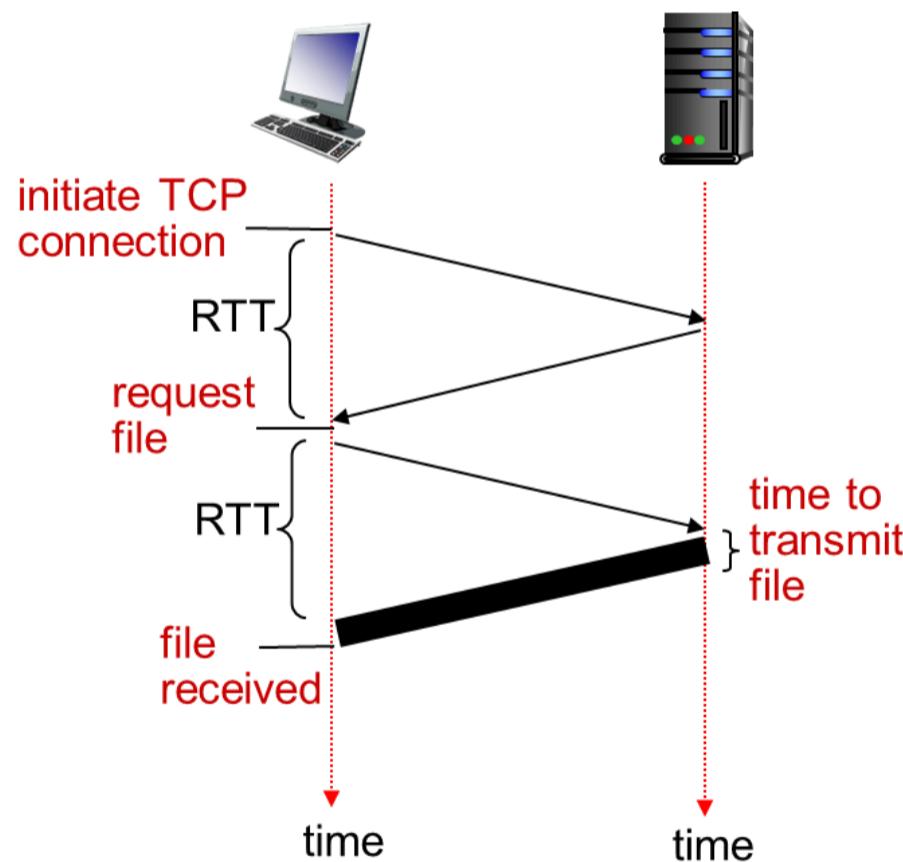
此外，HTTP是一个无状态协议(stateless protocol)，HTTP服务器并不保存关于客户的任何信息

### HTTP连接

#### 非持续连接

每个TCP连接在服务器发送一个对象后关闭，即该连接并不为其他的对象而持续下来。因此需要请求几个对象就产生几个连接

RTT：往返时间指一个短分组从客户到服务器然后再返回客户所花费的时间。



非持续连接有以下缺点：

- 必须为每一个请求的对象建立和维护一个全新的连接。需分配诸多缓存区和保持TCP变量，带来巨大负担
- 每一个对象经受两倍RTT的交付时延, 即一个RTT用于创建TCP,另一个RTT用于请求和接收一个对象

#### 持续连接

服务器在发送响应后保持该TCP连接打开。在相同的客户与服务器之间，后续的请求和响应报文能够通过相同的连接进行传送。

# HTTP报文

## 请求报文

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP请求报文的第一行叫作请求行(request line), 其后继的行叫作首部行(header line)。

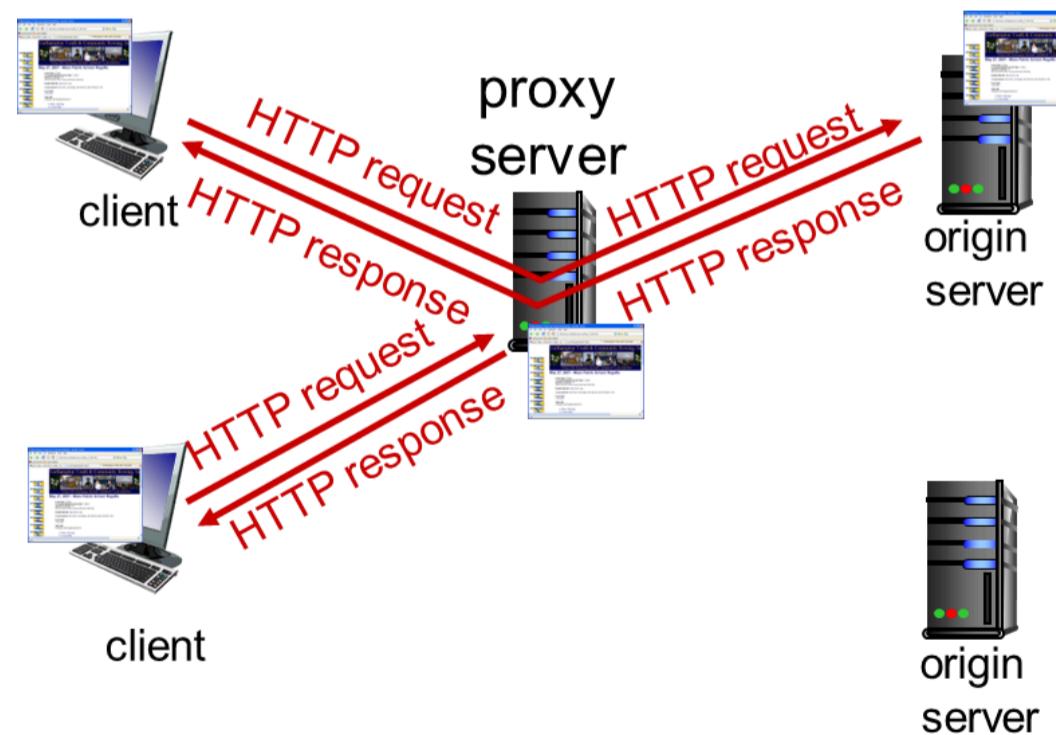
首部行Host: www. someschool. edu指明了对象所在的主机。你也许认为该首部行是不必要的, 因为在该主机中已经有一条TCP连接存在了。但是, 该首部行提供的信息是Web代理高速缓存所要求的。通过包含Connection: close首部行, 该浏览器告诉服务器不要麻烦地使用持续连接, 它要求服务器在发送完被请求的对象后就关闭这条连接。User-agent:首部行用来指明用户代理, 即向服务器发送请求的浏览器的类型。这里浏览器类型是Mozilla/5.0, 即Firefox浏览器。这个首部行是有用的, 因为服务器可以有效地为不同类型的用户代理实际发送相同对象的不同版本。(每个版本都由相同的URL寻址。) 最后, Accept-language: 首部行表示用户想得到该对象的法语版本(如果服务器中有这样的对象的话); 否则, 服务器应当发送它的默认版本。Accept-language:首部行仅是HTTP中可用的众多内容协商首部之一。

## Cookie

尽管HTTP是无状态的, 但对于web网站来说, 其希望能够识别用户。为此, HTTP使用了 cookie作为用户标识。

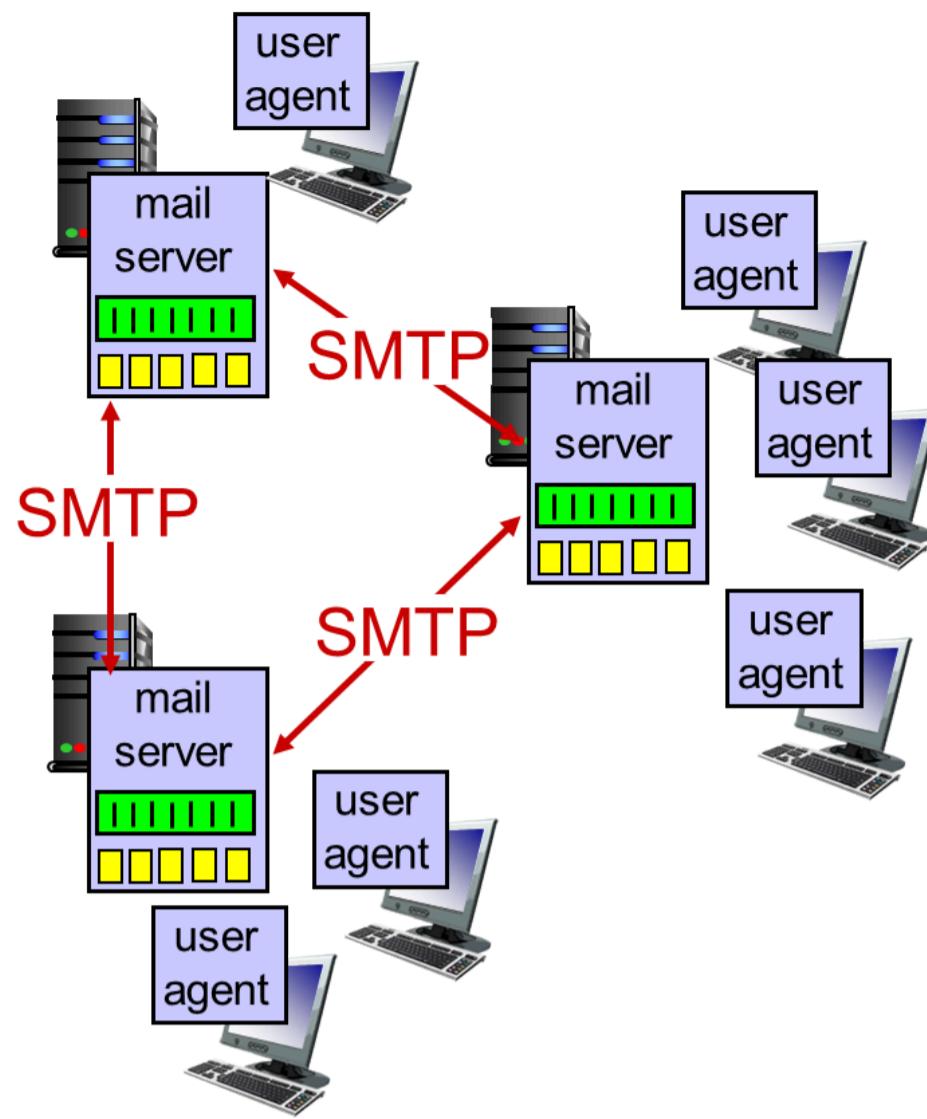
## Web缓存

Web缓存器 (Web cache) 也叫代理服务器 (proxy server), 它是能够代表初始Web服务器来满足HTTP请求的网络实体。一旦某浏览器被配置, 每个对某对象的浏览器请求首先被定向到该 Web缓存器。



## 电子邮件

因特网中电子邮件有3个主要组成部分: 用户代理 (user agent) 、邮件服务器 (mail server) 和简单邮件传输协议 (Simple Mail Transfer Protocol, SMTP)



一个典型的邮件发送过程是：从发送方的用户代理开始，传输到发送方的邮件服务器，再传输到接收方的邮件服务器，然后在这里被分发到接收方的邮箱中。当Bob要在他的邮箱中读取该报文时，包含他邮箱的邮件服务器（使用用户名和口令）来鉴别Bob。Alice的邮箱也必须能处理Bob的邮件服务器的故障。如果Alice的服务器不能将邮件交付给Bob的服务器，Alice的邮件服务器在一个报文队列（message queue）中保持该报文并在以后尝试再次发送。通常每30分钟左右进行一次尝试；如果几天后仍不能成功，服务器就删除该报文并以电子邮件的形式通知发送方（Alice）。

SMTP使用TCP进行可靠数据传输。每台邮件服务器上既运行SMTP的客户端也运行SMTP的服务器端。

## SMTP

SMTP是因特网电子邮件的核心。

SMTP一般不使用中间邮件服务器发送邮件，即使这两个邮件服务器位于地球的两端也是这样。

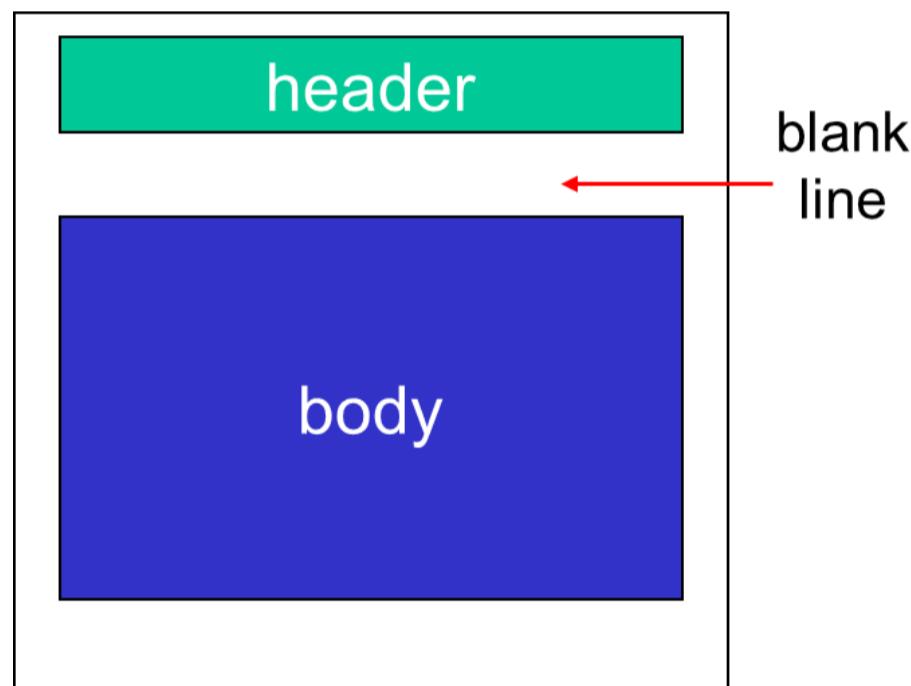
首先，客户SMTP（运行在发送邮件服务器主机上）在25号端口建立一个到服务器SMTP（运行在接收邮件服务器主机上）的TCP连接。如果服务器没有开机，客户会在稍后继续尝试连接。一旦连接建立，服务器和客户执行某些应用层的握手，就像人们在互相交流前先进行自我介绍一样。SMTP的客户和服务器在传输信息前先相互介绍。在SMTP握手的阶段，SMTP客户指示发送方的邮件地址（产生报文的那个人）和接收方的邮件地址。一旦该SMTP客户和服务器彼此介绍之后，客户发送该报文。SMTP能依赖TCP提供的可靠数据传输无差错地将邮件投递到接收服务器。该客户如果有另外的报文要发送到该服务器，就在该相同的TCP连接上重复这种处理；否则，它指示TCP关闭连接。

```

S: 220 hamburger.edu (server's host name)
C: HELO crepes.fr      (client's host name)
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr> (beginning of message)
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?          (two-line message)
C: How about pickles?
C: .                           (end of message)
S: 250 Message accepted for delivery
C: QUIT             (all messages have been sent)
S: 221 hamburger.edu closing connection

```

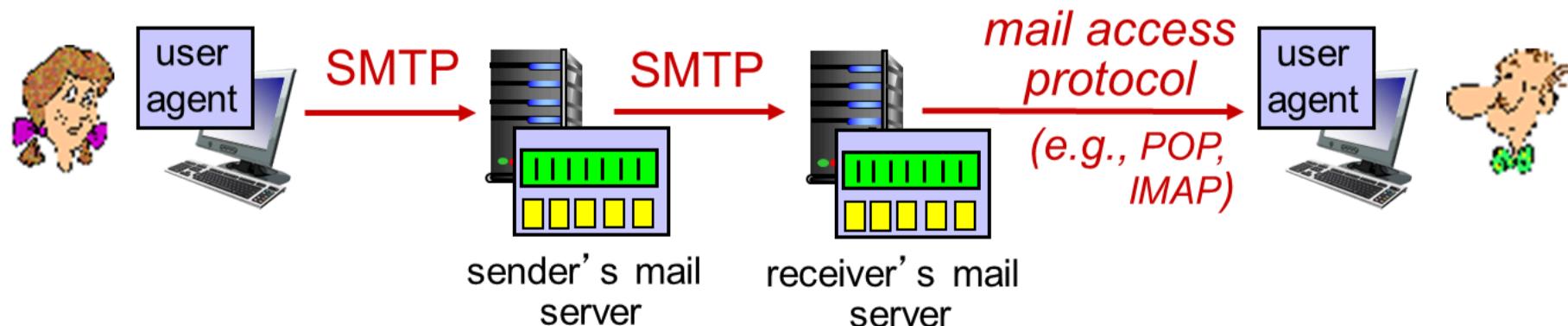
## 邮件报文格式



每个首部必须含有一个From: 首部行和一个To: 首部行；一个首部也许包含一个Subject: 首部行以及其他可选的首部行  
在报文首部之后，紧接着一个空白行，然后是以ASCII格式表示的报文体。

## 邮件访问协议

SMTP用来将邮件从发送方的邮件服务器传输到接收方的邮件服务器； SMTP也用来将邮件从发送方的用户代理传送到发送方的邮件服务器。如POP3这样的邮件访问协议用来将邮件从接收方的邮件服务器传送到接收方的用户代理。



- POP3：当用户代理（客户）打开了一个到邮件服务器（服务器）端口 110上的TCP连接后，POP3就开始工作了。POP3按照三个阶段进行工作：**特许 (authorization)**、**事务处理**以及**更新**。在第一个阶段即**特许**阶段，用户代理发送（以明文形式）用户名和口令以鉴别用户。在第二个阶段即**事务处理**阶段，用户代理取回报文；同时在这个阶段用户代理还能进行如下操作，对报文做删除标记，取消报文删除标记，以及获取邮件的统计信息。在第三个阶段即**更新**阶段，它出现在客户发出了 quit命令之后，目的是结束该POP3会话；
- IMAP：POP3协议没有给用户提供任何创建远程文件夹并为报文指派文件夹的方法，因此IMAP诞生。IMAP协议为用户提供了创建文件夹以及将邮件从一个文件夹移动到另一个文件夹的命令。IMAP还为用户提供了在远程文件夹中查询邮件的命令，按指定条件去查询匹配的邮件
- 基于WEB的电子邮件：许多用户与其远程邮箱之间通过HTTP进行通信。

## DNS

我们通常喜欢通过记忆主机名来访问远程主机，然而路由器通常记录定长的有着层次结构的IP地址。因此我们可以使用域名系统 (Domain Name System, DNS) 来进行主机名到IP地址转换的服务。

除此以外，DNS还提供其他服务：

主机别名 (host aliasing)

邮件服务器别名 (mail server aliasing)

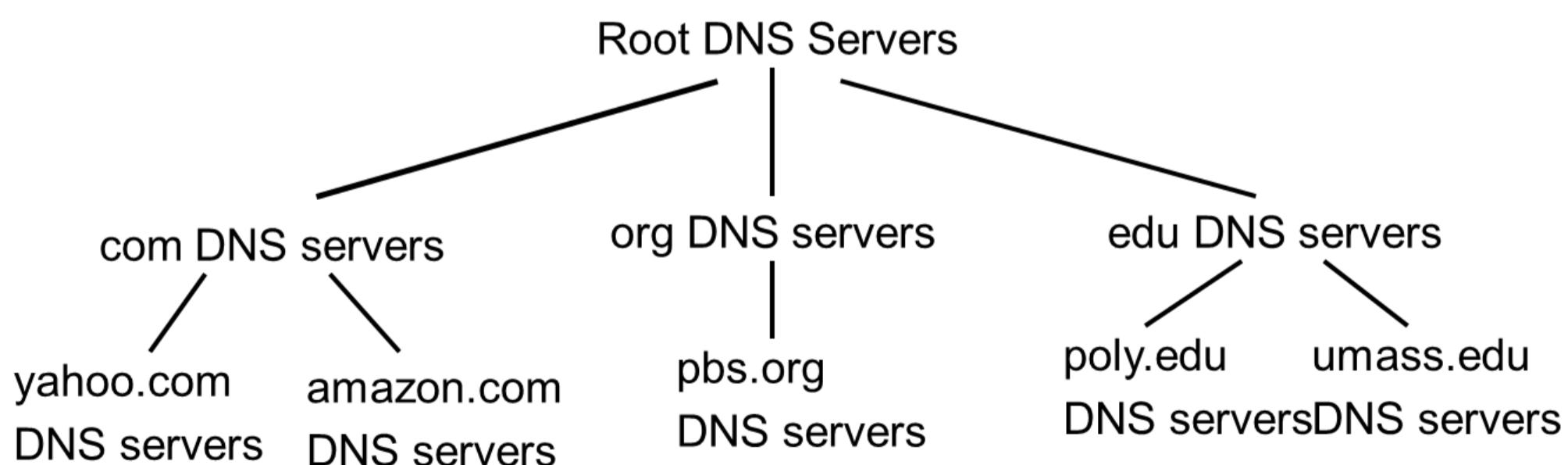
负载分配 (load distribution)：用于为繁忙的站点提供冗余服务器负载分配服务，缓解压力

## DNS 结构

我们不能中心化DNS，出于以下考虑

- 单点故障 (a single point of failure)：一点崩溃导致整个互联网崩溃
- 通信容量 (traffic volume)：单服务器难以处理所有DNS查询请求
- 远距离的集中式数据库(distant centralized database)：距离中心服务器遥远的查询请求将具有巨大时延
- 维护(maintenance)：单服务器保留所有因特网主机记录，维护成本巨大

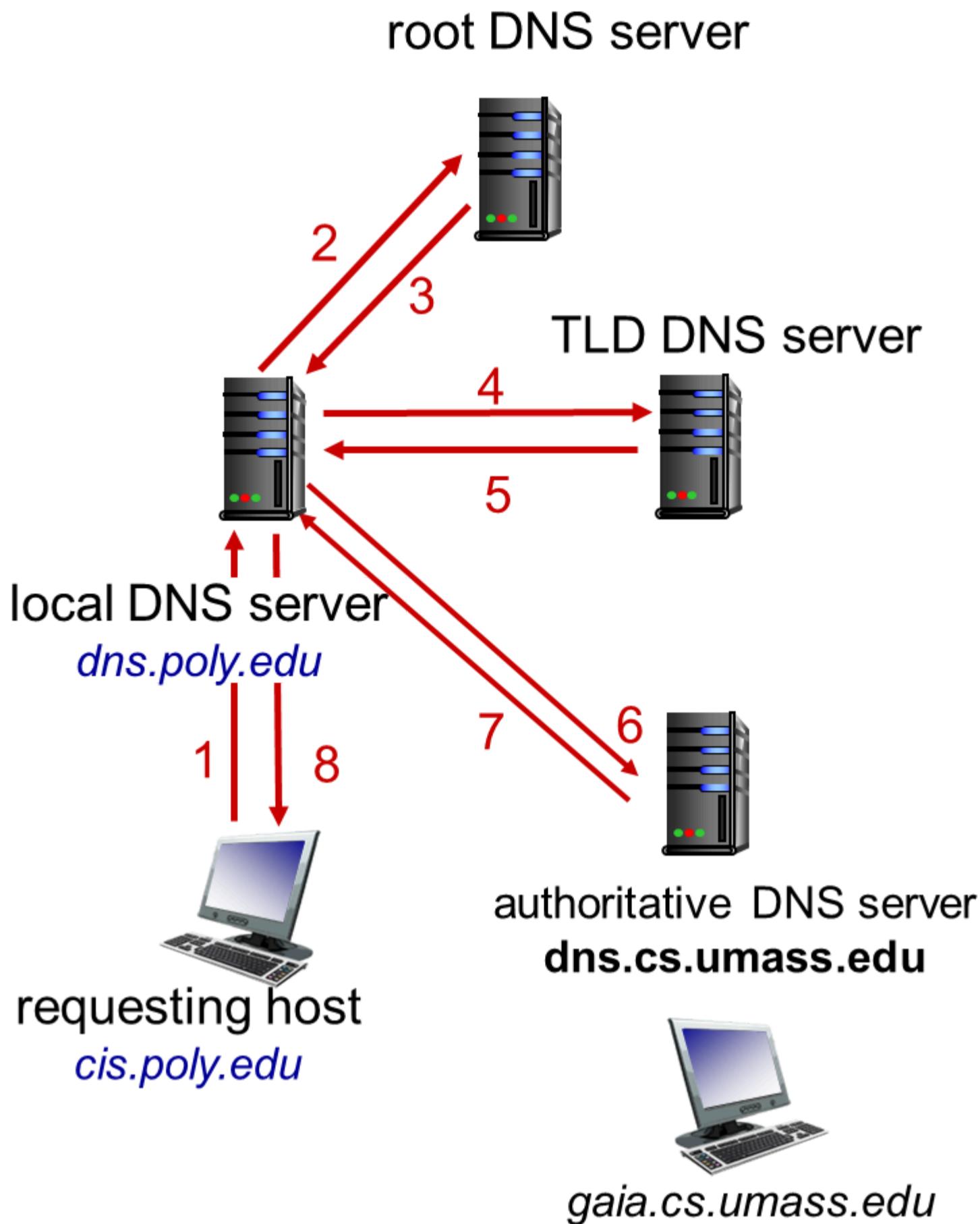
因此考虑分布式层次化数据库 a distributed, hierarchical database



父服务器保存子服务器的IP地址

- 根DNS服务器 Root Name Servers：当本地DNS服务器无IP信息时，联系根DNS服务器，其会返回TLD服务器的IP
- 顶级域(TLD)服务器：对所有顶级域如com, org, net, edu负责，提供权威服务器的地址
- 权威DNS服务器 Authoritative DNS servers：组织持有，管理其可访问主机的DNS记录，用于将主机名字映射成IP返回

本地DNS服务器 (local DNS server) : 不属于层次内, 在每个ISP (residential ISP, company, university) 内存在。当主机发出 DNS请求时, 该请求被发往本地DNS服务 器, 它起着代理的作用, 并将该请求转发 到DNS服务器层次结构中。



## DNS caching

当某DNS 服务器接收一个DNS回答(例如, 包含某 主机名到IP地址的映射)时, 它能将映射缓存在本地存储器中。当另一个对相同主机名的查询到达该DNS服务器时, 该DNS服务器就能够提供所要求的IP地址, 即使它不是该主机名的权威服务器。映射并不是永久的, DNS服务器在一段时间后(通常设置为两天)将丢弃缓存的信息。

## DNS 记录

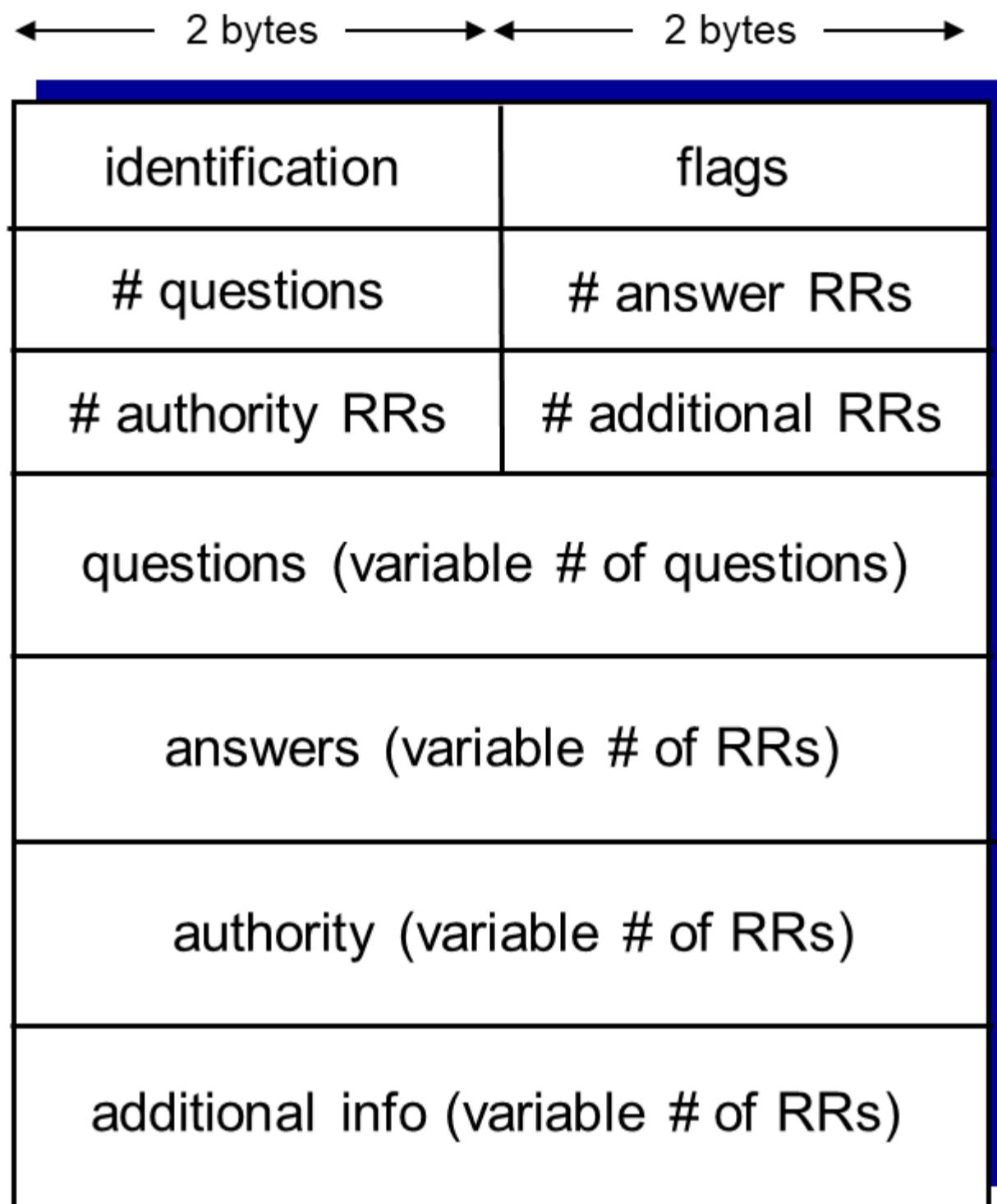
共同实现DNS分布式数据库的所有DNS服务器存储了资源记录(Resource Record, RR), RR提供了主机名到IP地址的映射

**RR format: (name, value, type, ttl)**

- 如果Type = A,则Name是主机名, Value是该主机名对应的IP地址。

- 如果Type = NS,则Name是个域(如foo. com),而Value是个知道如何获得该域中主机IP地址的权威DNS服务器的主机名。
- 如果Type=CNAME, 则Value是别名为Name的主机对应的规范主机名
- 如果Type = MX,则Value是个别名为Name的邮件服务器的规范主机名

## DNS报文



- 前12个字节是头部区域。包含16bits标识符, 16bits标志, 指出报文是查询报文还是回答报文。并可设置是否希望递归, 是否支持递归,
- 问题区域包含着正在进行的查询信息。该区域包括: 名字字段, 类型字段
- 回答区域包含回应的RR

## 插入DNS数据库

当新建公司, 需要:

1. 注册域名 register name 提供你的基本和辅助权威DNS服务器的名字和IP地址
2. 创建RR, 含NS和A类型

## P2P

P2P结构下没有集中的服务器, 终端之间直接进行通信

## 文件分发对比

问题：将文件（大小为  $F$ ）从一台服务器发送到  $N$  个对等服务器需要多少时间？

CS结构：

- 服务器首先向所有对等方发送一个副本，即  $NF$ ，假设上传速率为  $u_s$ ，则耗时  $NF/u_s$
- 设  $d_{min}$  表示具有最小下载速率的对等方的下载速率，则需要  $F/d_{min}$  时间进行下载

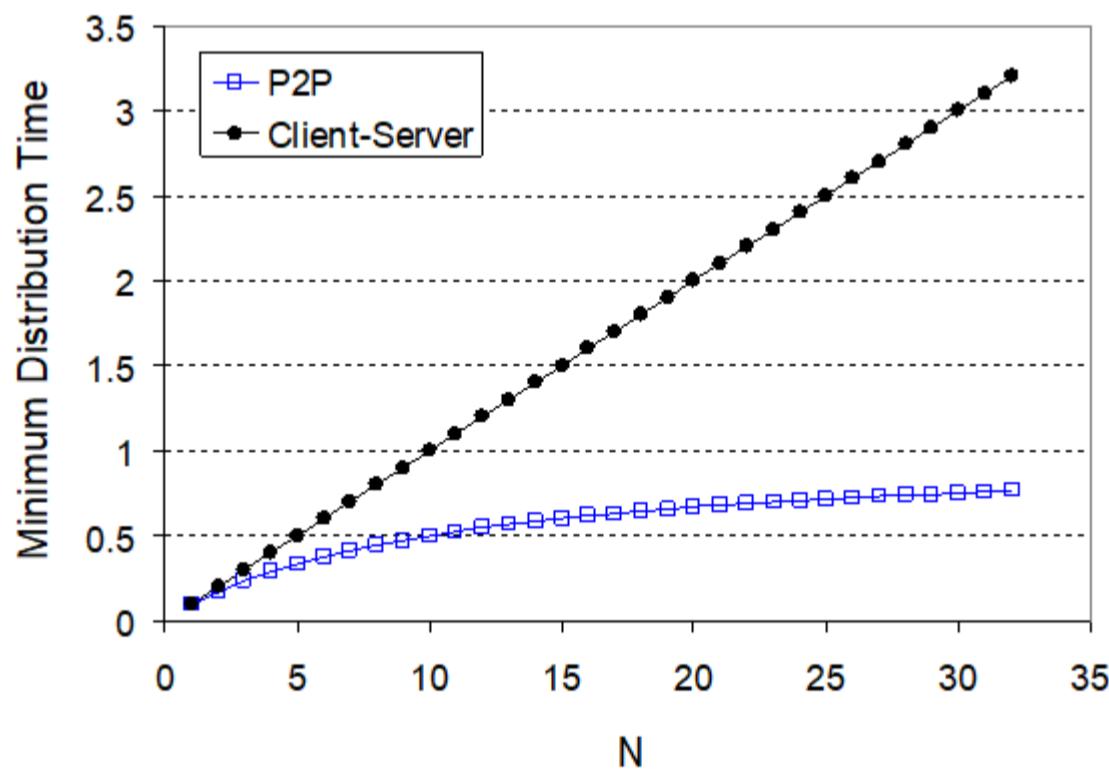
因此  $D_{cs} \geq \max\{NF/u_s, F/d_{min}\}$

P2P结构：

每个对等方能够帮助服务器分发该文件。特别是，当一个对等方接收到某些文件数据，它能够使用自己的上载能力重新将数据分发给其他对等方

- 服务器至少需要发送一个副本， $F/u_s$
- 具有最低下载速率的对等方下载  $F/d_{min}$
- 即使在帮助下，也需要上传共  $NF$ ，总上传速率为  $u_s + u_1 + \dots + u_N$

因此  $D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + u_1 + \dots + u_N)\}$



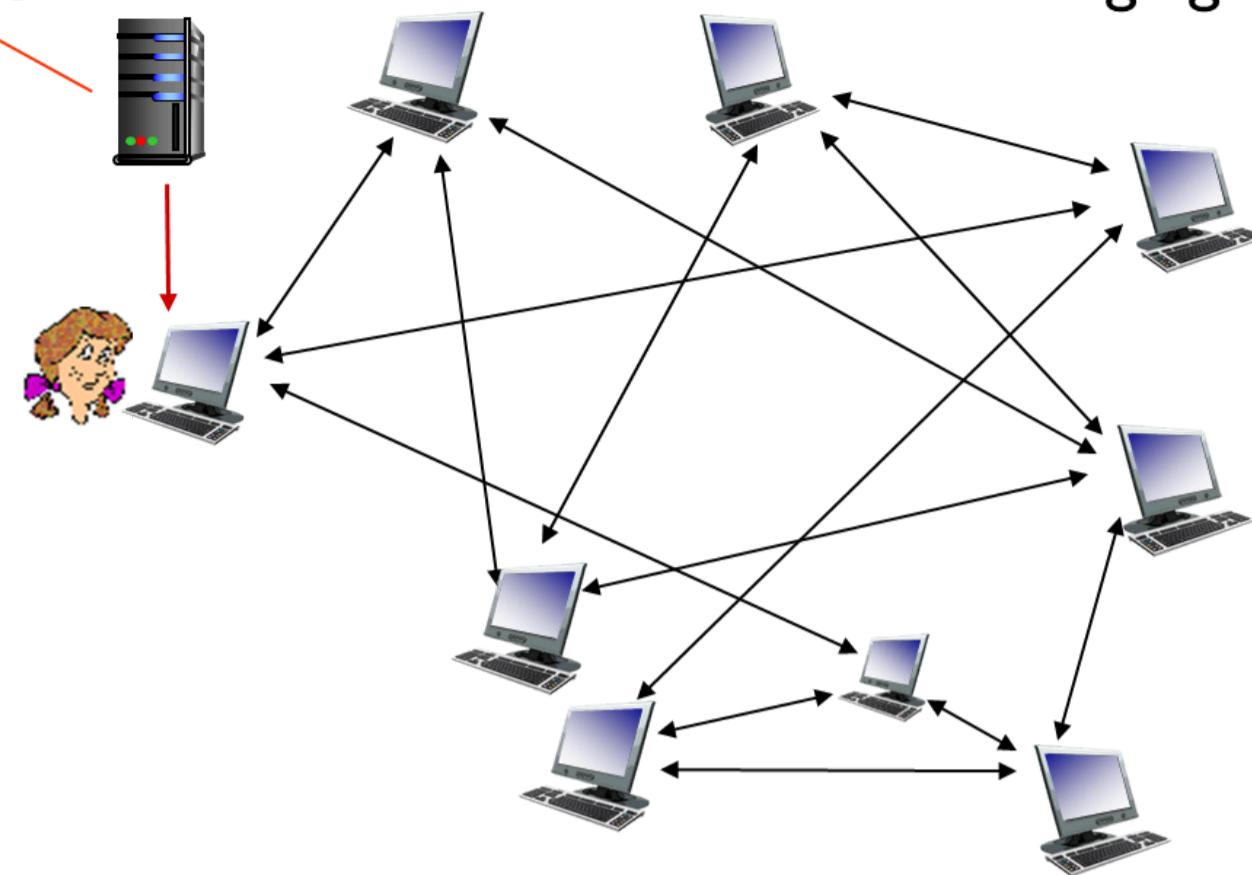
对等方除了是比特的消费者外还是它们的重新分发者。

## BitTorrent

参与一个特定文件分发的所有对等方的集合被称为一个洪流（torrent）。在一个洪流中的对等方彼此下载等长度的文件块（chunk），典型的块长度为 256KB。当一个对等方首次加入一个洪流时，它没有块。随着时间的流逝，它累积了越来越多的块。当它下载块时，也为其他对等方上载了多个块。一旦某对等方获得了整个文件，它也许（自私地）离开洪流，或（大公无私地）留在该洪流中并继续向其他对等方上载块。同时，任何对等方可能在任何时候仅具有块的子集就离开该洪流，并在以后重新加入该洪流中。

**tracker:** tracks peers participating in torrent

**torrent:** group of peers exchanging chunks of a file



每个洪流具有一个基础设施节点，称为追踪器（tracker）。当一个对等方加入某洪流时，它向追踪器注册自己，并周期性地通知追踪器它仍在该洪流中。以这种方式，追踪器跟踪参与在洪流中的对等方。

当一个新的对等方Alice加入该洪流时，追踪器随机地从参与对等方的集合中选择对等方的一个子集（为了具体起见，设有50个对等方），并将这50个对等方的IP地址发送给Alice。Alice持有对等方的这张列表，试图与该列表上的所有对等方创建并行的TCP连接。并通过这些邻居请求块和发送块，不同的邻居持有不同的块，Alice周期性地询问他们拥有的块列表。

**向谁请求块？：**

最稀缺优先 (rarest first)：针对她没有的块在她的邻居中决定最稀缺的块（最稀缺的块就是那些在她的邻居中副本数量最少的块），并首先请求那些最稀缺的块。这种方法可以大致平衡torrent中各个块的数量

**向谁发送块？：**

一报还一报 (tit-for-tat)：Alice根据当前能够以最高速率向她提供数据的邻居，给出其优先权。确定以最高速率流入的4个邻居，被称为疏通 (unchoked)。除了以上4个外，还要随机选择1个进行交换，这样的话这双方都有可能会互相发送（速度够快的话），如果成果，则放入4个最优邻居列表之中。因此一共向4+1个邻居进行发送。

## CDNs

视频占据很大部分网络带宽，Youtube甚至有约1B用户

这种情况下不可以使用单一的视频服务器，因为用户的网络能力差距显著，因此应该考虑分布式应用程序基础设施 (distributed, application-level infrastructure)

此外可通过压缩视频的比特率来生成不同版本的视频，让用户根据其网络能力自由选择

## HTTP Streaming and DASH

在HTTP流中，视频只是一个URL，用户发送GET请求后服务器与用户建立TCP连接并发送视频文件。

然而不同用户在不同时间内可用的带宽大小可能有很大差别，因此我们需要经HTTP的动态适应性流(Dynamic Adaptive Streaming over HTTP, DASH)。

视频编码为几个不同的版本，其中每个版本具有不同的比特率，对应于不同的质量水平。客户动态地请求来自不同版本且长度为几秒的视频段数据块。当可用带宽量较高时，客户自然地选择来自高速率版本的块；当可用带宽量较低时，客户自然地选择来自低速率版本的块

服务端：

- 将视频分为多个块
- 每个存储的块用不同速率存储
- 使用告示文件 (manifest file) 来为每个版本提供了一个URL及其比特率

客户端：

- 定期测量带宽
- 根据带宽选择URL及下载的块
- 在不同时间点选择不同的块

## 内容分发网 Content distribution networks

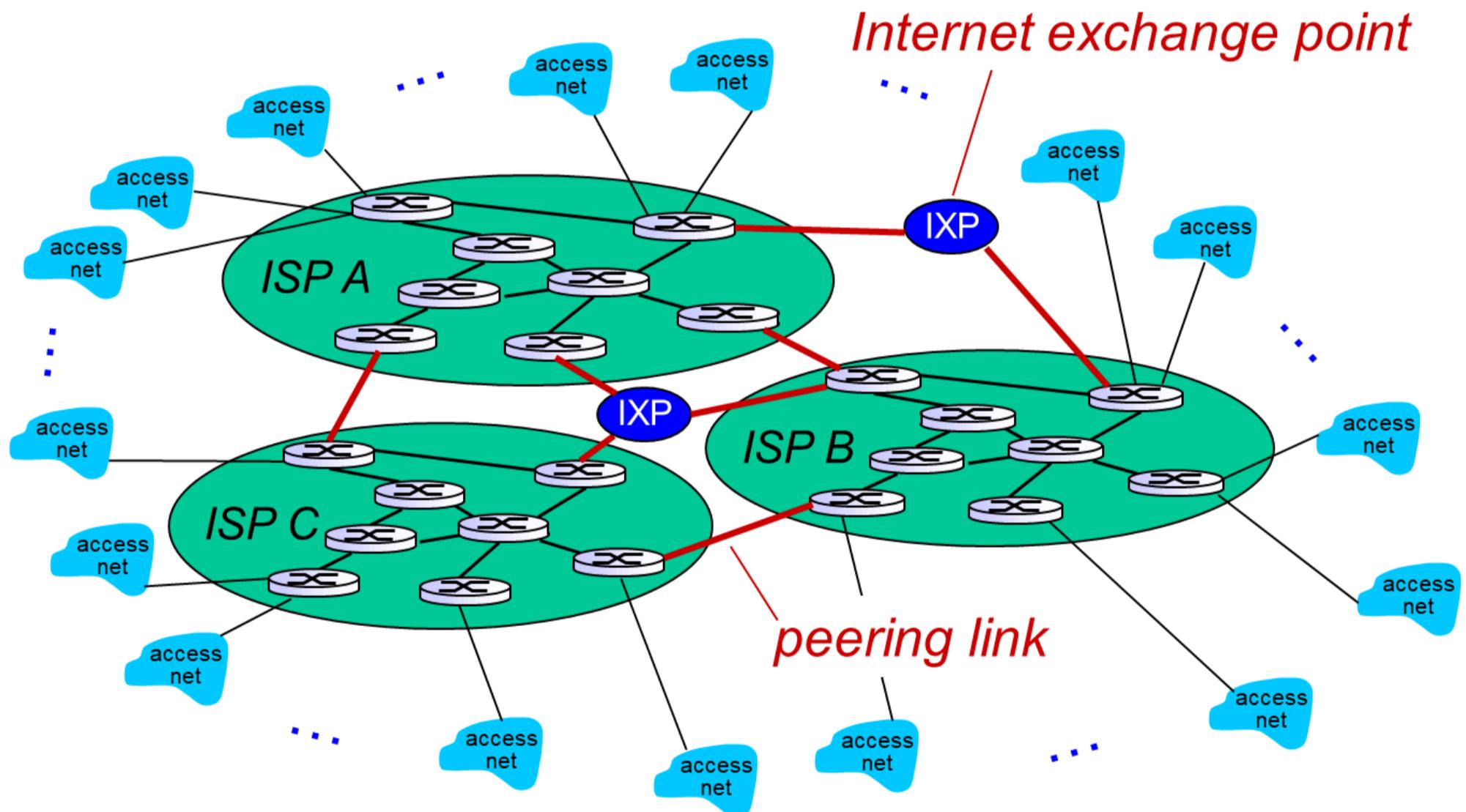
视频公司如何向位于全世界的所有用户流式传输所有流量同时提供连续播放和高交互性？

方法1：建立单一的巨大服务器（不现实）

方法2：在多个分散的站点存储/提供多个视频副本

CDN两种理念：

- 深入 (*enter deep*)：在多个ISP中部署服务器群，尽可能接近用户
- 邀请回家 (*bring home*)：在少量（例如10个）关键位置如在因特网交换点 (IXP) 建造服务器群，由ISP邀请进入。维护成本较低



CDN存储副本，用户从CDN请求

## 3. Transport Layer

### 运输层服务

运输层协议为运行在不同主机上的应用进程之间提供了逻辑通信 (logic communication) 功能，使得运行不同进程的主机好像直接相连一样

运输层协议是在端系统中而不是在路由器中

发送端：运输层将从发送应用程序进程接收到的报文转换成运输层分组（segment）

接收端：将分组组合成消息递交给应用层

网络层负责主机(host)之间的逻辑通信

传输层负责进程之间的逻辑通信

两种协议：

**TCP**: 可靠的有序交付

- 拥塞控制 (congestion control)
- 流控制 (flow control)
- 连接设置 (connection setup)

**UDP**: 不可靠、无序的交付

- 尽力而为 (best-effort")

## 多路复用与多路分解 multiplexing and demultiplexing

---

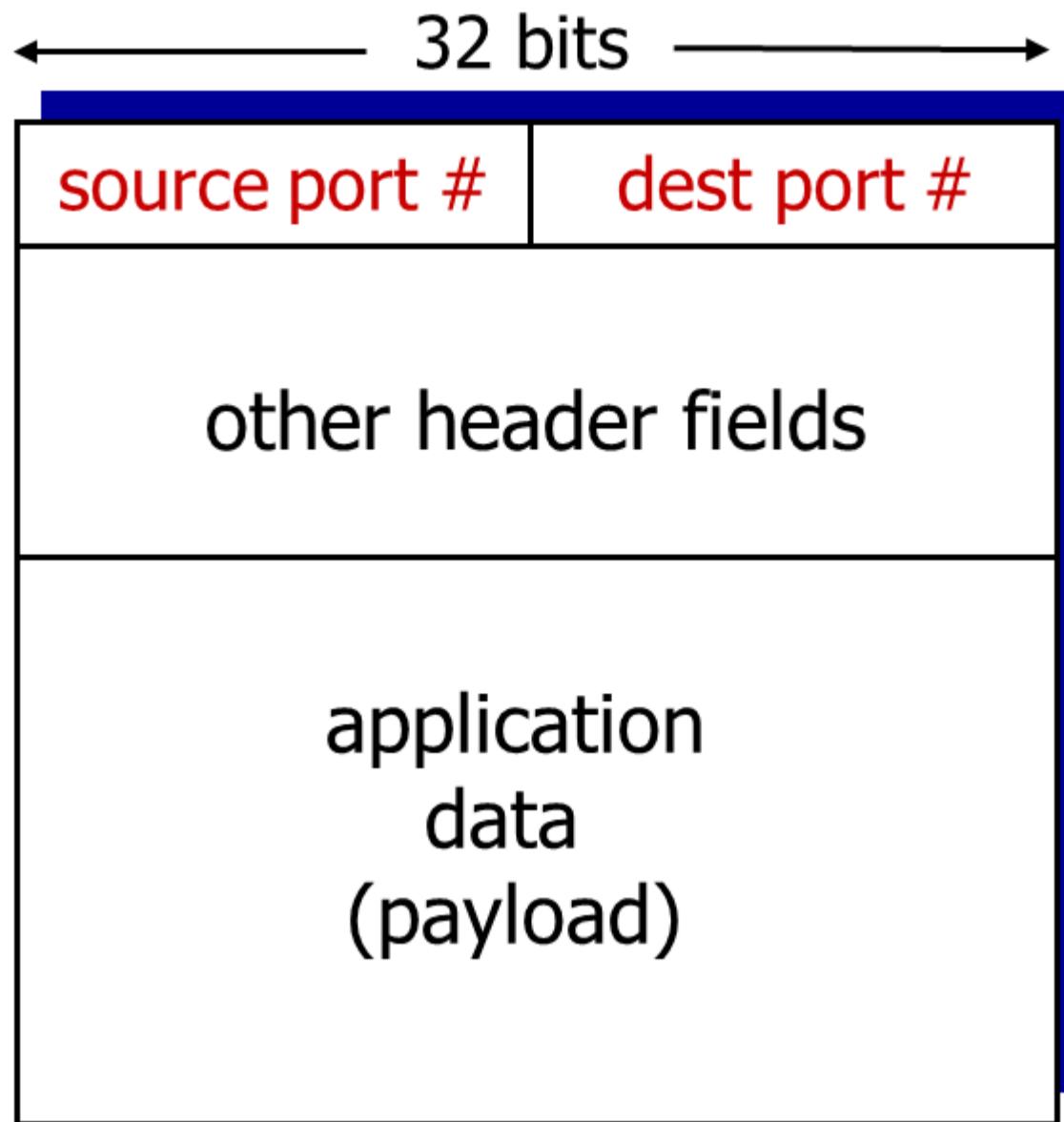
发送方的多路复用：处理来自多个 sockets的数据，添加传输标头，将 segment 传递到网络层

接收方的多路分解：使用header信息来将接收到的segment交付到正确套接字

多路分解是如何工作的？

主机使用 目标IP 地址和端口号将分段定向到适当的套接字

传输层报文段 (segment) 有源端口和目标端口以及源IP和目标IP，由此将其交给目标socket并由其交给正确进程



## TCP or UDP segment format

无连接的多路分解：

- UDP **套接字**由一个由（目标IP地址和目标端口号）组成的二元组完全标识。
- 如果两个UDP报文段有不同的源IP地址和/或源端口号，但具有相同的目的IP地址和目的端口号，那么这两个报文段将通过相同的目的套接字被定向到相同的目的进程

面向连接多路分解：

- TCP**套接字**是由一个四元组（源IP地址, 源端口号, 目的IP地址, 目的端口号）来标识的
- 在TCP多路分解中，只有四元组完全相同才会定向到同一socket，若只是有相同的目的IP地址和目的端口号，那么会被多路分解到不同套接字中

## UDP

UDP有以下特点：

- 尽力交付：可能丢失 (lost)，无序交付 (out-of-order)
- 无连接 (connectionless)：无握手，UDP分段间独立

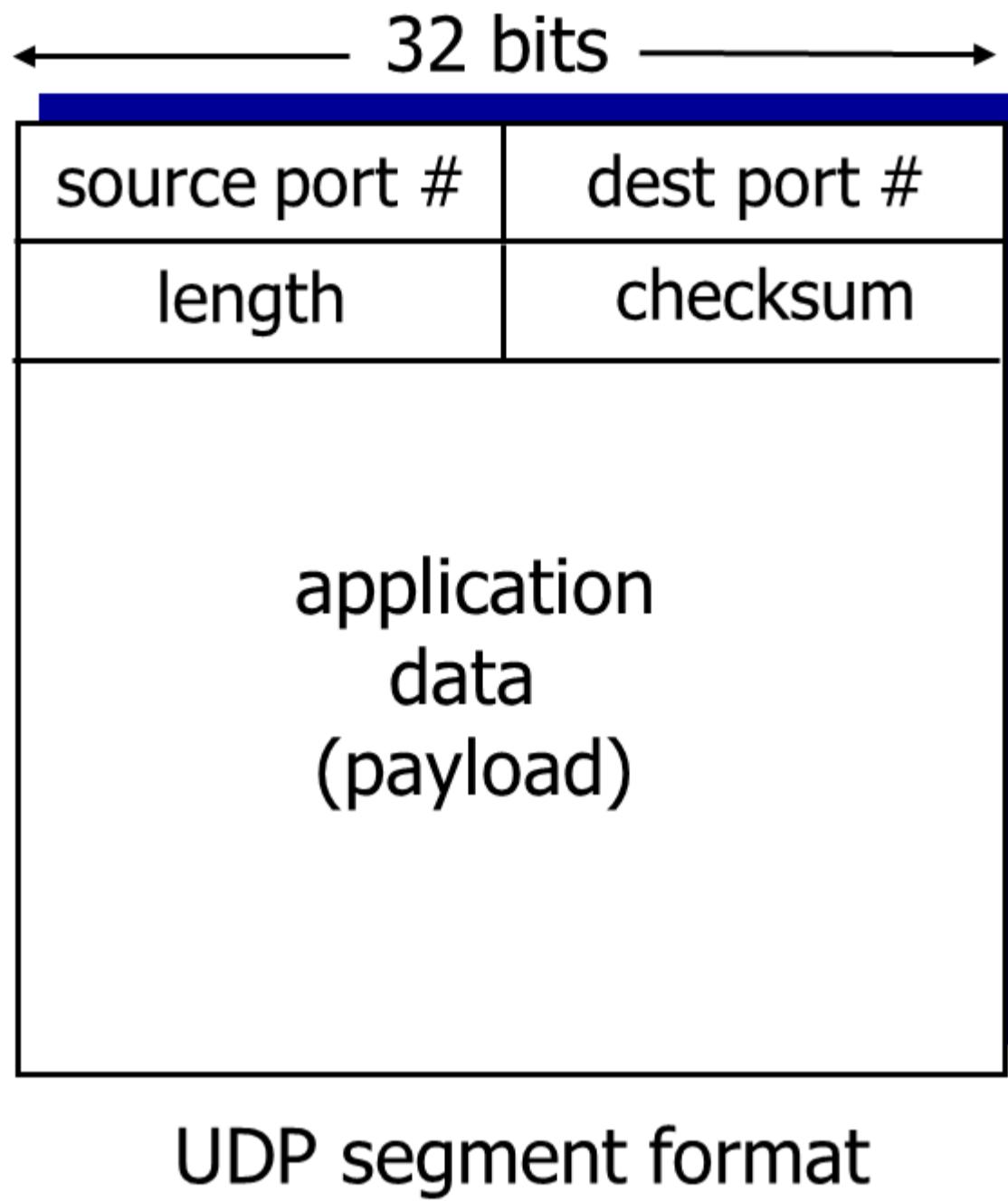
常见应用：

- 流式传输多媒体应用程序（容错、速率敏感） streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- 简单网络管理协议 (Simple network management protocol, SNMP)

如果想要基于UDP的可靠传输，那么就要增加应用层的可靠性或错误恢复功能

## UDP报头结构

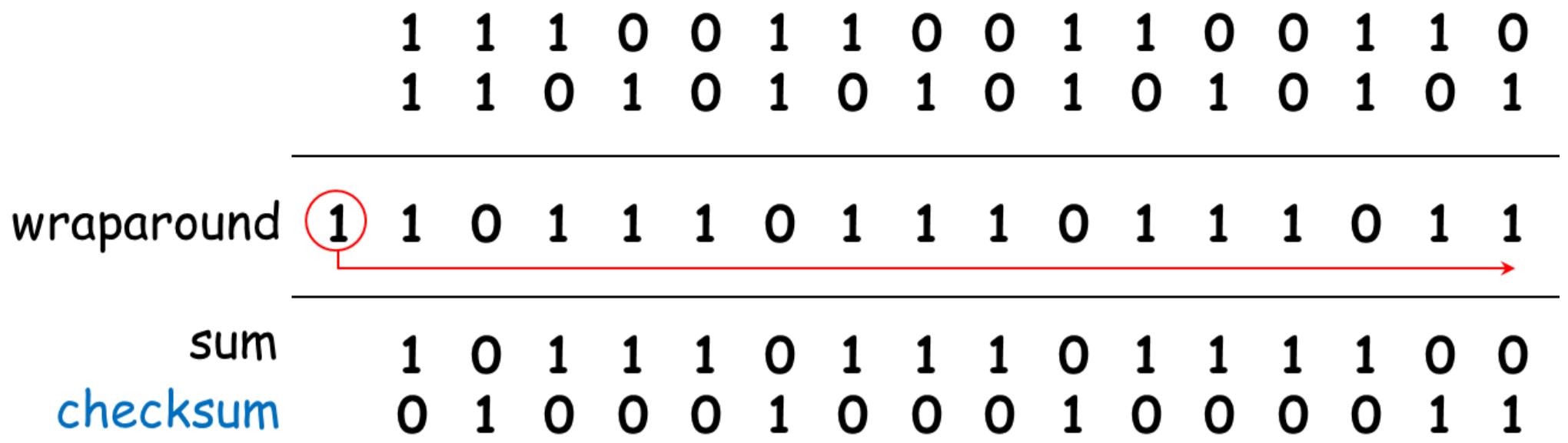
长度字段指示了在UDP报文段中的长度，数据和头部的总和，以字节为单位



UDP的优点如下：

- 无连接建立，减小了时延 delay
- 无连接，不需要复杂的维护
- 头部长度小
- 无拥塞控制，可以尽快传输

UDP校验和：发送方的UDP对报文段中的所有16比特字的和进行反码运算，求和时遇到的任何溢出都被回卷（将溢出的1加回到末尾）



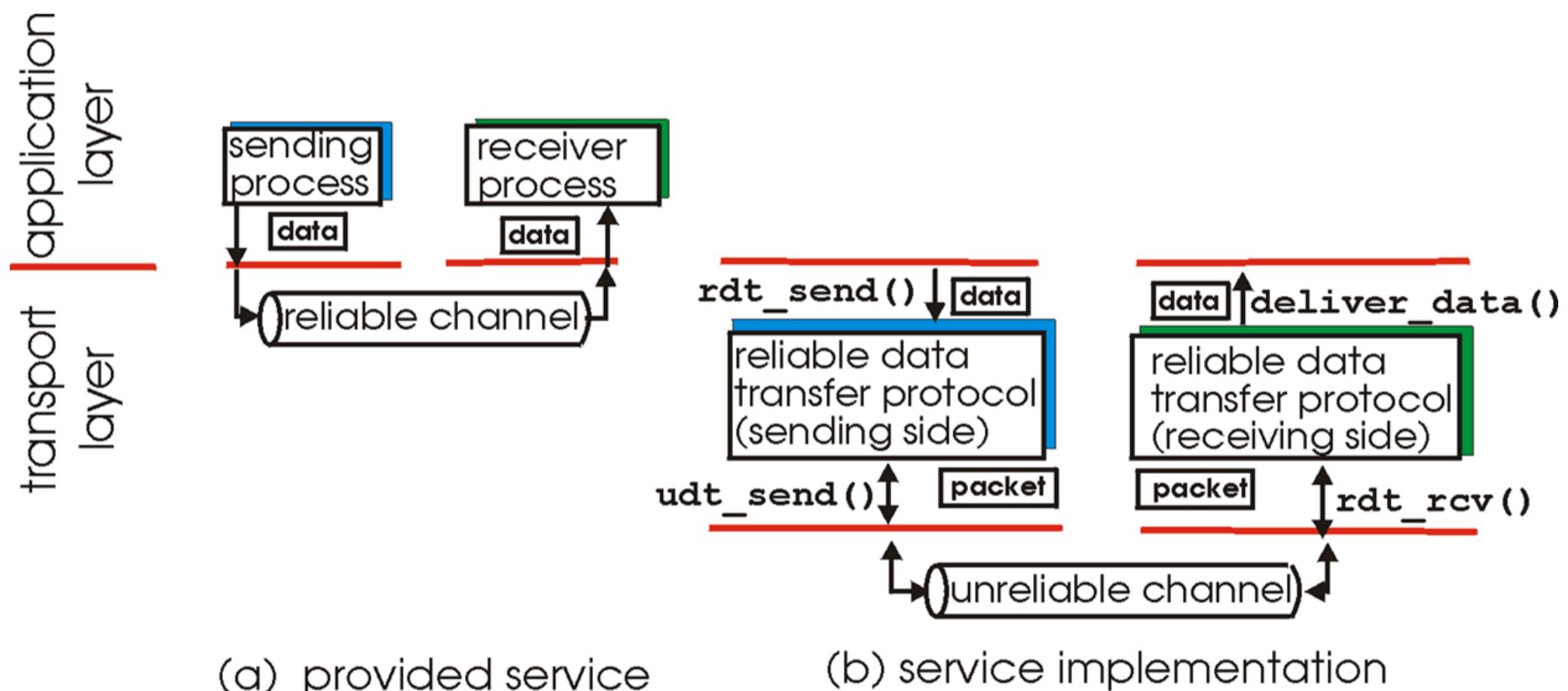
将sum进行反码即可得到校验和，一同发送给接收方。如果没有差错，那么接收方将受到的报文段以及校验和全部加在一起，得到的结果为16bit的1

# 可靠传输原理

可靠传输的实现不仅在运输层出现，也会在链路层以及应用层

在上层实体看来，数据可以通过一条可靠的信道进行传输，而这其实是TCP向调用它的因特网应用所提供的服务模型。

实际上下层的服务实现可能是通过不可靠的信道实现的，因此可靠数据传输协议（reliable data transfer protocol, rdt）的责任是如何通过不可靠的下层协议实现可靠的服务。

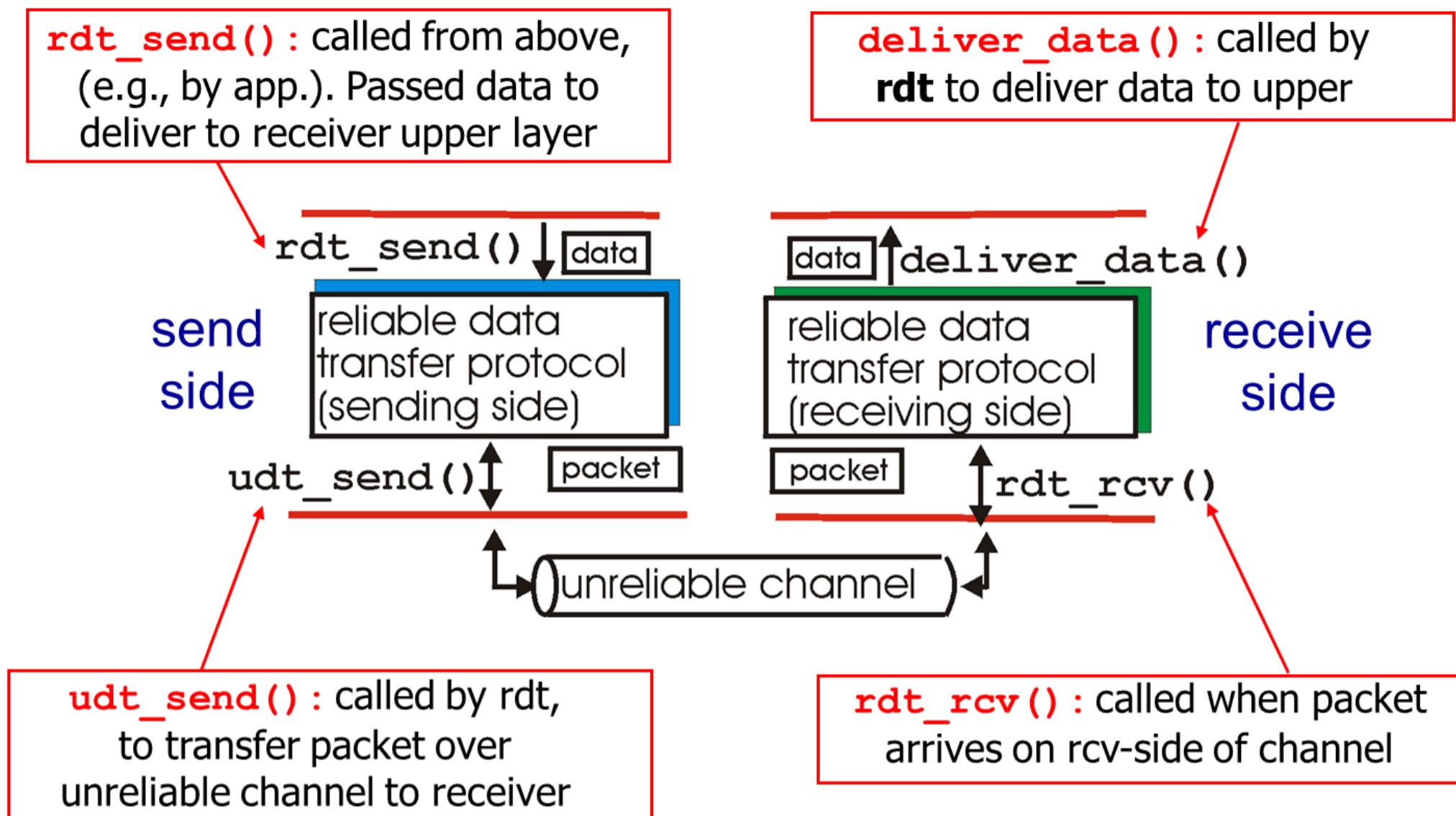


rdt send(): 上层调用，将要发送的数据交付给位于接收方的较高层。

udt send(): rdt调用，将数据包经过不可靠通道传输出给接收方

`rdt_rcv()`: 当数据包通过通道到达时被调用

deliverData(): rdt调用，将数据传输给上层

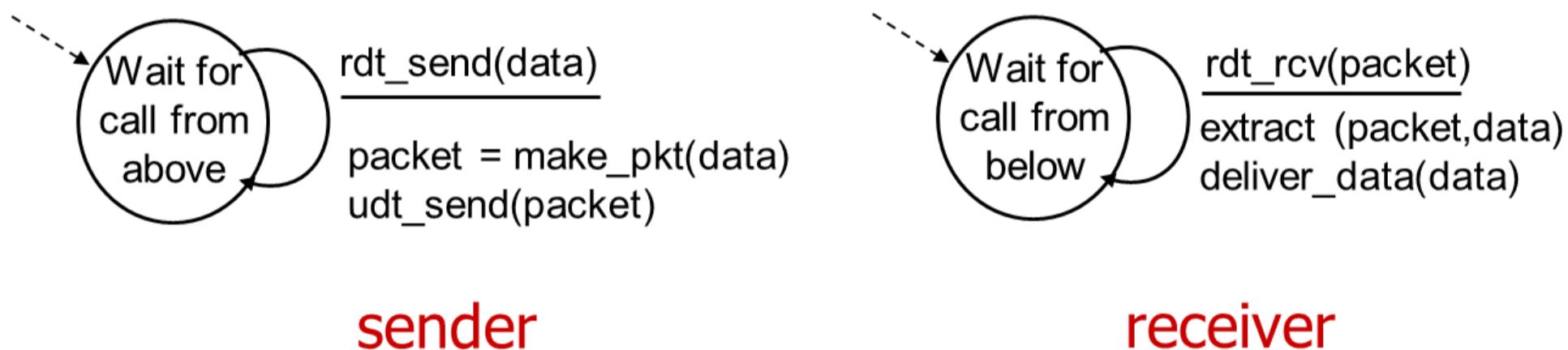


## rdt1.0

经完全可靠信道的可靠数据传输: rdt1.0

首先考虑最简单的情况，即底层信道是完全可靠的。

没有任何错误也没有任何丢失

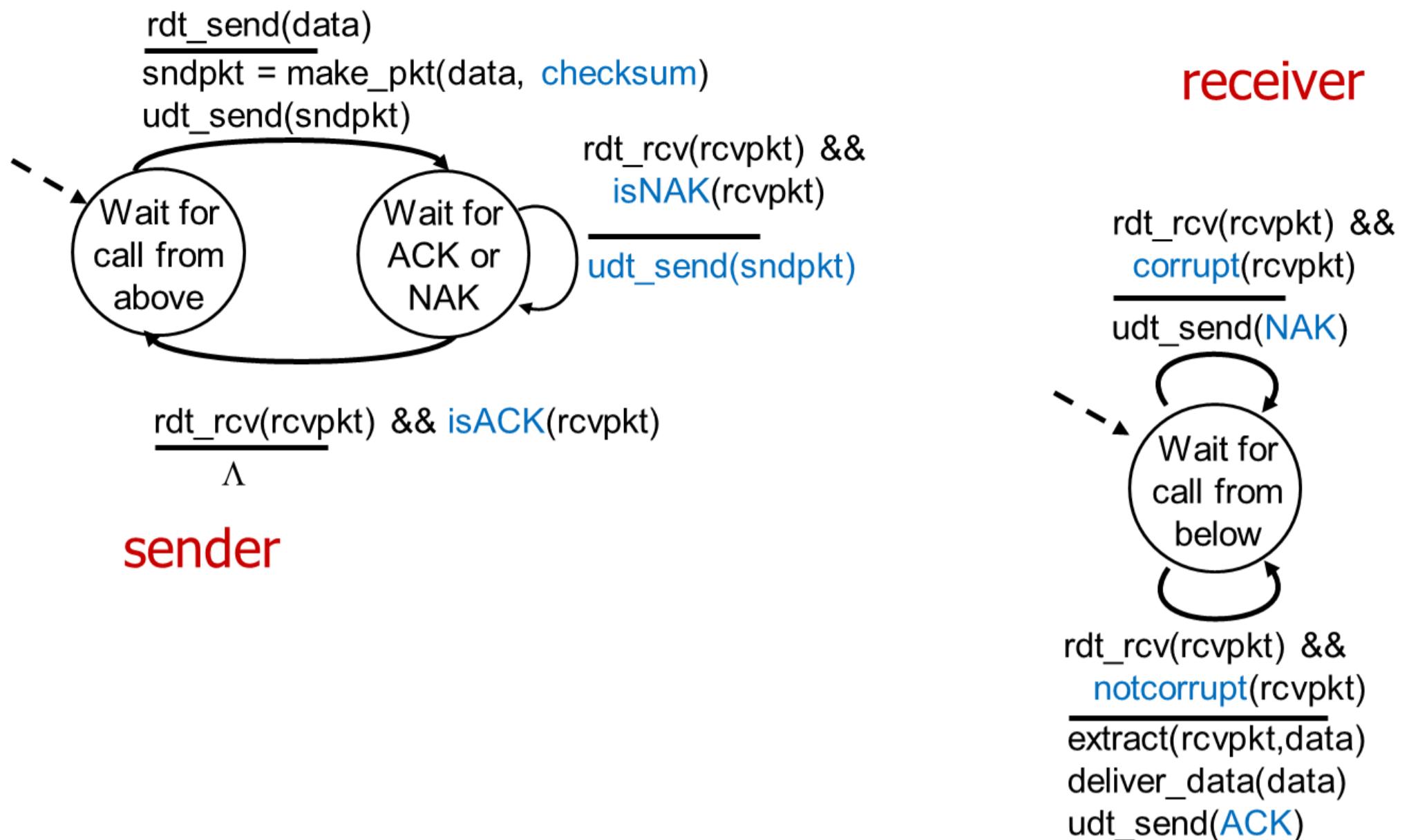


## rdt2.0

更为实际的模型是分组中的比特可能受损的模型

这种情况下需要引入差错检测error detection以及反馈信息ACK,NAK

发送方根据接收方的反馈决定是否重发信息



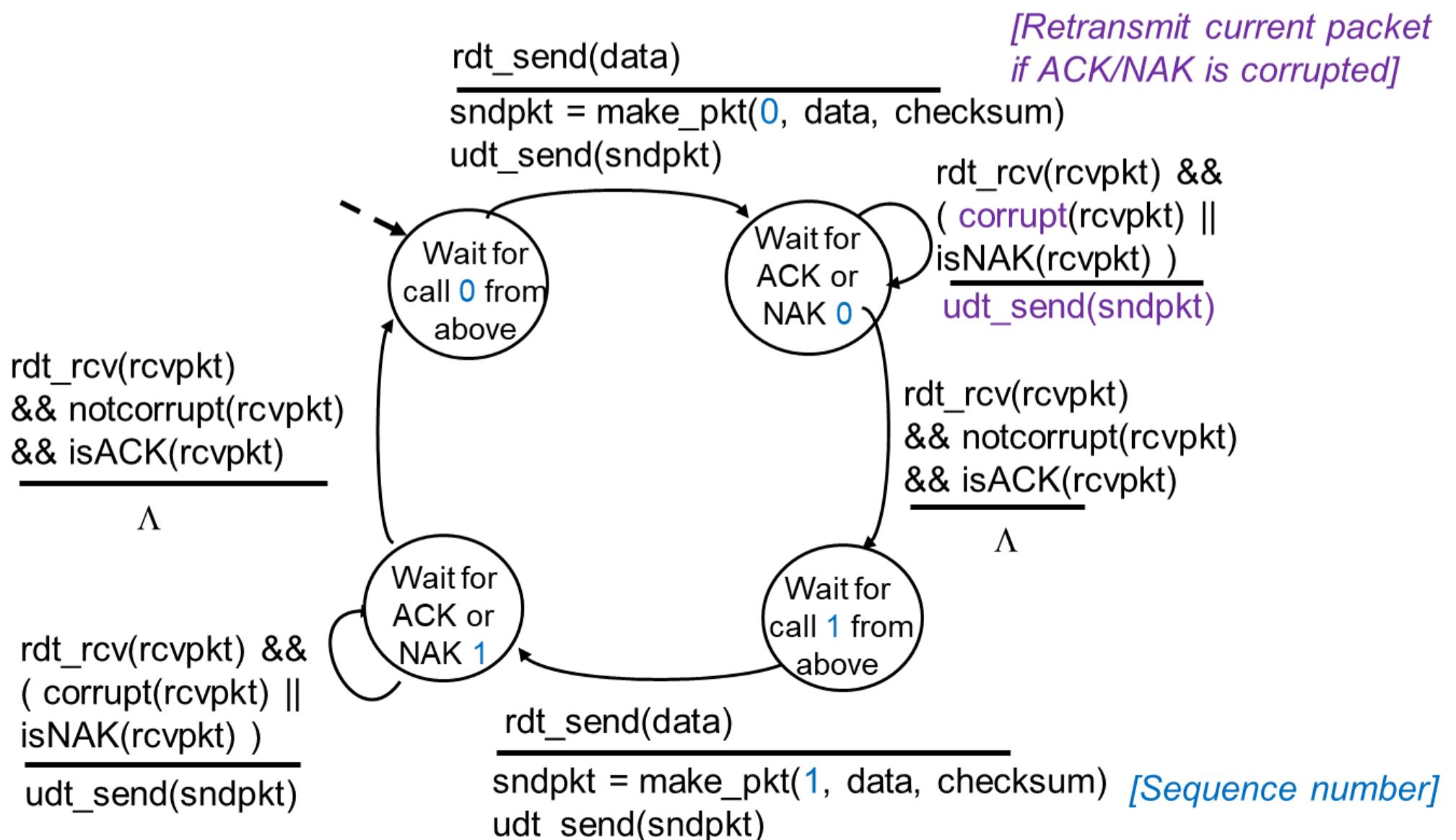
但是如果ACK或NAK出现差错重复怎么办？发送方可能无限重发，接收方无法得知是否收到应该收到的数据包

因此我们需要考虑加入序列号（sequence number）来应对此情况

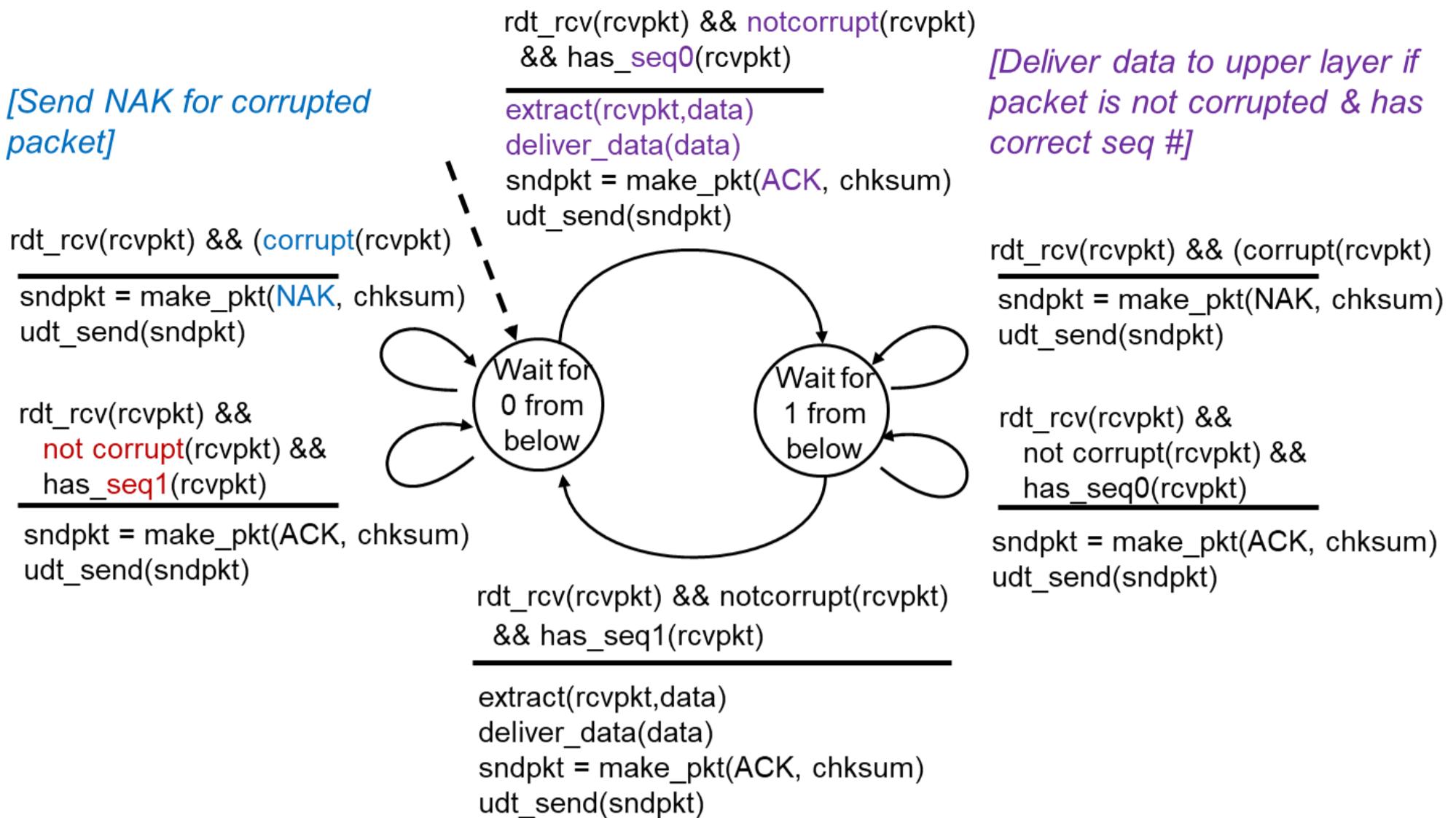
## rdt2.1

引入序列号0或1，因此接收可以精确确认哪个数据包需要发送方重传，接收方可以丢弃掉失误重传的数据包

正因为有两种序列号，FSM变成双倍，而且引入了校验和来确定ACK和NAK是否正确



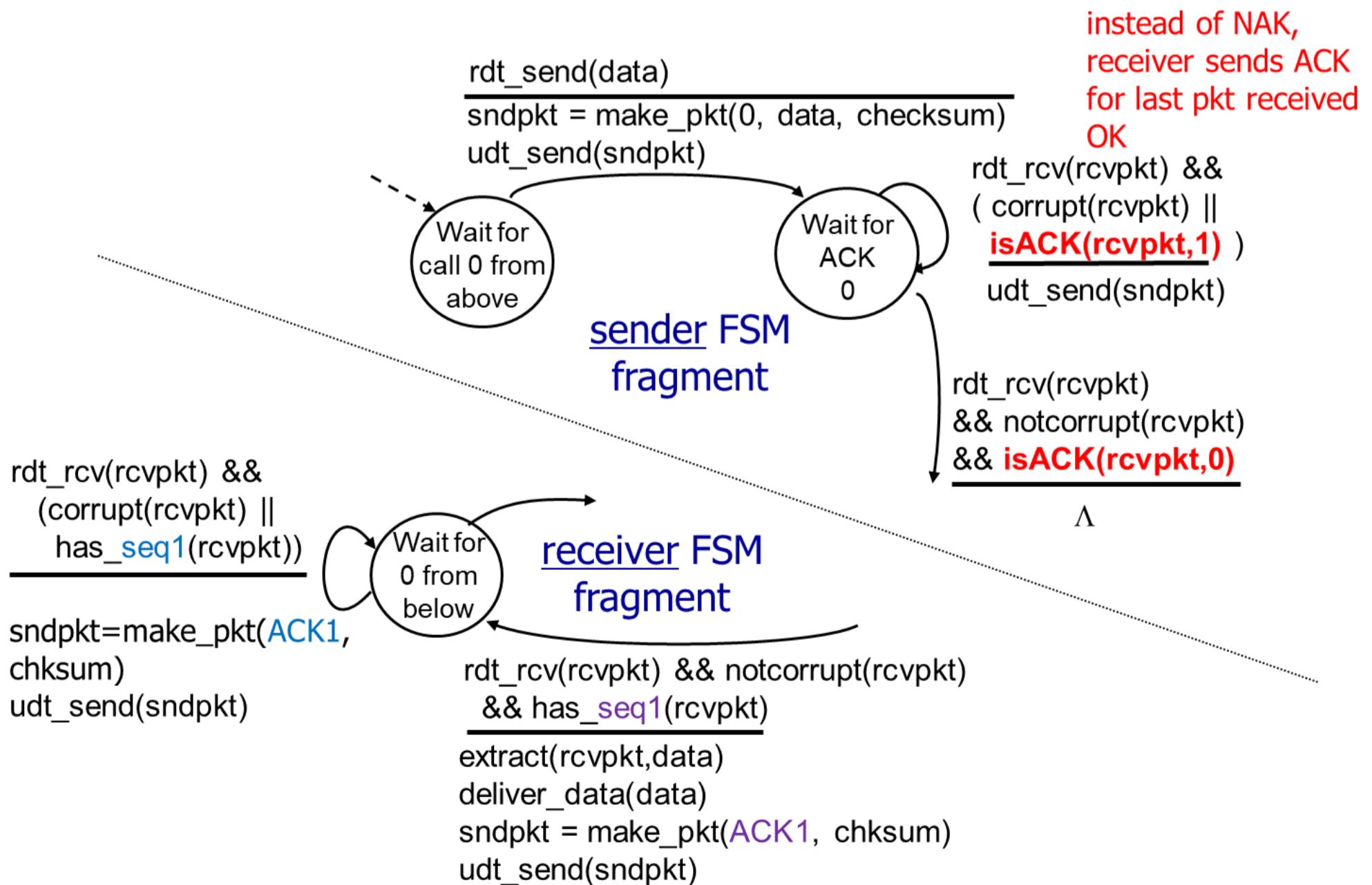
接收方会检测接收到的分组是否重复



NAK实际上是可以消除的，如果重复发送了同一个序号的ACK，那么发送方就可以得知上次发送是否有误。因此考虑ACK+SEQ

## rdt2.2

接收方发送最后一次受到的序列号的ACK，如果重复ACK同一个序号，那么发送方就得知上次发送有误，即可重传，替代NAK作用



## rdt3.0

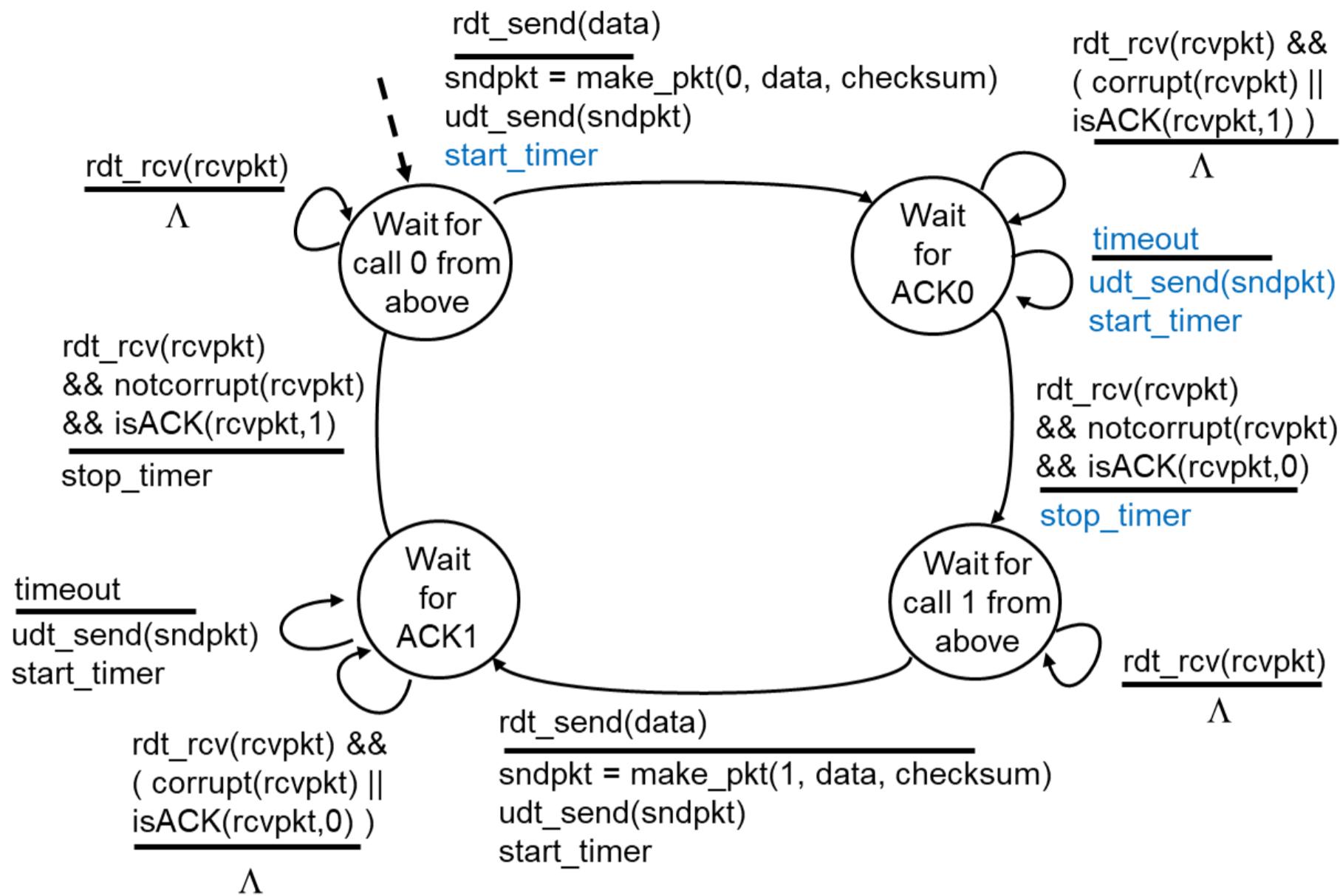
假定除了受损还有丢包情况出现，如何检测丢包？

如果没有检测丢包，那么双方会陷入死锁：

发送方等待确认，接收方等待数据包

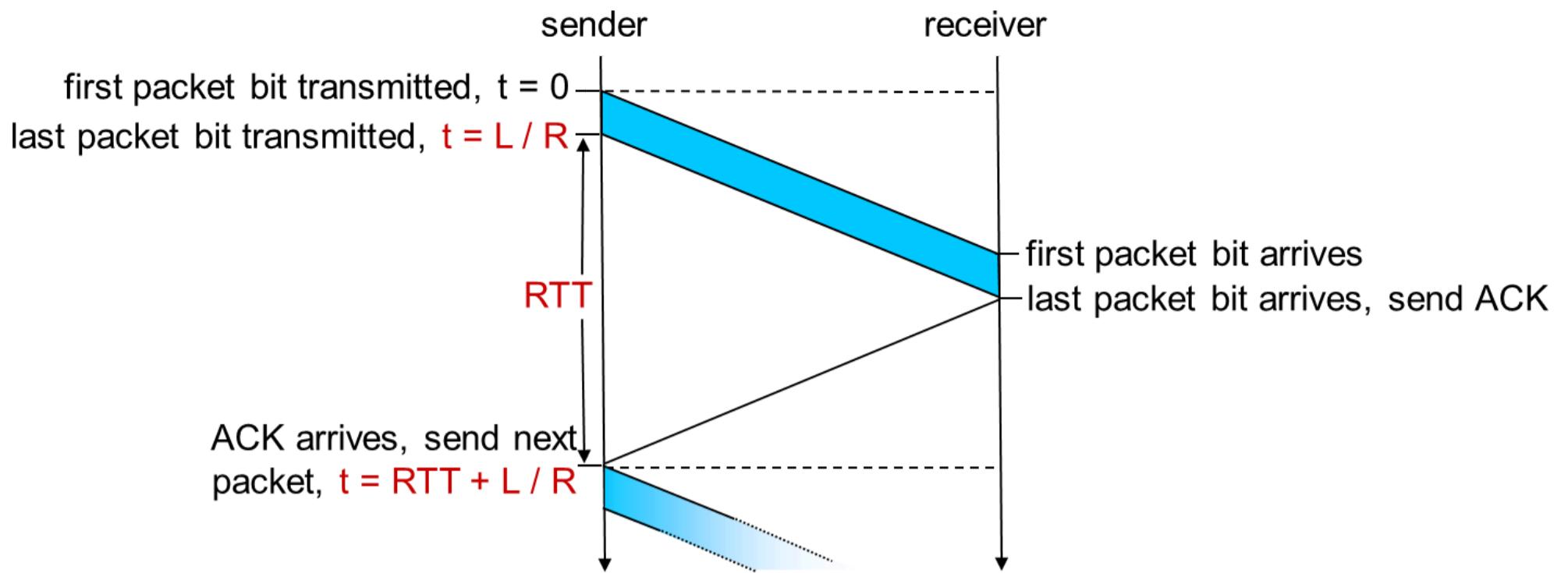
因此引入一个倒计数定时器（countdown timer），由发送方负责维护，只要倒计时结束后没有受到确认包则进行重传

*[Retransmits packet after timeout]*



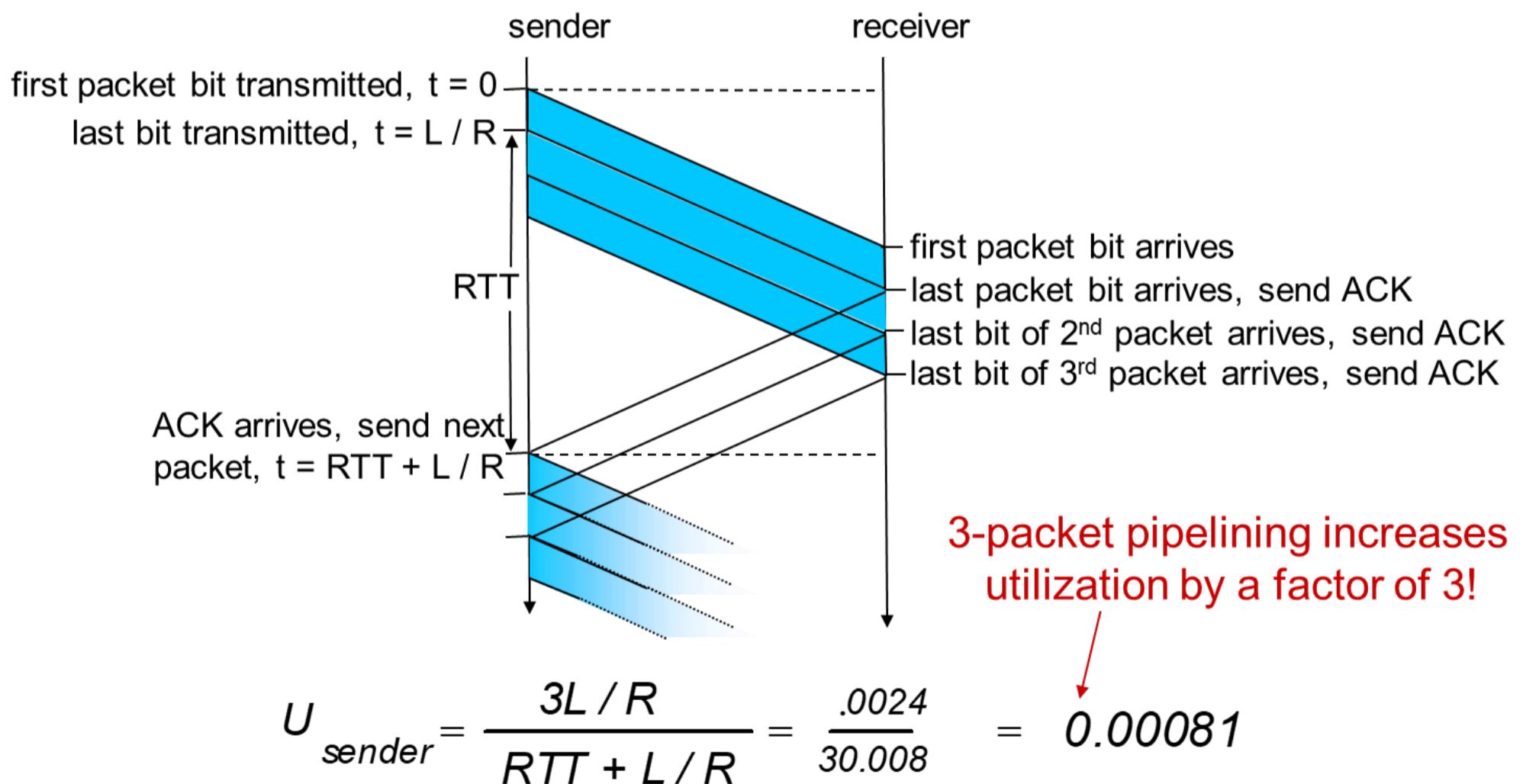
由于定时器由发送方维护，接收方的FSM和rdt2.2相同

但是rdt3.0的性能表现很差，因为它是停等（stop-and-wait）协议！利用率很低



- $U_{\text{sender}}$ : *utilization*, i.e., fraction of time sender busy sending
- Throughput =  $1 \text{ KB} / 30.008 \text{ msec} = 267 \text{ kbps}$   
given 1 Gbps link!
- ❖ network protocol limits use of physical resources!

因此考虑引入流水线技术来提供效率



这要求我们:

- 增加序号范围
- 发送和接收到的分组需要缓存: 发送方缓存刚发送的分组以确定是否需要重传, 接收方缓存接收的分组以确保按序交付给上层

为应对缓存和分组损坏, 丢失等情况, 引入流水线协议: 回退N步 (Go-Back-N, GBN)和选择重传(Selective Repeat, SR)

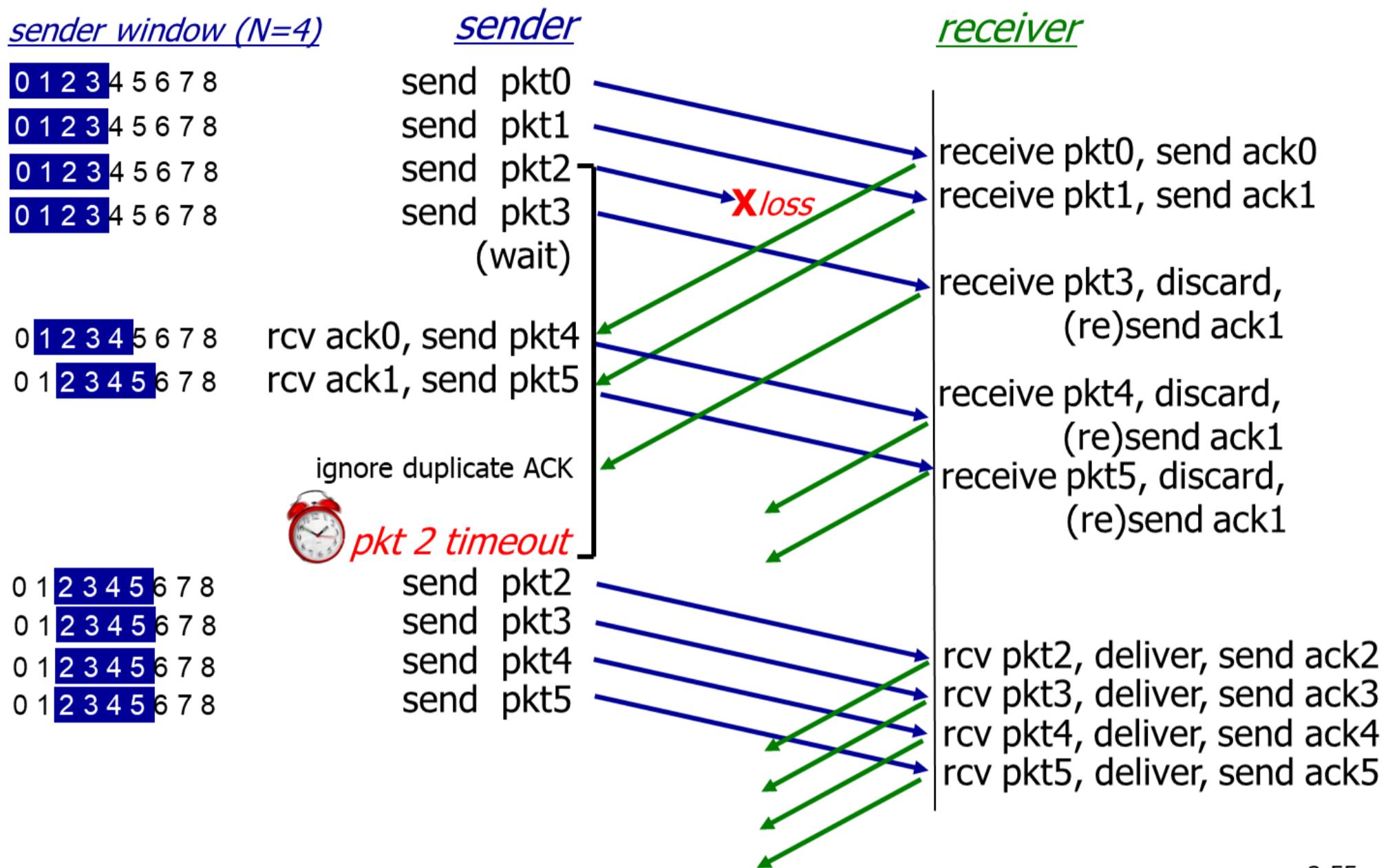
## Go-Back-N

发送方有发送窗口N，发送时连续发送窗口内数据包

接收方检查受到的包序号是否正确，如果有遗漏的序号则不发送该序号之后的ACK。接收方只发送累积ACK (cumulative ack) 用于告知该ACK之前的所有包接收无误。

发送方同时具有序号最小的未确认的包的计时器，如果计时器到期，则发送以该包为起点的之后的窗口内所有的包

发送方每当接收到一个ACK，则将窗口滑动到对应的序列数据包之后，表示窗口前的数据包发送完成



3-55

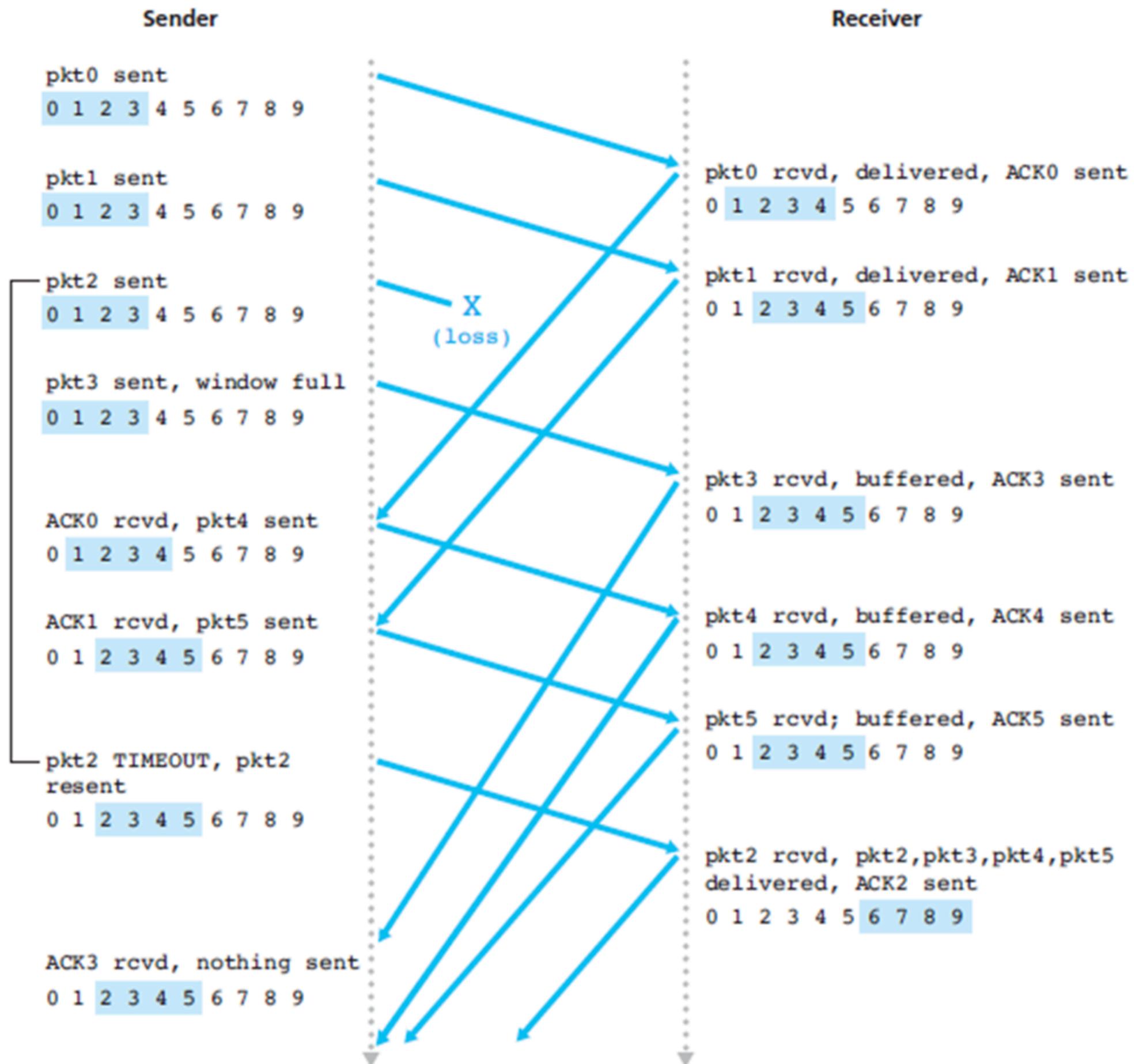
看似需要重新发送分组（例如pkt3,4,5），然而由于接收方出于按序交付的考虑丢弃了这些分组，这一方法给接收方的缓存减负了。

## Selective Repeat (SR)

发送方有发送窗口N，发送时连续发送窗口内数据包

接收方对每一个包都发送单独ACK

发送方对每个未确认数据包都保存ACK，当计时器过期时只单独发送该包

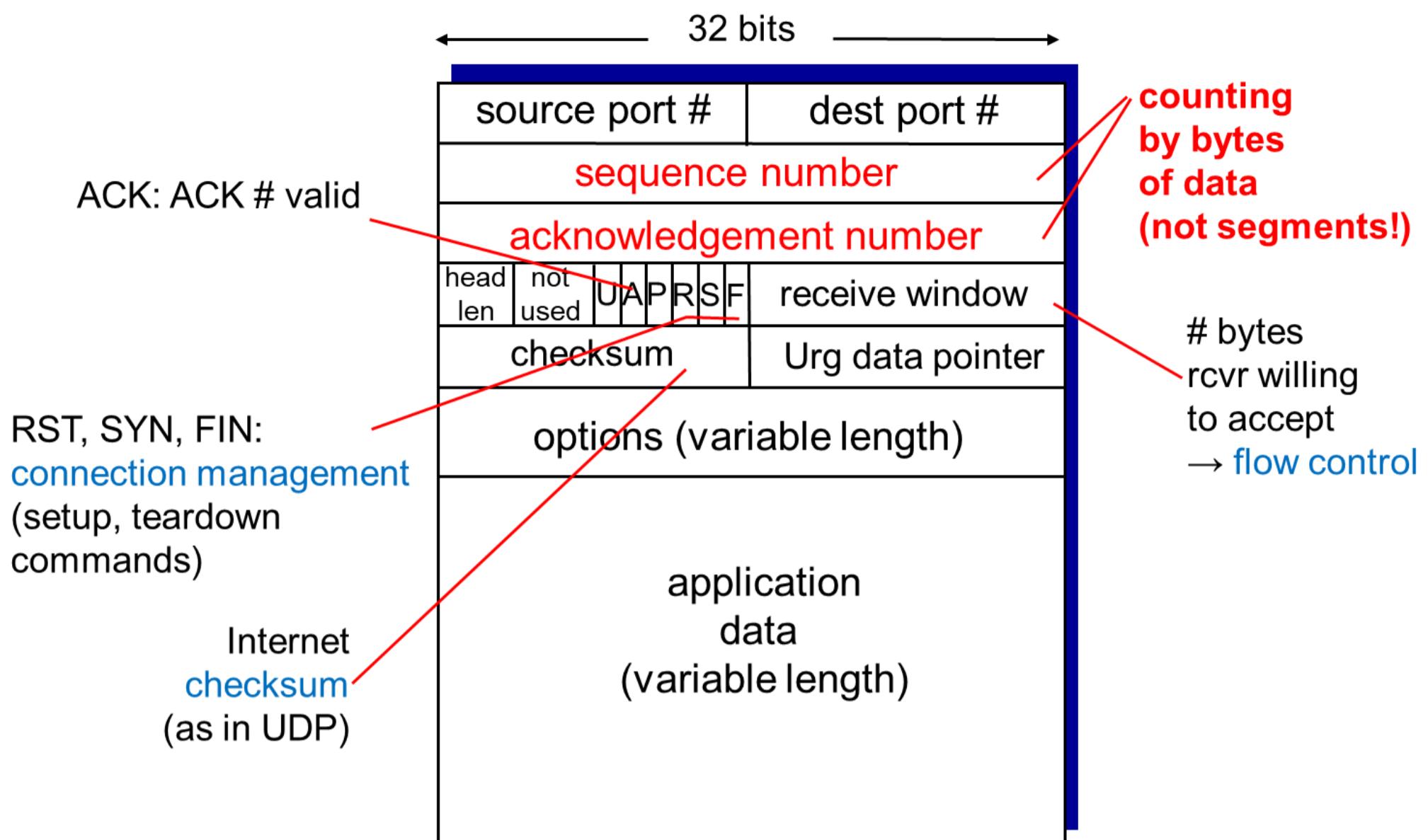


## TCP

特点：

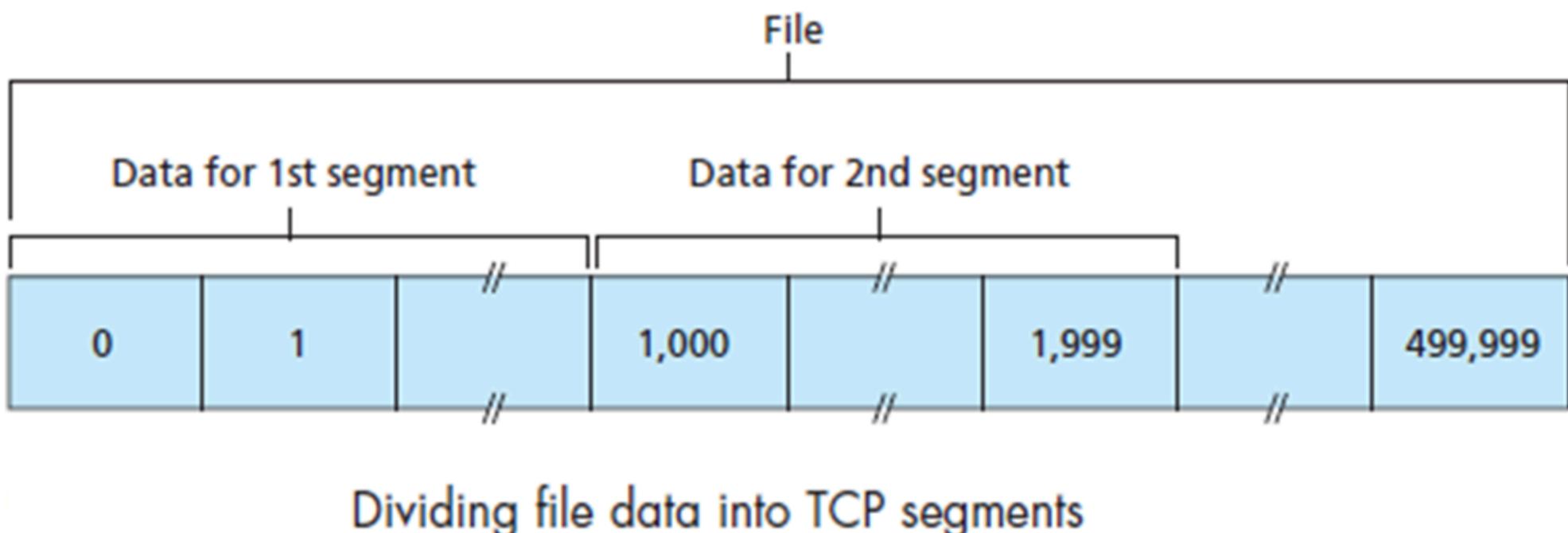
- 点对点 (point-to-point) : 单发送方单接收方
- 可靠按序的字节流
- 流水线: 拥塞和流控制, 设置窗口大小
- 全双工数据: 同一连接中的双向数据流
- 面向连接: 具有握手这一初始化双方状态的过程
- 流量控制

## 报文结构



3-62

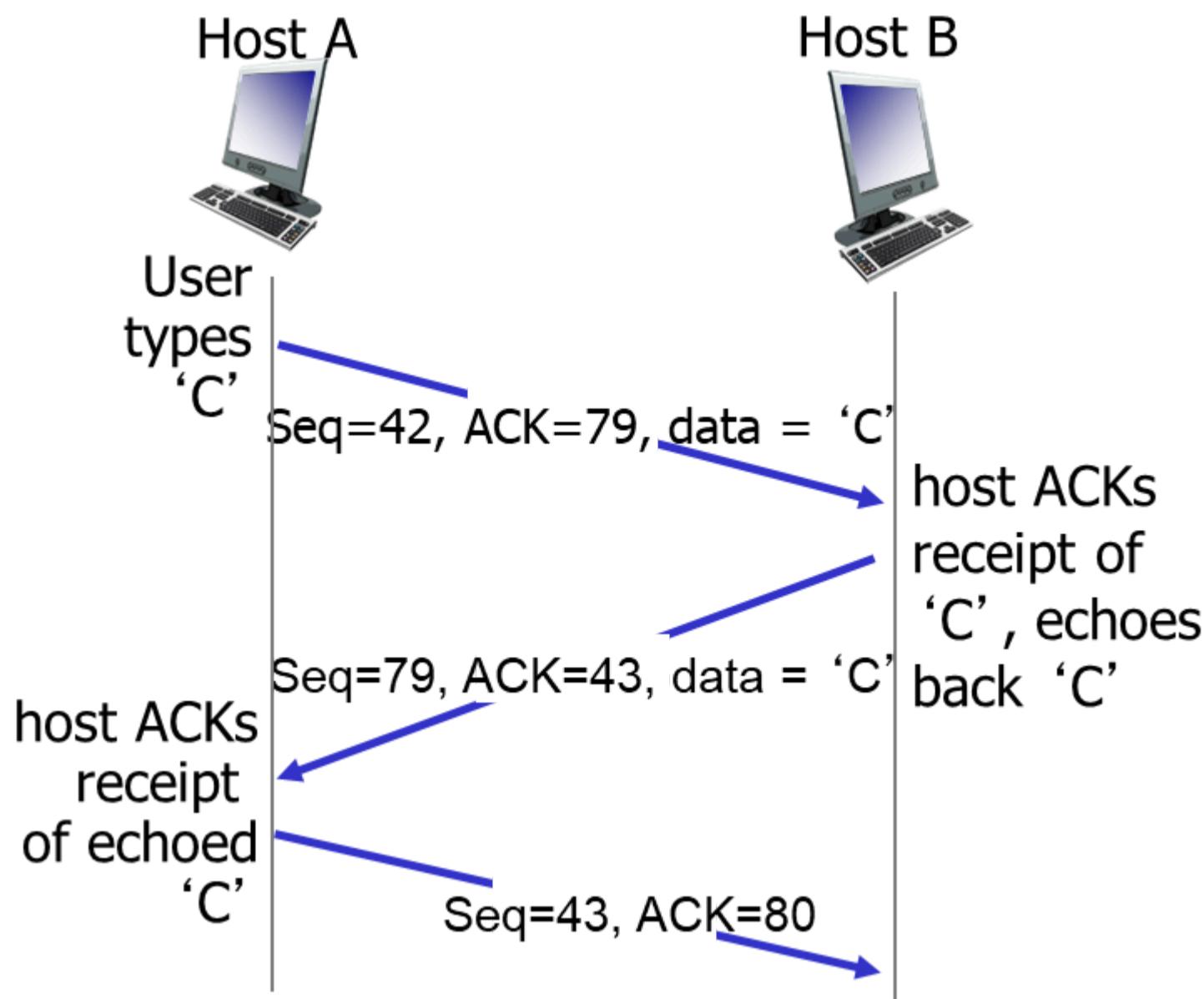
序号Seq：TCP把数据看成一个无结构的、有序的字节流，并将文件根据MSS (Maximum segment size) 分割成多个报文。报文的序号其实指的是该报文段首字节的字节流编号。例如文件大小为500000字节，MSS=1000字节，那么第一个报文的seq=0，第二个的seq=1000，共有500个报文。Seq只需要考虑数据部分的大小，各个首部的大小不需要考虑在内



确认号ACK：表示接收方希望收到的来自对方的下一个包的序号。

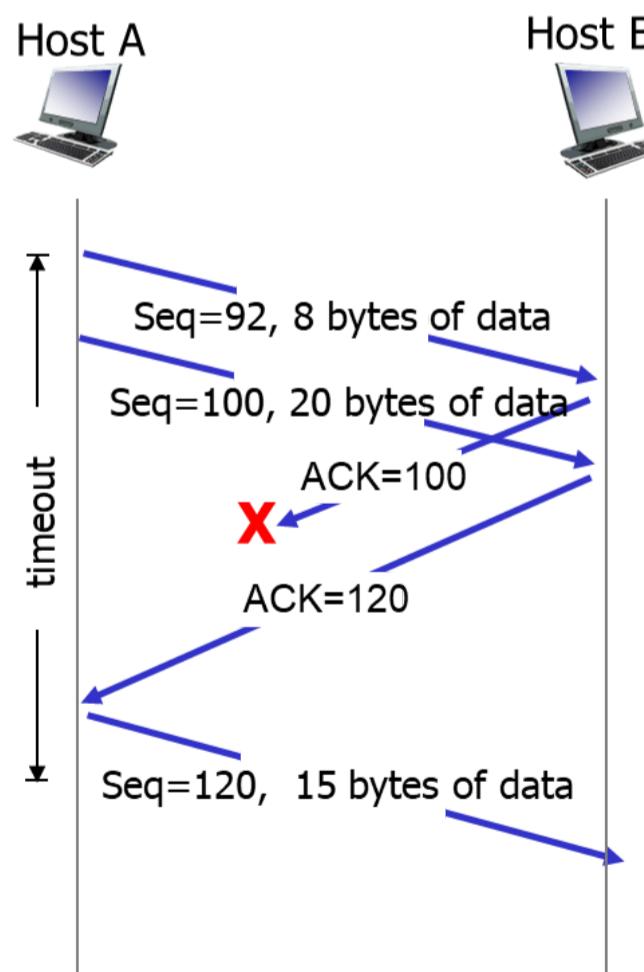
假如收到了0-535字节序号的报文，那么ACK=536

假如收到了0-535以及900-1000字节序号报文，ACK仍然=536



## simple telnet scenario

TCP接收方在收到多个数据包后，只发送一个确认（ACK）来确认所有已收到的数据包，称为提供累积确认（cumulative acknowledgment）



cumulative ACK

在TCP中如何估计RTT呢？不能太短也不能太长

需要使用**SampleRTT**测量从报文发出到收到ACK的时间来估算，多测量几次并取平均值。

每当得到一次测量结果时，按照以下公式更新：

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

通常 $\alpha$ 取0.125，也就是说新值通常由原本值和测量时间组合而成

此外还有RTT偏差  $\text{DevRTT}$ ，用于估算**SampleRTT**一般会偏离**EstimatedRTT**的程度：

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

$\beta$ 通常取0.25

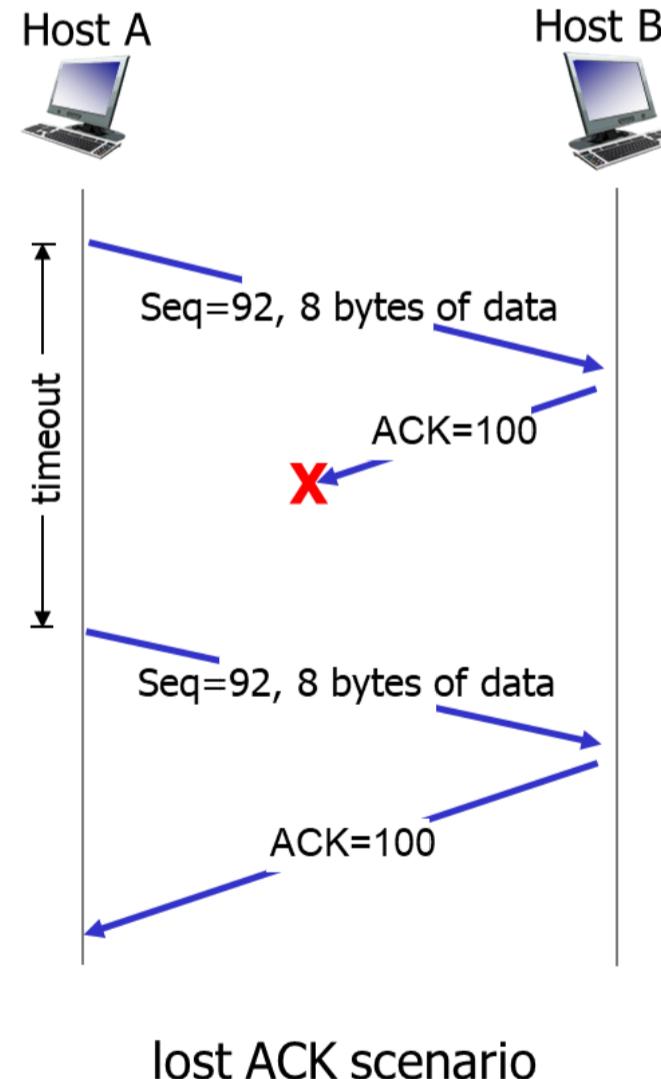
得到以上两个值后我们就可以用于计算TCP超时间隔。该间隔应该略大于**EstimatedRTT**又不能大太多，并且会根据偏离**EstimatedRTT**的程度自动调节，有

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

## 可靠数据传输

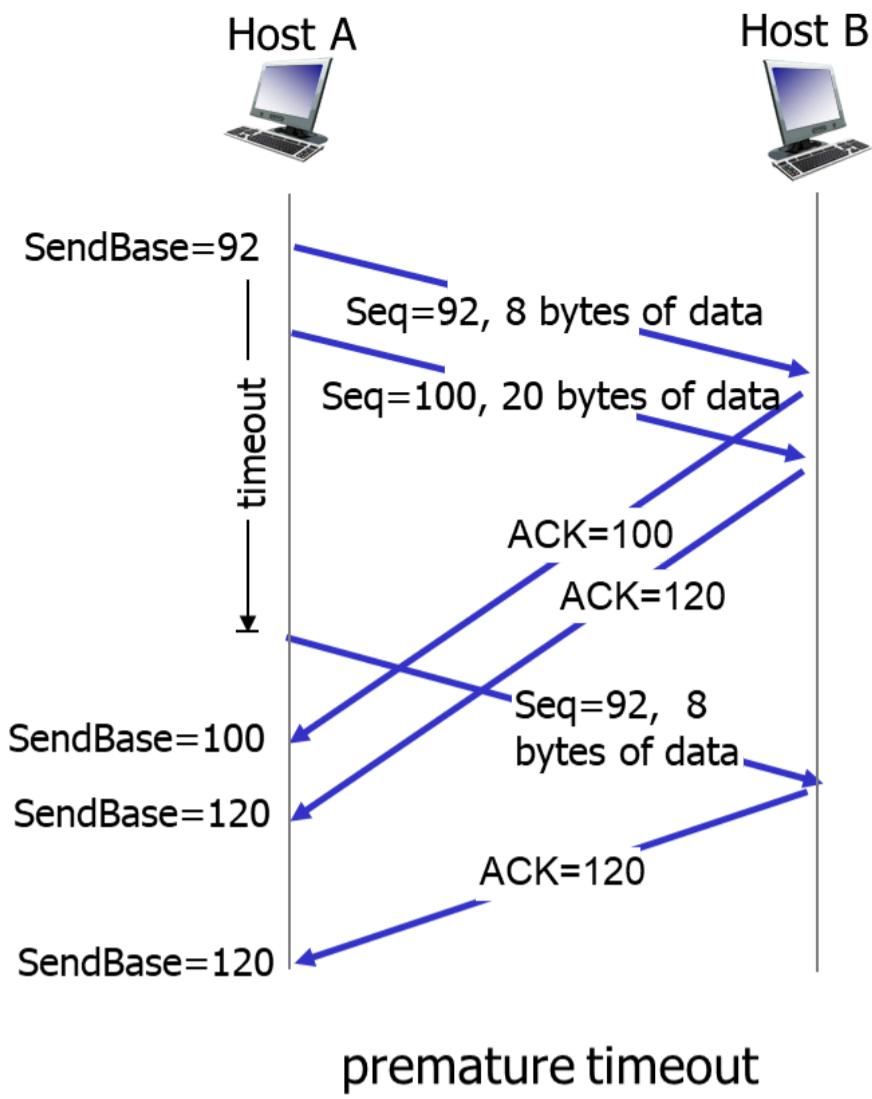
TCP通过以下机制在IP的不可靠服务之上提供可靠传输服务：

- 管道化报文
- 累积ACK
- 单重传计时器



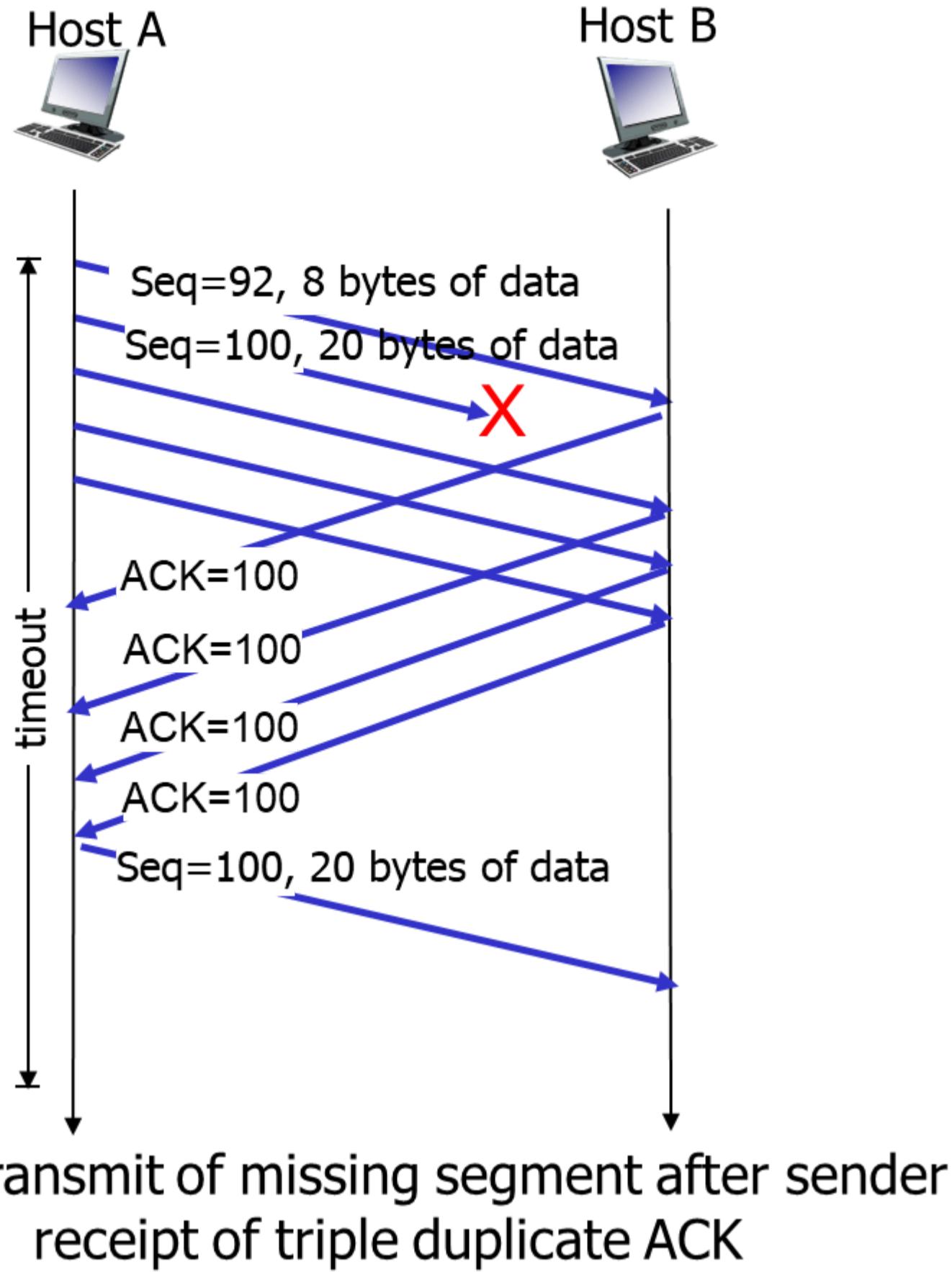
重传由以下事件之一触发：

- 超时事件
- 重复ACK



premature timeout

对于为发送方一个接一个地发送大量的报文段的情形。假如超时时间设置的比较长，那么包丢失后可能导致发送方在较长事件后才能重传丢失的分组。因此引入快速重传（fast retransmit）机制，如果TCP发送方接收到对相同数据的3个冗余ACK，它把这作为一种指示，说明跟在这个已被确认过3次的报文段之后的报文段已经丢失，直接重传。这一机制有效提高了TCP的效率。

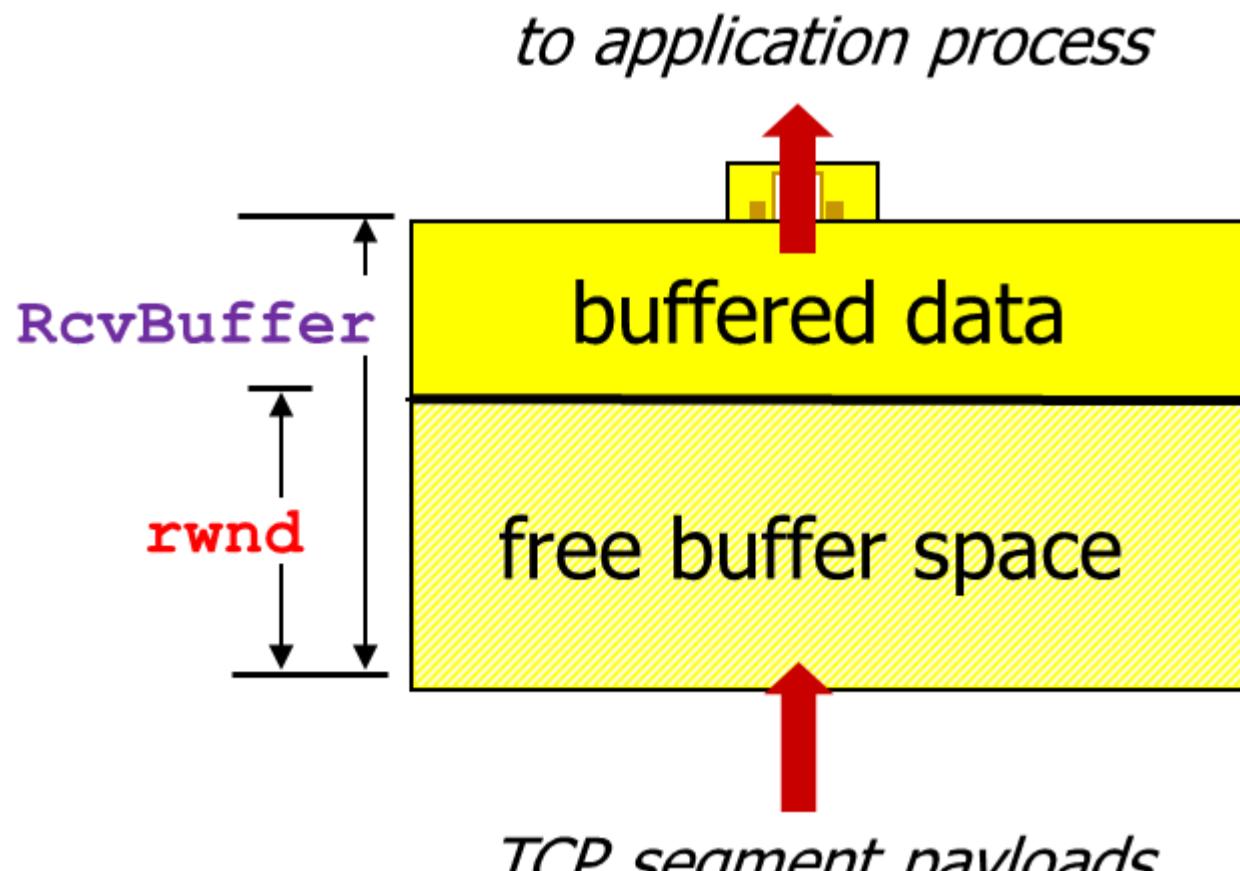


## 流量控制 Flow Control

由接收方控制发送方，防止发送方发出太多导致接收方缓冲区溢出

RcvBuffer: 接收方缓冲区大小

接收方发送ACK报文时，其内包含rwnd这一值，用于标识接收方可用缓冲区空间。发送方接收到后就自行控制发送数量，防止溢出。



## *receiver-side buffering*

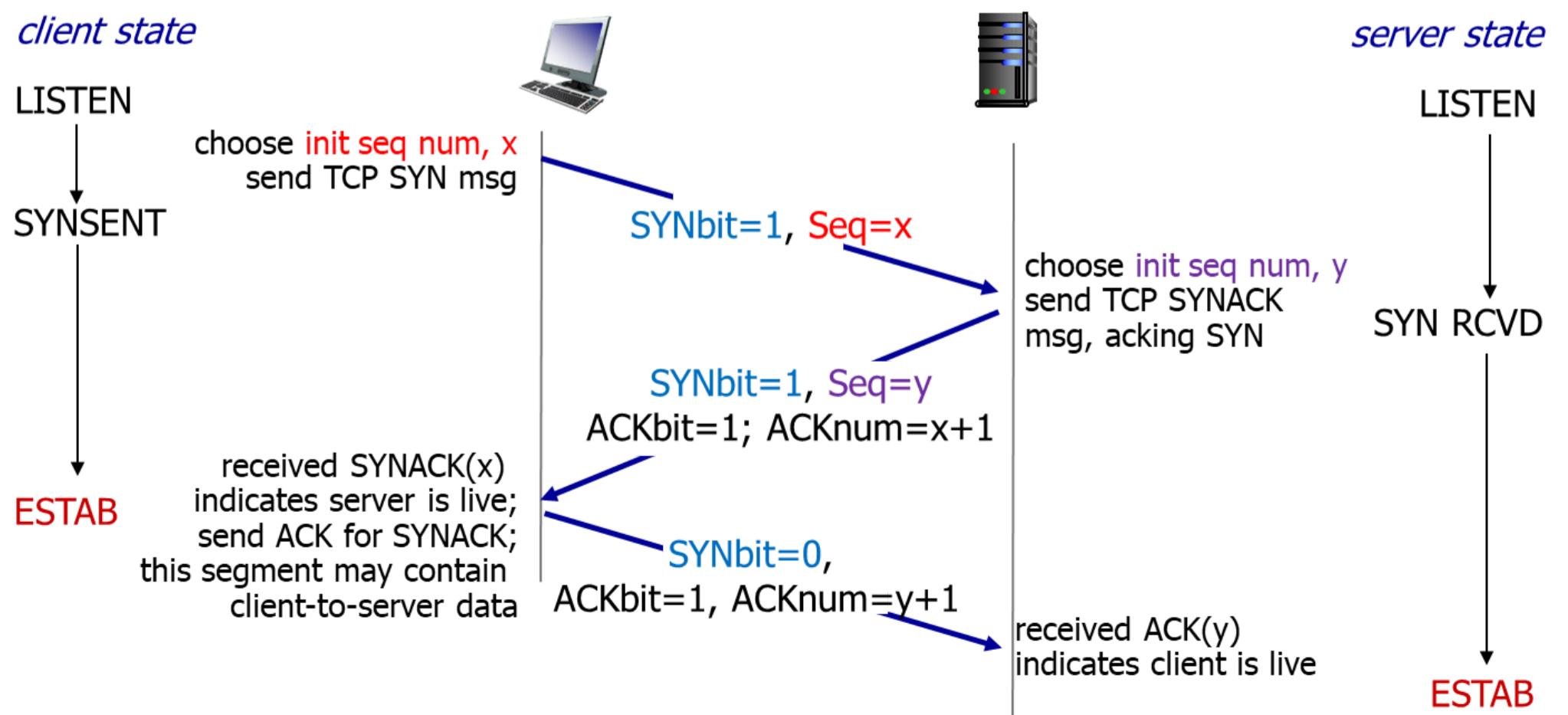
## 连接管理 Connection Management

在数据交换开始前，发送方和接收方通过握手（handshake）来：同意建立连接 agree to establish connection，协商连接参数 agree on connection parameters

3握建立连接 3-way handshake：

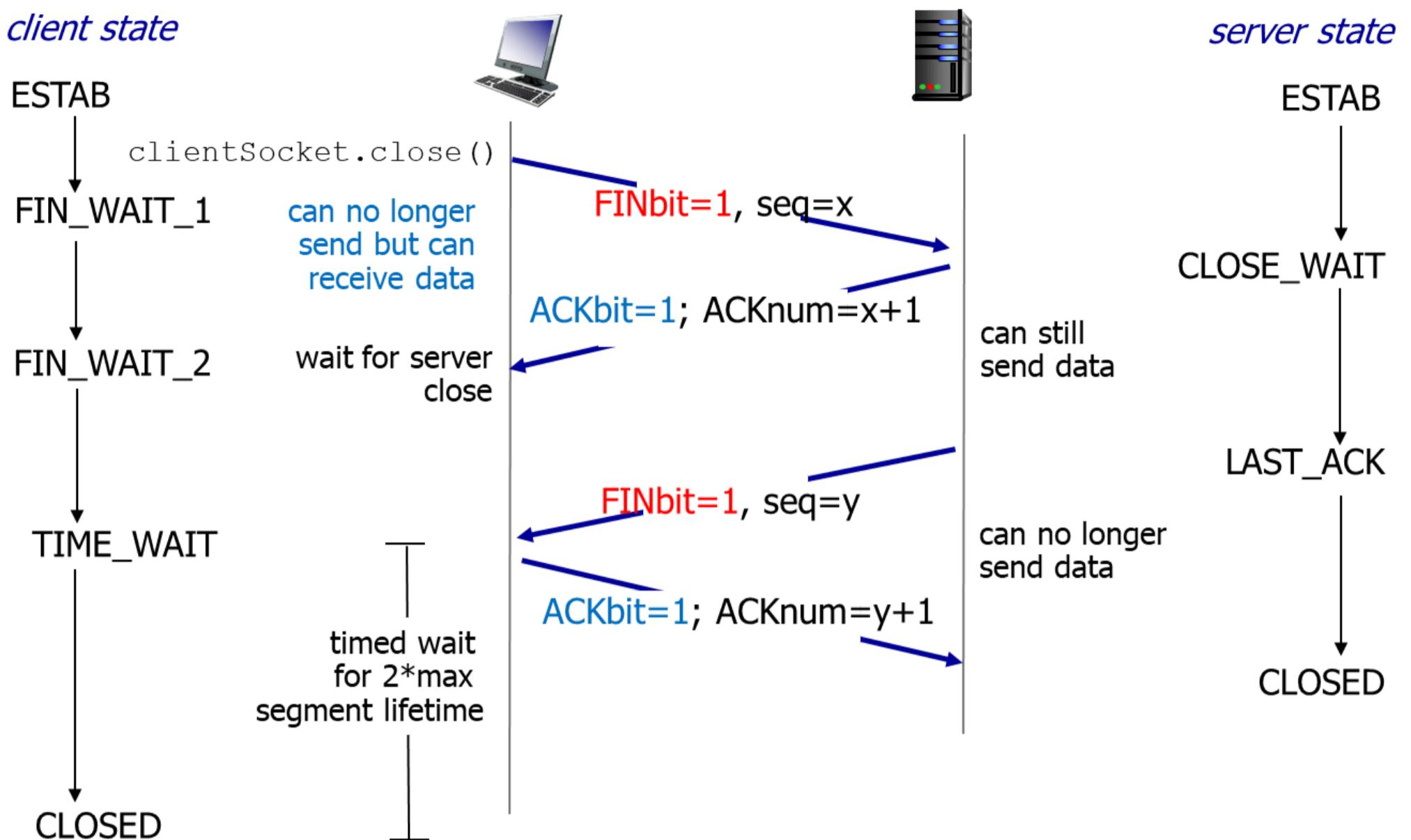
1. 客户发送SYN=1，表示建立连接。选择一个Seq=x，发送该报文给服务端
2. 服务端接收到报文后，为该连接分配变量和缓存。随后也发送SYN=1，选择Seq=y，ACK=x+1，发送给客户端
3. 客户端也分配变量和缓存，随后发送SYN=0（连接已经建立），Seq=x+1，ACK=y+1

由于连接建立报文的数据大小都仅为1字节，因此序号都只加1



4挥手关闭连接：

1. 客户发送FIN=1, 表示结束连接, Seq=x
2. 服务端发送ACK=x+1
3. 服务端发送FIN=1, Seq=y
4. 客户端发送ACK=y+1



## 拥塞控制 Congestion Control

- 太多的数据通过网络发送
- 和流量控制有所不同
- 导致丢包和高时延

两种控制方法：

端到端 end-to-end：

- 没有网络层给出的反馈
- 终端根据loss和delay来推断拥塞
- TCP所采用的方法

网络辅助拥塞控制 network-assisted congestion control

- 路由器提供协助
- 使用一个bit位来指示链路中的拥塞情况

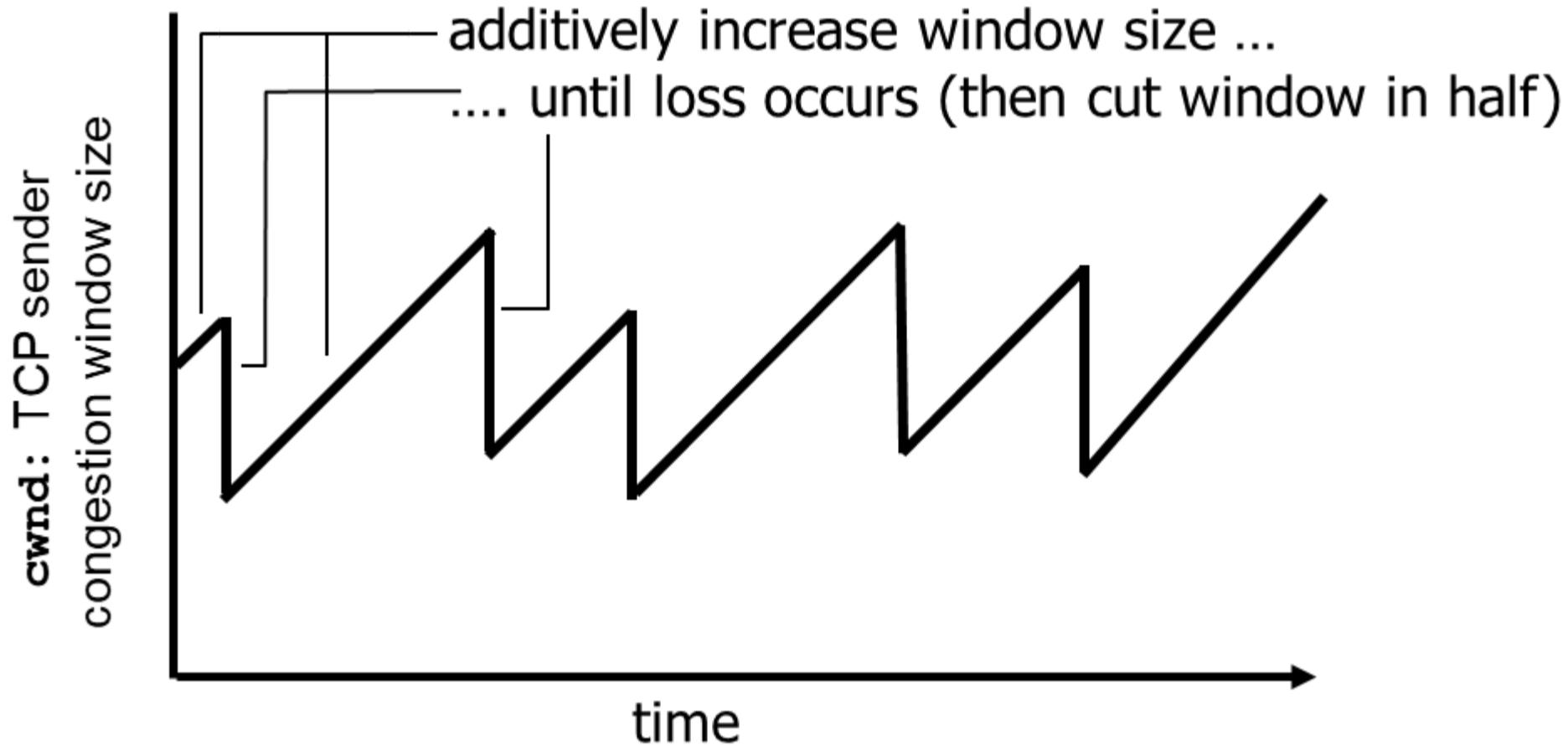
# TCP拥塞控制

## 基本思想

一个基本思想：TCP持续增加传输速率（窗口大小），探测可用带宽直到发生丢包。

以下算法可称为AIMD(**Additive Increase Multiplicative Decrease**)：

- 持续增加cwnd (拥塞窗口)
- 发送丢包后迅速减小cwnd



与流量控制中的rwnd类似，cwnd也是发送方对自身发送行为的限制，因此其每次选择发送的数据大小 (**LastByteSent - LastByteAcked**)

都满足：

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

如果只考虑拥塞控制，那么根据TCP发送过程，即发送数据随后等待确认 (RTT内完成)，发送速率如下：

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

## TCP Tahoe

一种基于丢失的拥塞控制机制

如何检测到丢失：

1. RTT超时
2. 3个重复ACK

这一机制通过两个阶段来调整cwnd

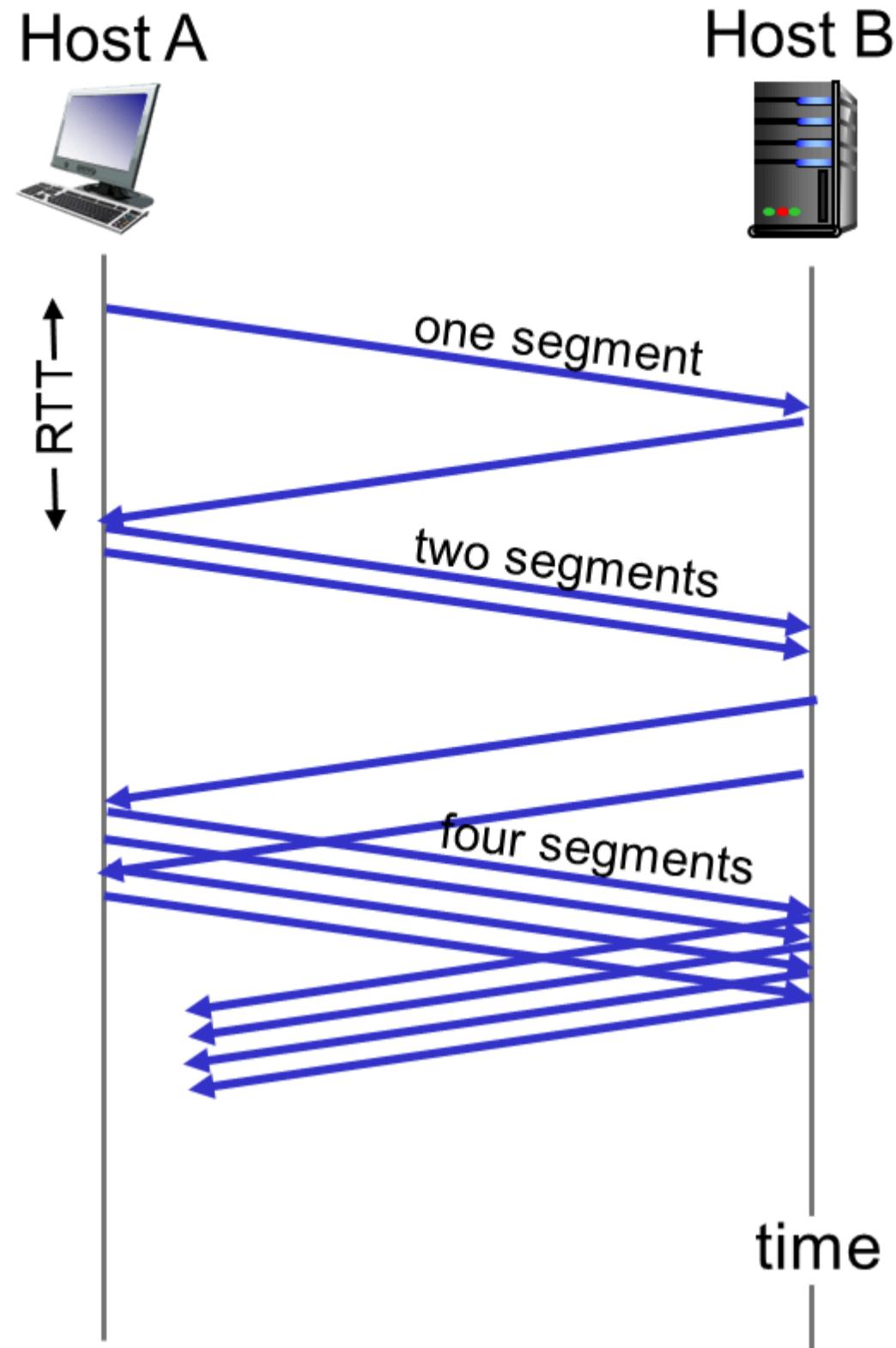
## Slow Start

在传输刚开始和发送超时后使用

1. 一开始时设置cwnd=1MSS (即一个报文)，然后逐步增加传输的数量，避免网络带宽被浪费

2. 每个RTT都翻倍cwnd的值，其实就是每次收到ACK后翻倍cwnd

这一方法会使得传输速率初始缓慢但成指数级增长



## CA (Congestion Avoidance)

这一阶段慢速增加cwnd，避免拥塞发生

每个RTT都慢速增加1MSS的cwnd，成线性增长

那么TCP如何决定什么时候SS什么时候CA呢？

其维护一个ssthresh变量，用于作为阶段区分。

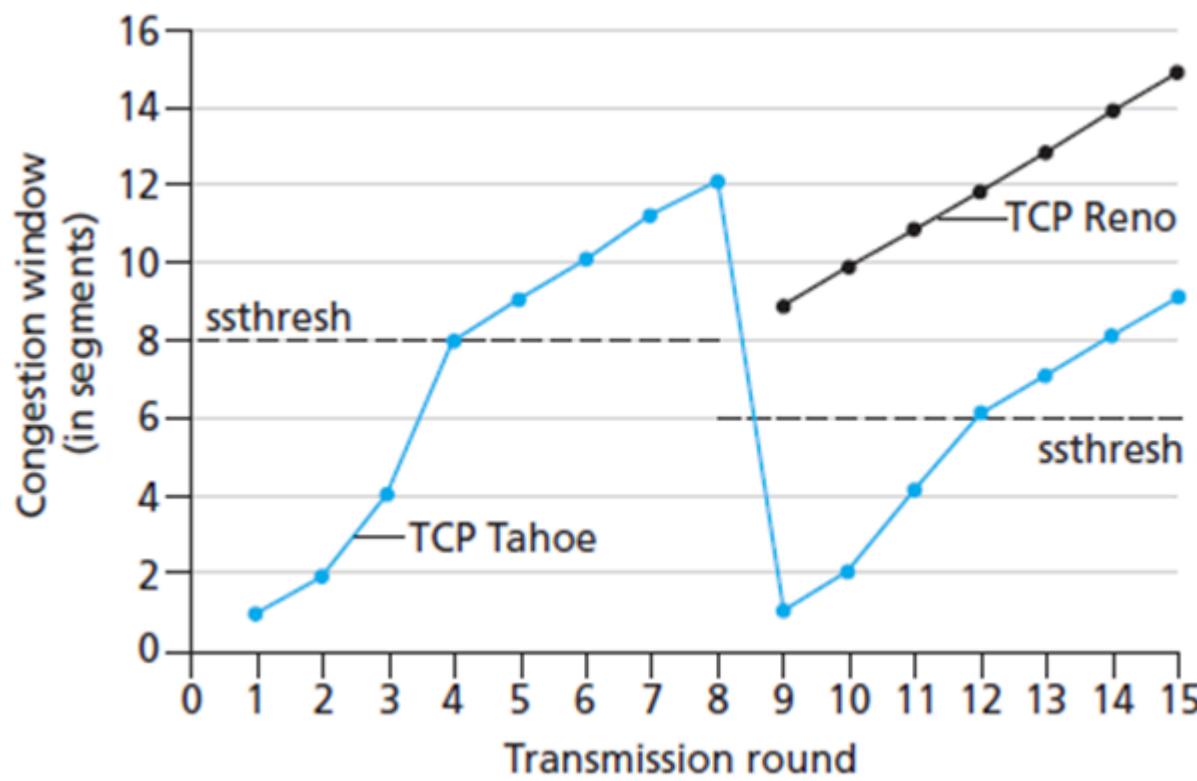
若发生了ssthresh通常设为上一次发送拥塞时cwnd值的一半。

cwnd < ssthresh，执行SS

cwnd > ssthresh，执行CA

若发送了拥塞则：

- 由丢包导致：cwnd降到1
- 由重复ACK导致：cwnd降到一半



Evolution of TCP's congestion window (Tahoe and Reno)

## TCO Reno

Tahoe算法对于丢包过于敏感，即使是偶然发送的小丢包也会导致吞吐量急速下降

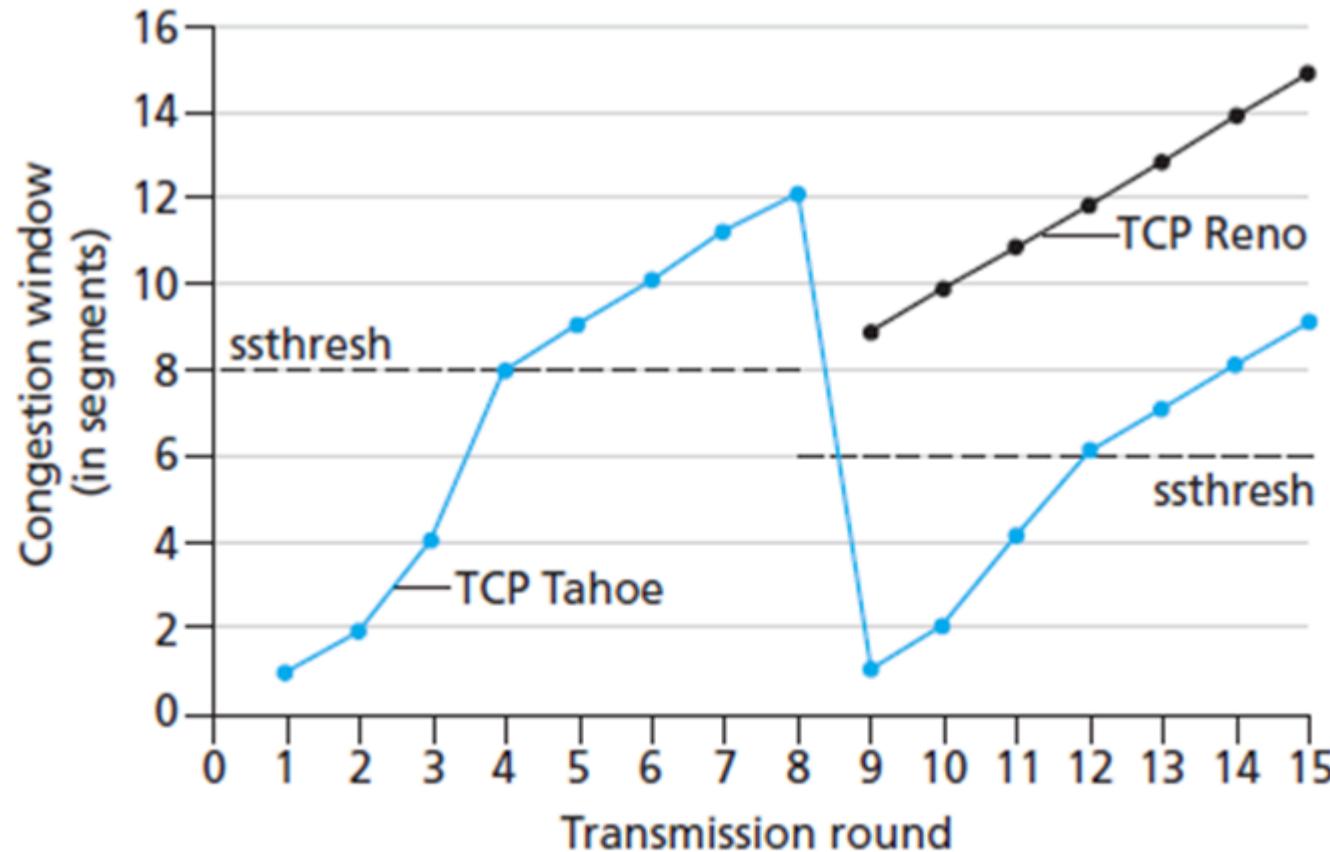
因此Reno算法引入了快速恢复（Fast Recovery）算法，被广泛使用

## Fast Recovery

若发送丢包， $ssthresh = cwnd/2$

$cwnd = ssthresh + 3$ ，随后线性增加

值得注意的是，Reno算法实际上综合了SS, CA以及FR。其大部分时间都在执行CA和FR



Evolution of TCP's congestion window (Tahoe and Reno)

## 对TCP吞吐量的宏观描述

对于一个发生丢包时窗口大小为W的连接，由于其通常在 $2/W$ 和W之间波动，平均窗口大小为 $3/4W$

平均吞吐量为 $3/4W$ 每RTT

## 公平性

TCP通过拥塞控制的AIMD算法保证了各个连接平等地使用网络带宽

而UDP因为缺乏拥塞控制机制，各个应用程序将最大限度争抢带宽

## 4. Network Layer: Data Plane

### 概述

网络层的作用是将分组从一个主机发送到另一个主机

各个路由器和主机都包含网络层协议

两个功能：

- 转发 (forwarding)：将数据包从路由器输入移动到正确的路由器输出。也就是在路由器之中选择适当出口的过程
- 路由选择(routing): 决定从起点到终点经过的路由。也就是对路由路线的选择。

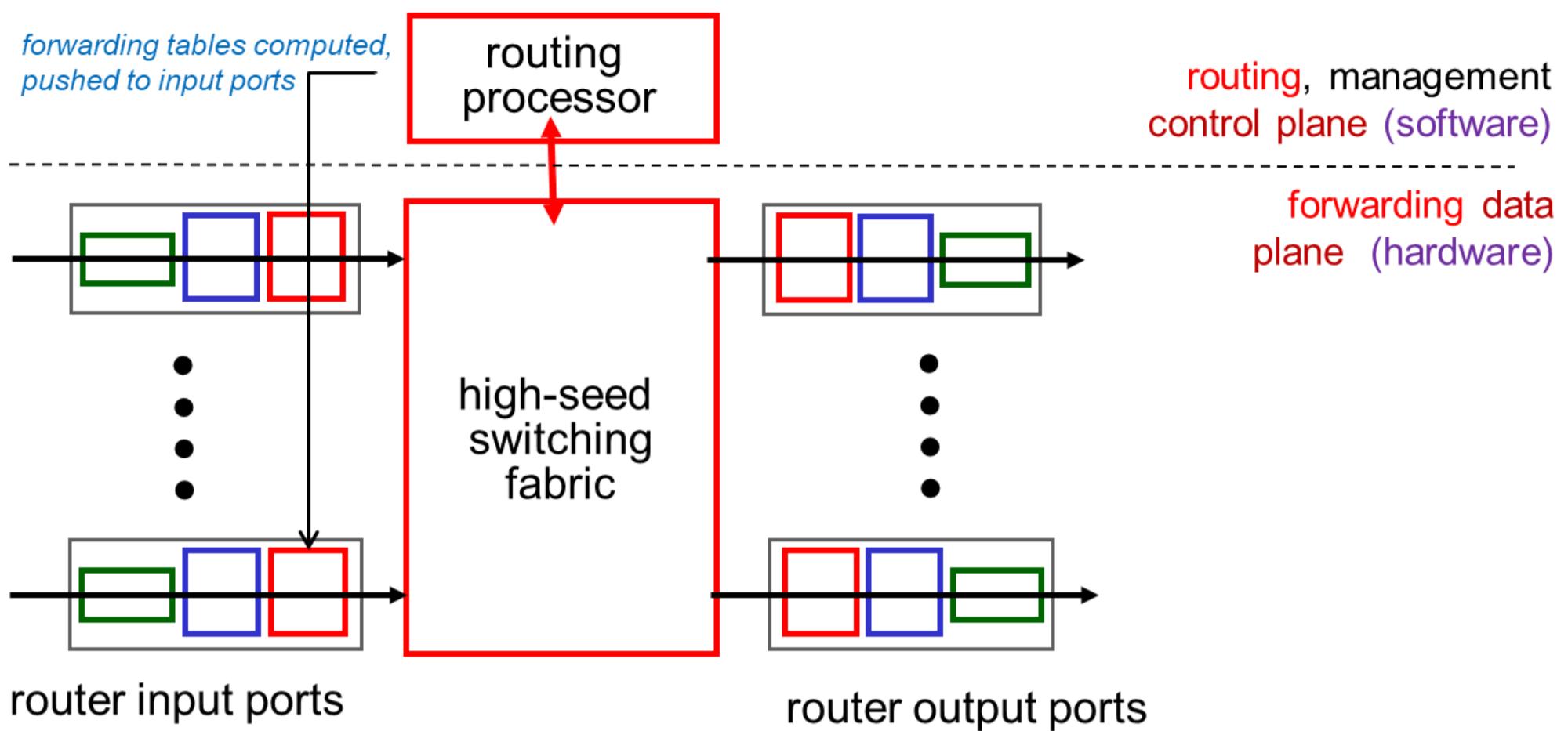
对应以上两个功能，网络层分成了两个平面：

- 数据层面 (Data Plane)：
  - 本地的，各个路由负责的功能
  - 确定数据包如何从路由输入端口到达输出端口
  - 转发功能
- 控制层面 (Control plane)：
  - 网络范围功能
  - 确定从源主机到目的主机之间的路由路径
  - 路由选择功能
  - 两种方法：
    - 传统路由算法 *traditional routing algorithms*: 路由器中实现，路由器之间通过路由算法交互
    - SDN方法：远程服务器中实现，远程控制器和本地控制代理CA交互

网络服务模型 Network service model:

### 路由器内部

路由器需要执行两个功能：路由选择和转发



## 输入端口处理与基于目的地转发

基于目标的转发：仅基于目标 IP 地址的转发（传统）

最长前缀匹配：在查找给定目标地址的转发表条目时，使用与目标地址匹配的最长地址前缀。

Destination Address Range	Link interface
11001000 00010111 00010*** **** *****	0
11001000 00010111 00011000 **** *****	1
11001000 00010111 00011*** **** *****	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001 0

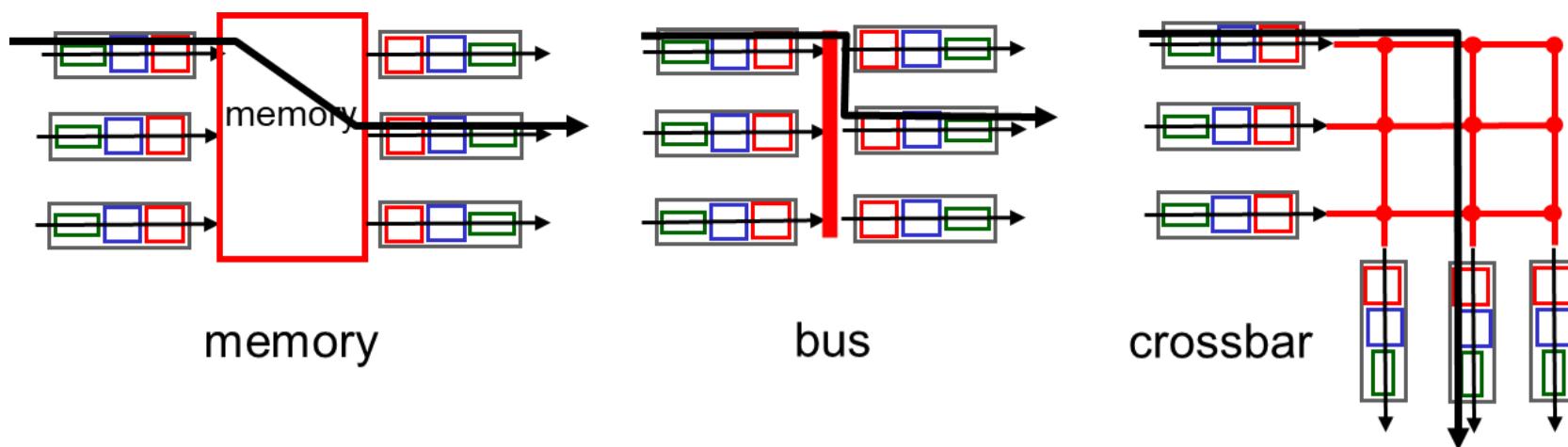
DA: 11001000 00010111 00011000 10101010 |

## 交换 switching

将数据包从输入缓冲区传输到适当的输出缓冲区的过程

交换速率switching rate：数据包从输入到输出的速度

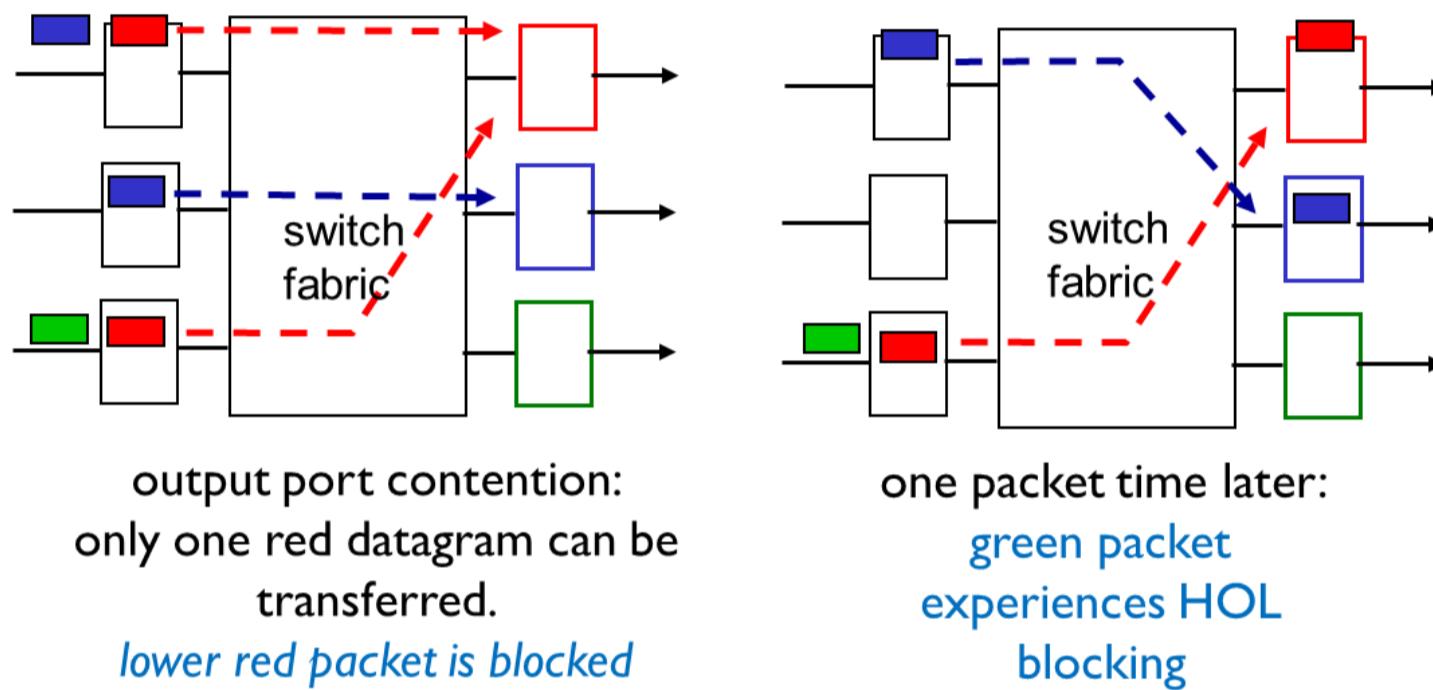
三种交换结构：switching fabrics



## 输入排队

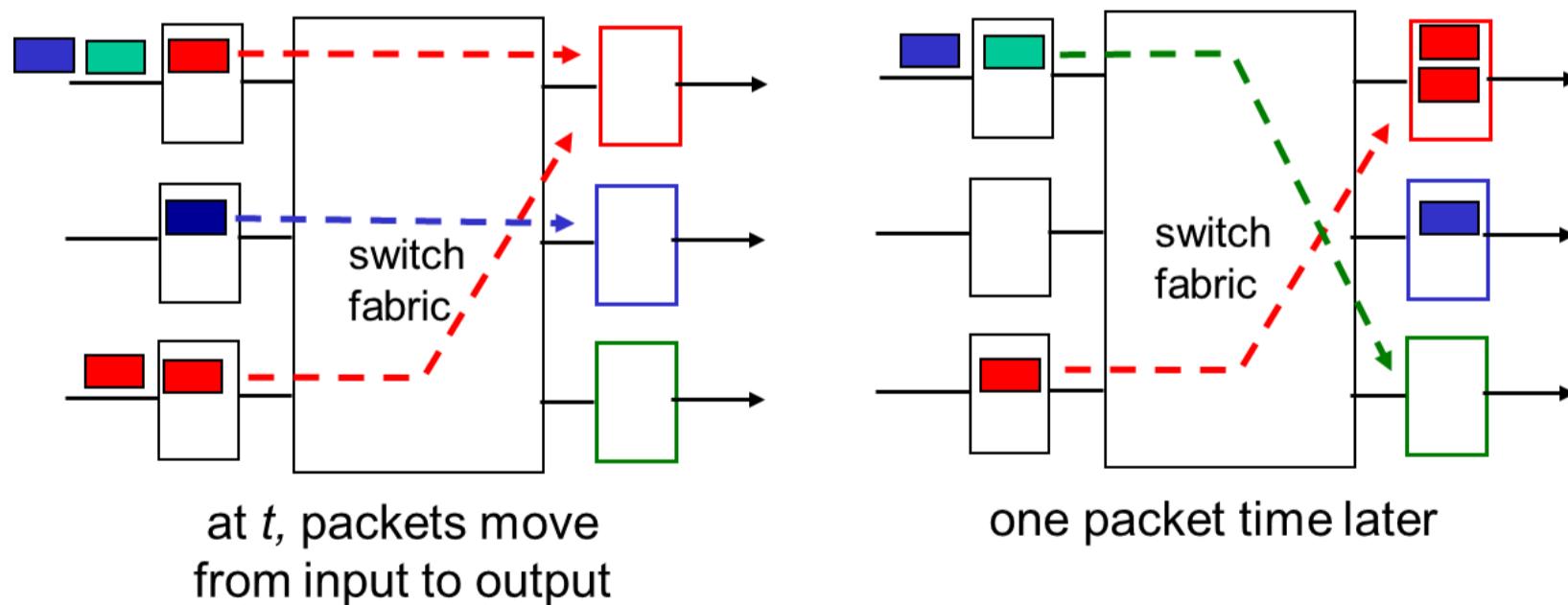
交换速度比输入线路速度慢的话，到达的数据包无法无时延通过，那么就会在输入端口排队，甚至可能出现缓存区溢出导致丢包

线路前部 (Head-Of-the-Line, HOL) 阻塞：一个输入队列中排队的分组必须等待通过交换结构发送（即使输出端口是空闲的），因为其被线路前部的另一个分组阻碍



## 输出排队

输出同样会出现排队，当交换速度比输出线路速度快，那么就会在输出端口排队，同样可能出现缓存区溢出导致丢包

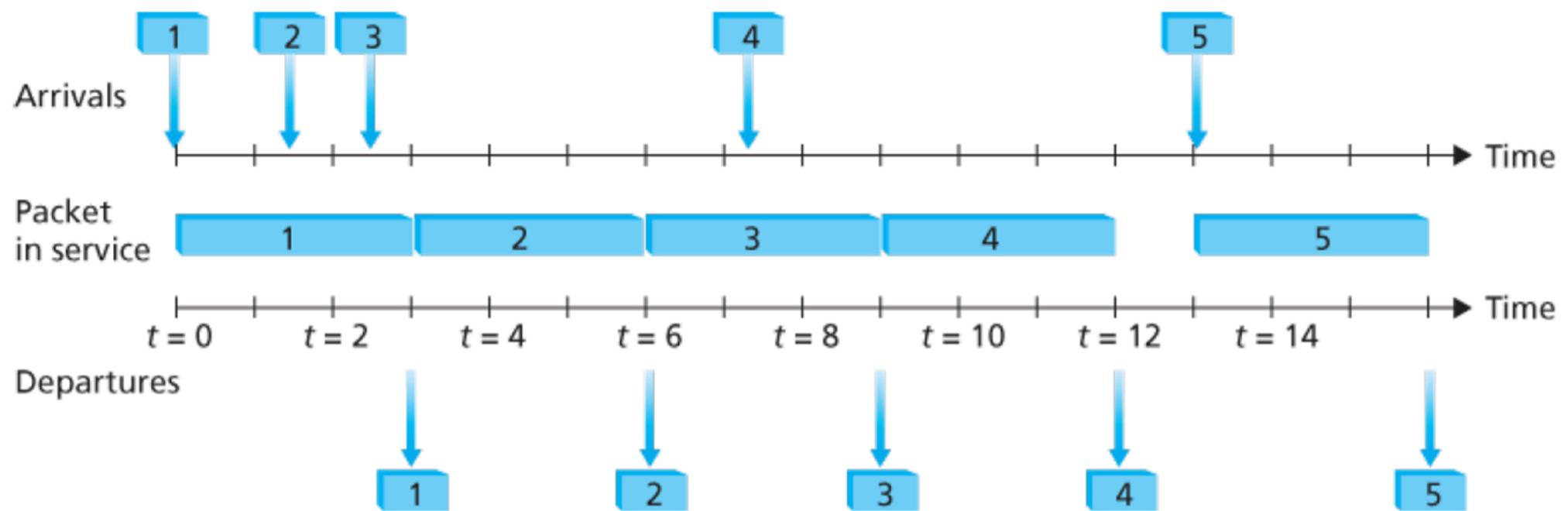


## 分组调度

使用不同调度规则来选择下一个发送的数据包

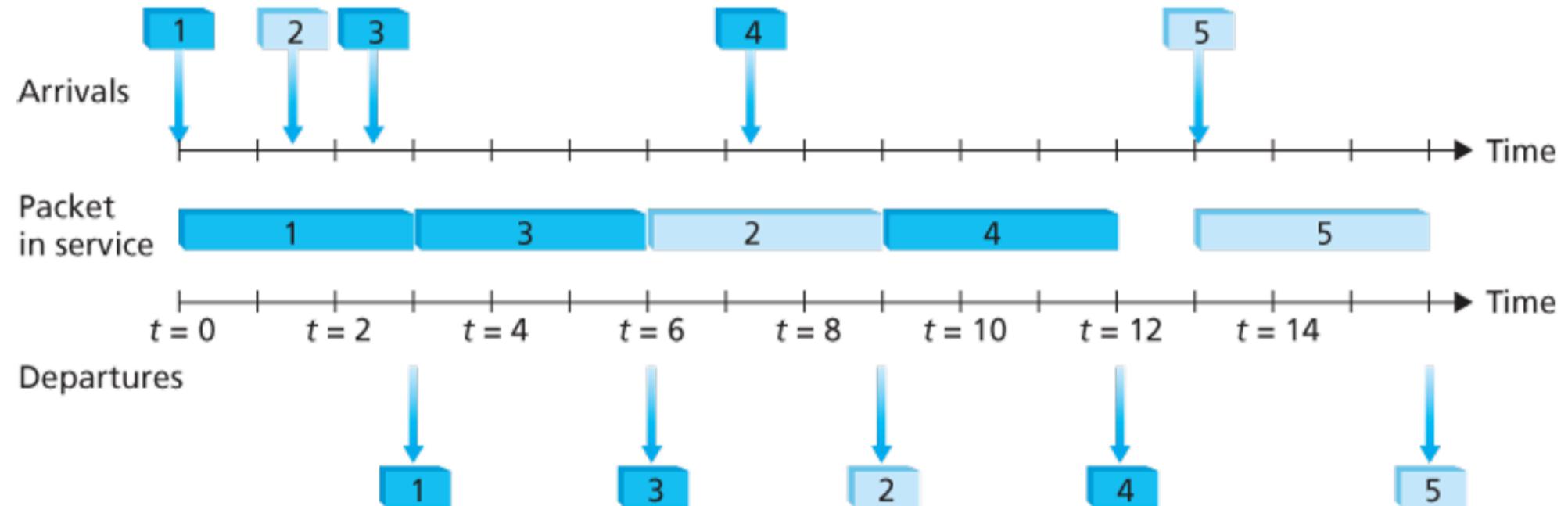
## FIFO

First In First Out: 以到达的顺序发送



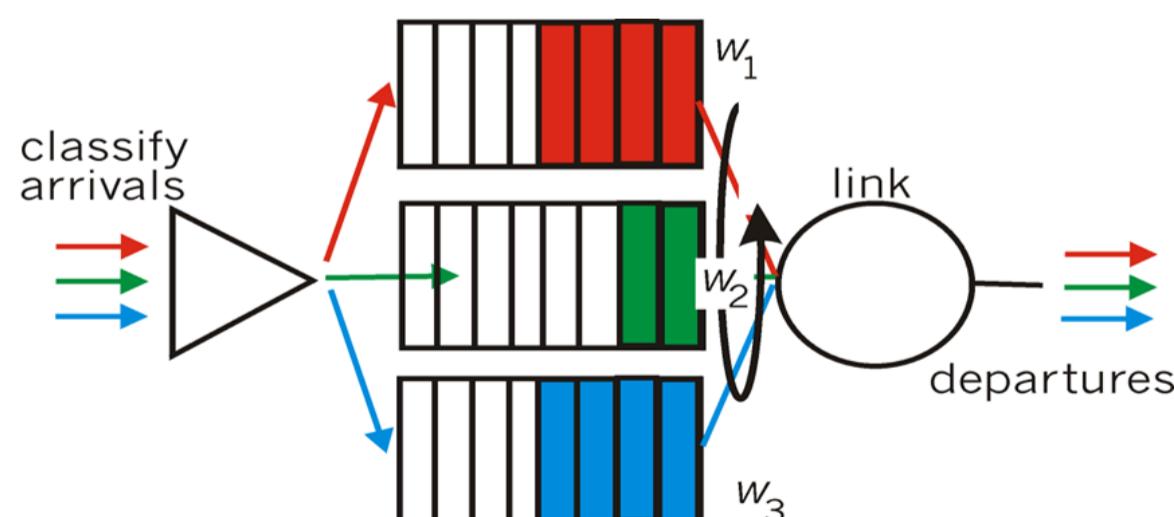
## 优先权 Priority

为数据包指定不同的优先级，优先发送高优先级的数据包



## 加权公平队列 Weighted Fair Queueing (WFQ)

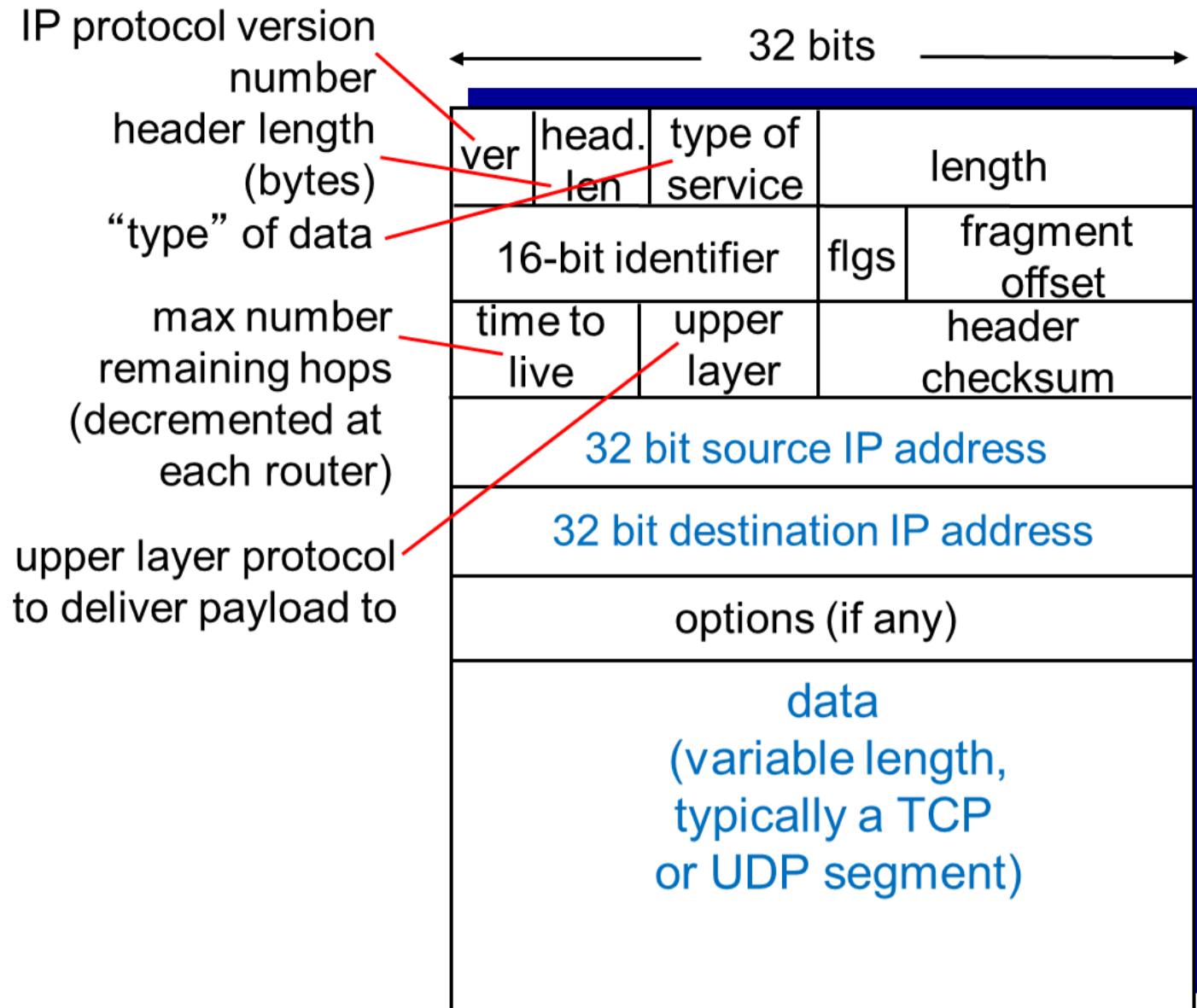
每个类在每个周期中获得加权的服务量



## IP协议

### 首部

IP首部通常占20字节，TCP首部也是20字节，总共占40字节，随后再加上应用层数据长度



## 分片

不同的链路层协议能承载不同长度的数据包，例如以太网可以承载1500字节。

一个链路层帧能承载的最大数据量叫作最大传送单元 (Maximum Transmission Unit, MTU)

因此，对于比较大的IP数据包，将会被分成多个小数据包 divided into several smaller datagrams

当达到目的地后再根据标识ID、标志flag和片偏移offset重新组装 reassemble

分割后的每个片都有初始数据包的源地址、目的地址与标识号，而片中除了最后一个结尾的片有0标志，其他均为1标志，并通过偏移确定该片放置于哪个位置 (就像拼图)

### example:

- ❖ 4000 byte datagram
- ❖ MTU = 1500 bytes

	length =4000	ID =x	fragflag =0	offset =0	
--	-----------------	----------	----------------	--------------	--

*one large datagram becomes several smaller datagrams*

1480 bytes in data field

	length =1500	ID =x	fragflag =1	offset =0	
--	-----------------	----------	----------------	--------------	--

offset =  
 $1480/8$

	length =1500	ID =x	fragflag =1	offset =185	
--	-----------------	----------	----------------	----------------	--

	length =1040	ID =x	fragflag =0	offset =370	
--	-----------------	----------	----------------	----------------	--

## IPv4

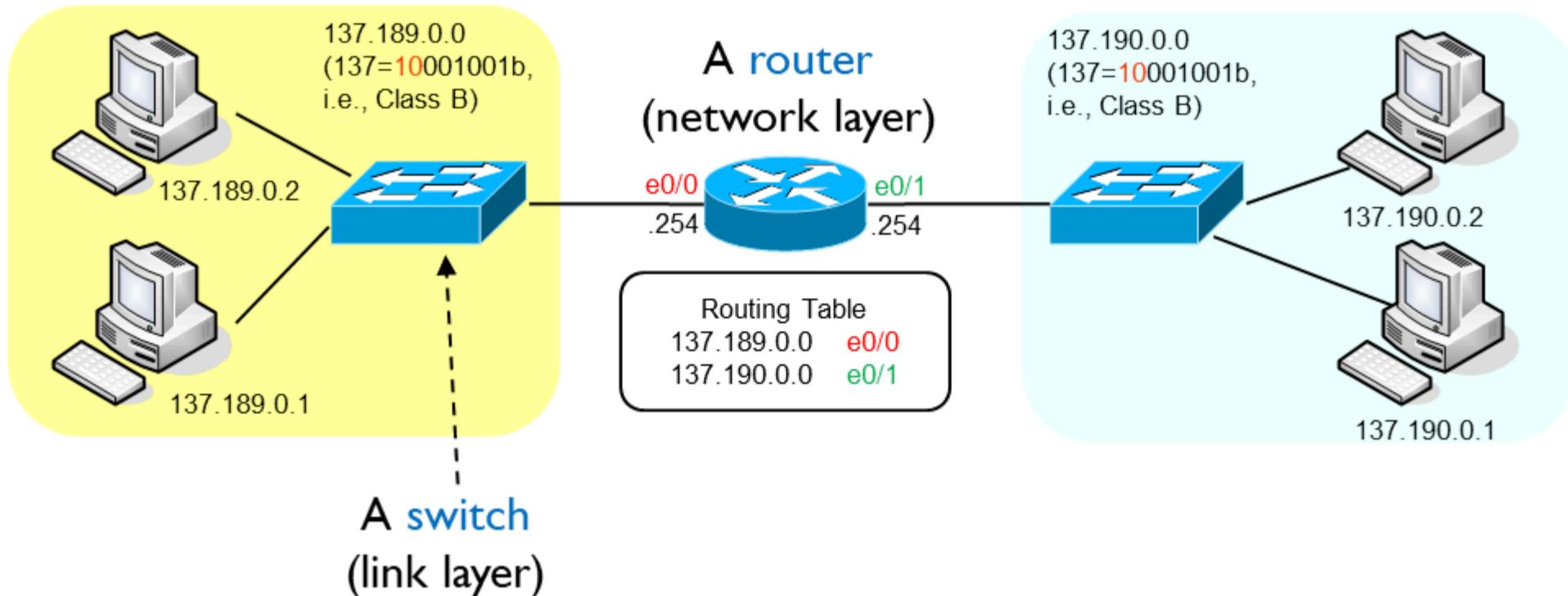
每个IP地址长度为32比特（等价为4字节），每个字节用它的十进制形式书写，各字节间以句点隔开

每台主机和路由器上的每个接口，都必须有一个全球唯一的IP地址（在NAT后面的接口除外）

子网掩码：如223.1.1.0/24,其中的/24记法，有时称为子网掩码（network mask），指示32比特中的最左侧24比特定义了子网地址

具有相同子网地址的设备连接在一起，可以直接通过链路连接。

路由器根据路由表中IP地址范围进行转发，子网掩码即可看成IP地址范围指示符



因此，在以上规则下，距离近的IP地址将共享同一固定长度前缀，例如前8, 16, 24，这称为分类编址（classful addressing）。但这一方法地址分配的灵活性不够。

Class A IP Address:	<table border="1"><tr><td>01010101</td><td>10101000</td><td>00000001</td><td>00000001</td></tr></table>	01010101	10101000	00000001	00000001	For large organizations
01010101	10101000	00000001	00000001			
Class B IP Address:	<table border="1"><tr><td>10010011</td><td>10101000</td><td>00000001</td><td>00000001</td></tr></table>	10010011	10101000	00000001	00000001	For medium organizations
10010011	10101000	00000001	00000001			
Class C IP Address:	<table border="1"><tr><td>11000101</td><td>10101000</td><td>00000001</td><td>00000001</td></tr></table>	11000101	10101000	00000001	00000001	For small organizations
11000101	10101000	00000001	00000001			
Class D IP Address:	<table border="1"><tr><td>11100101</td><td>10101000</td><td>00000001</td><td>00000001</td></tr></table>	11100101	10101000	00000001	00000001	Multicast addresses (local network)
11100101	10101000	00000001	00000001			
Class E IP Address:	<table border="1"><tr><td>11110000</td><td>10101000</td><td>00000001</td><td>00000001</td></tr></table>	11110000	10101000	00000001	00000001	For research purposes
11110000	10101000	00000001	00000001			

Note: The number of network bits is implicit to a given IP address via **prefix coding**.

所以引入了无类别域间路由Classless Inter-Domain Routing (CIDR):

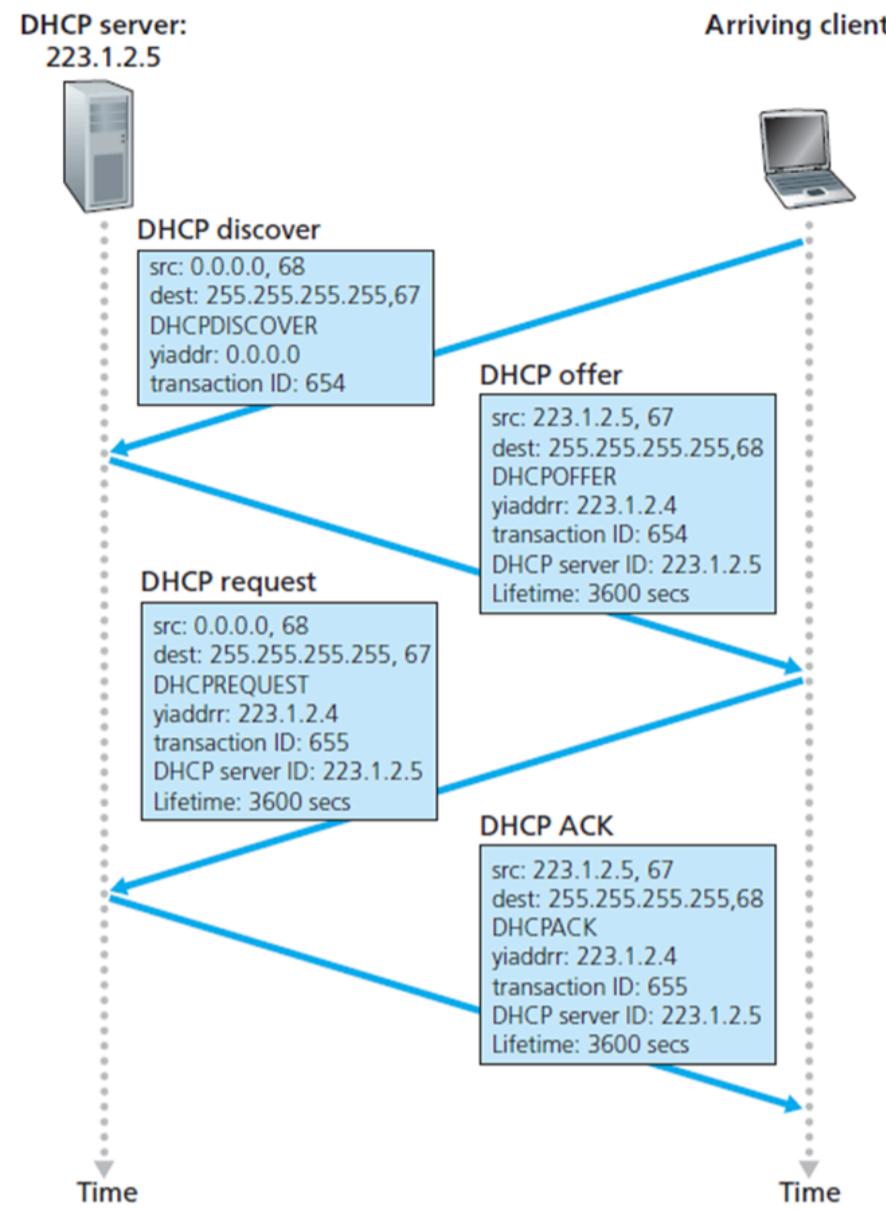
- 取消了IP地址中固定前缀限制
- 可以自由分配可变长掩码，如/17
- 路由器需要同时交换子网掩码，因为长度可变，无法直接通过IP地址推断得出

## DHCP

动态主机配置协议（Dynamic Host Configuration Protocol）：新主机在加入网络时从网络服务器动态获取其IP地址

1. DHCP服务器发现 DHCP discover: 主机向255.255.255.255地址进行广播，发现一个要与其交互的DHCP服务器

2. DHCP服务器提供 DHCP offer: 服务器收到后向子网内所有节点广播, 仍用255.255.255.255。广播报文包含服务器可提供的IP地址以及可用期限
3. DHCP请求 DHCP Request: 主机从收到的广播中选择一个进行响应, 告知自己将使用此地址
4. DHCP ACK: 服务器进行响应



DHCP 还能获取第一跳路由器地址, DNS服务器地址, 网络掩码

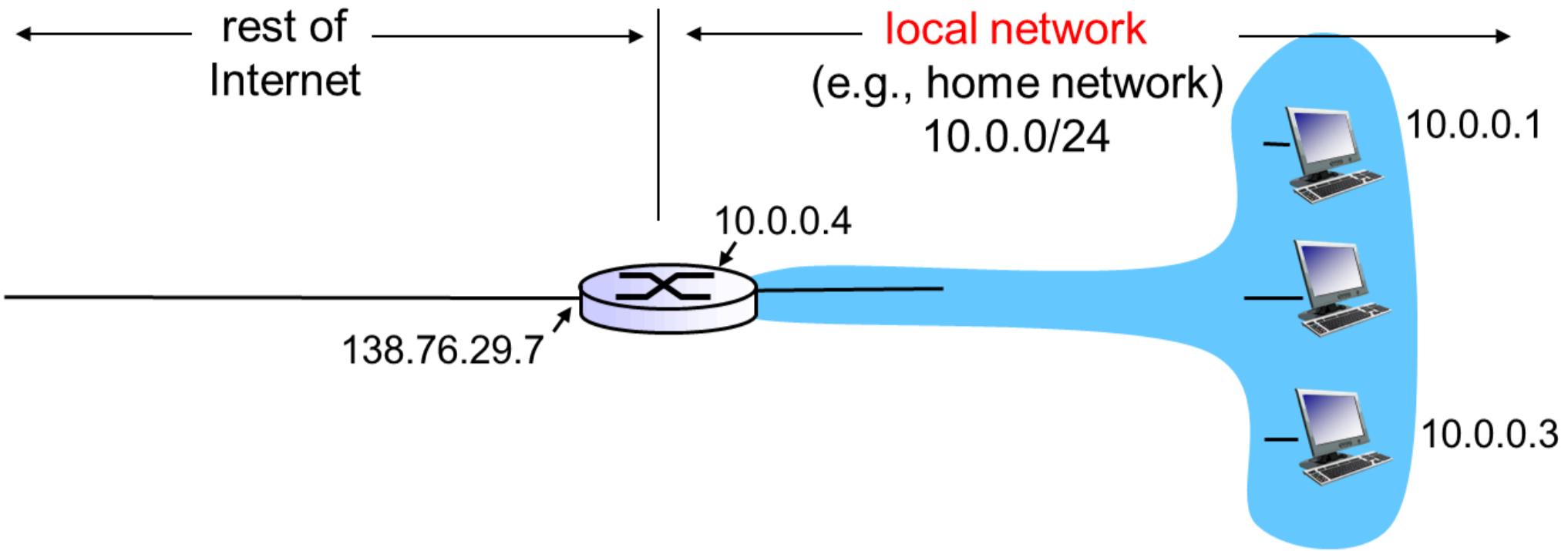
DHCP是用于单主机请求地址的, 如果是一个组织想要获取一大块地址, 那么需要联系ISP, ISP从自己的子网中划分部分给他, 例如200.23.16.0/20内可包含200.23.16.0/23, 200.23.18.0/23等

这种情况下路由表内可能会出现多个匹配项, 仍然使用最长前缀匹配原则

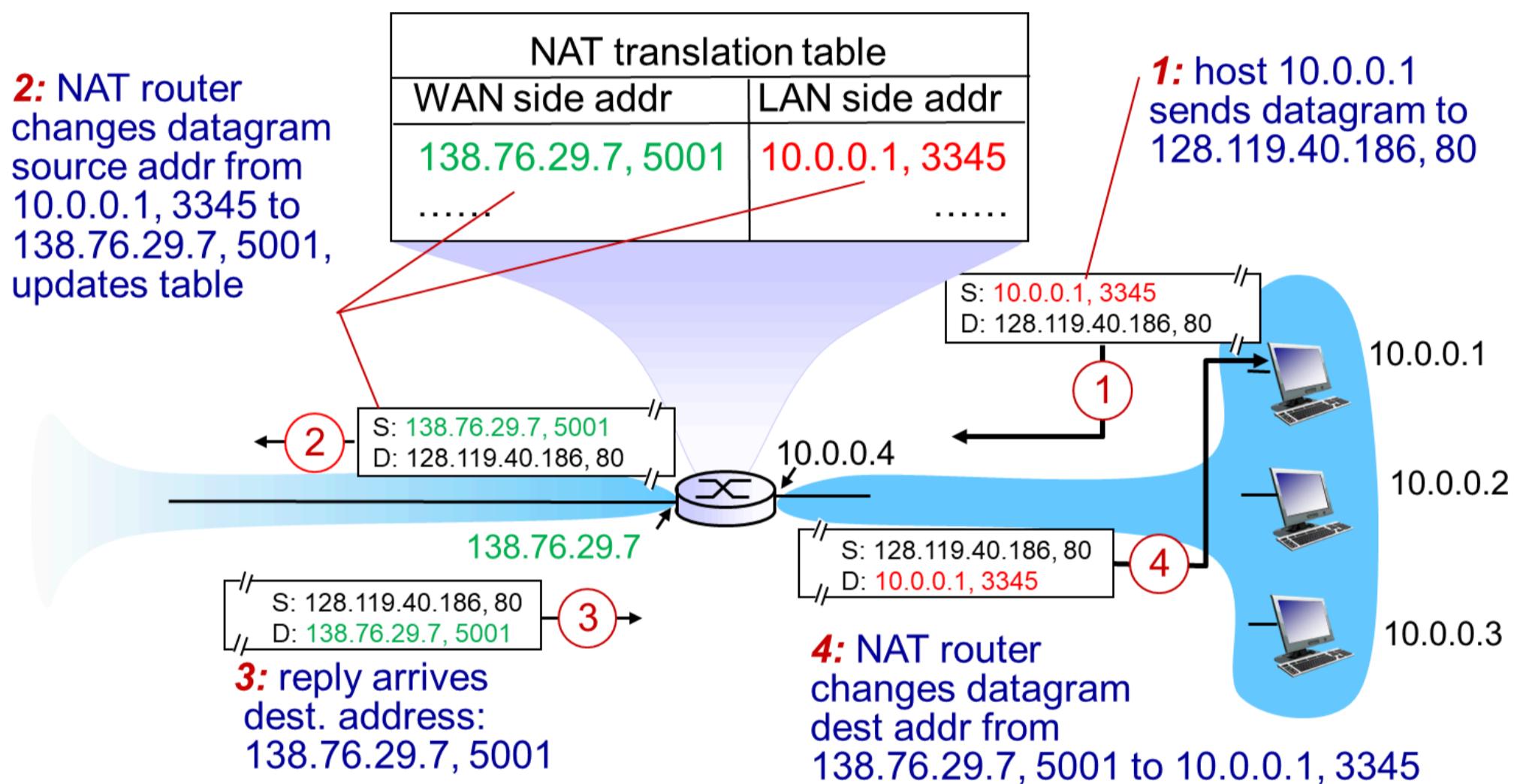
IP address (decimal):	192	168	1	140
IP address (binary):	11000000	10101000	00000001	10001100
Routing table entries	192.168.1.0/24:	11000000	10101000	00000001 00000000 Match 24 prefix bits
	192.168.1.128/25:	11000000	10101000	00000001 10000000 Match 25 prefix bits
	192.168.0.0/16:	11000000	10101000	00000000 00000000 Match 16 prefix bits

## NAT

由于公共IPv4地址已经用尽, 我们需要一种方法来使得多个内部设备共享同一个外部IP, 由此缓解IPv4地址压力。



为得知外部传来的数据包具体是发送给哪个内部设备，路由器需要维护一个NAT转换表（NAT translation table），其内包含了内部主机IP及内部端口号对应外部IP及端口号，当接收到数据报后根据该表进行相应的源IP源端口或目标IP目标端口的转换



## IPv6

为解决IPv4耗尽问题，推出IPv6地址。

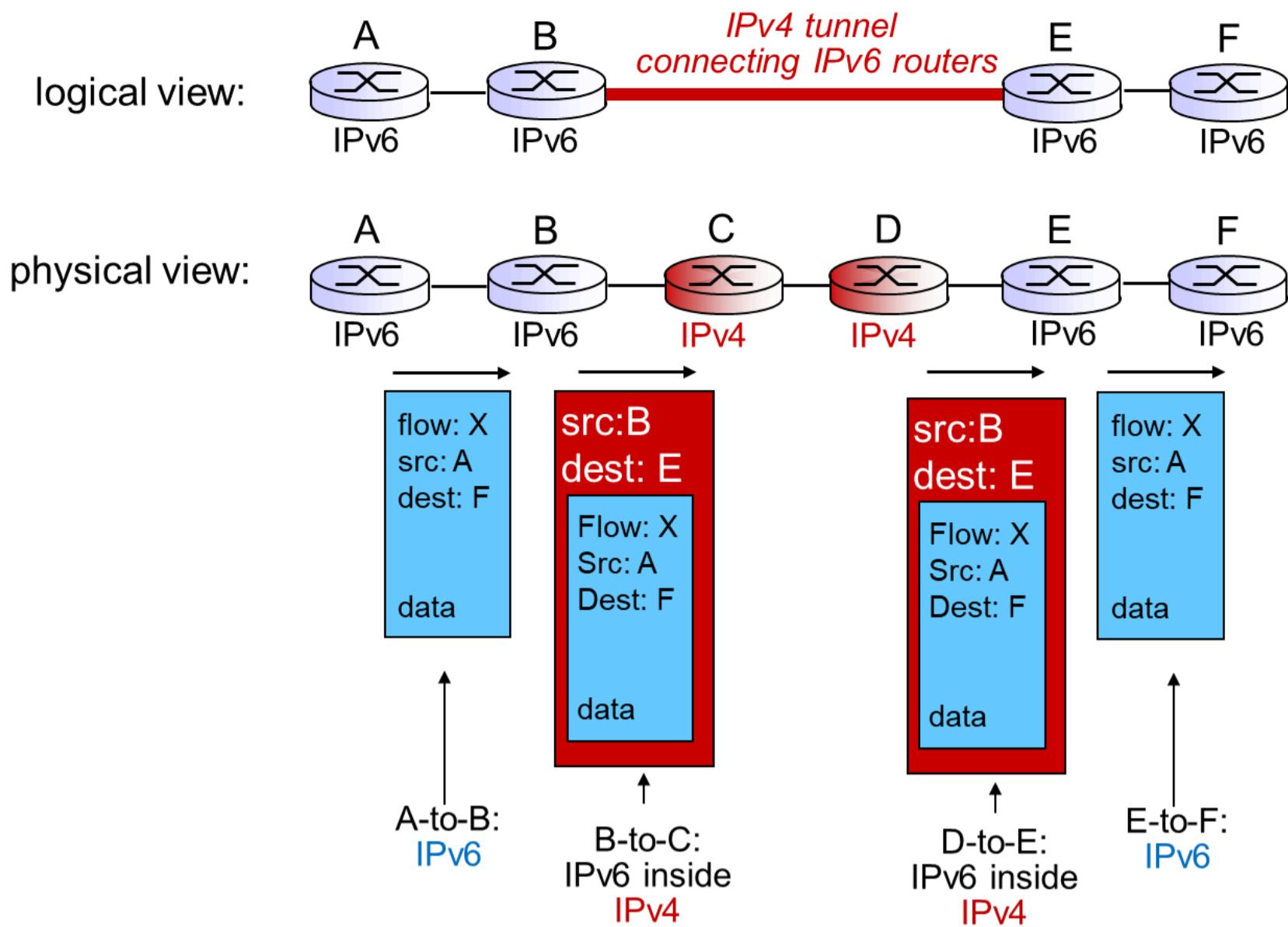
其首部长度仍然是40字节，但地址表示却变为了128位，不存在耗尽问题

但其与IPv4不兼容

而且其不允许分片，不包含首部检验和

既然不兼容，如何从IPV4迁移到IPV6？

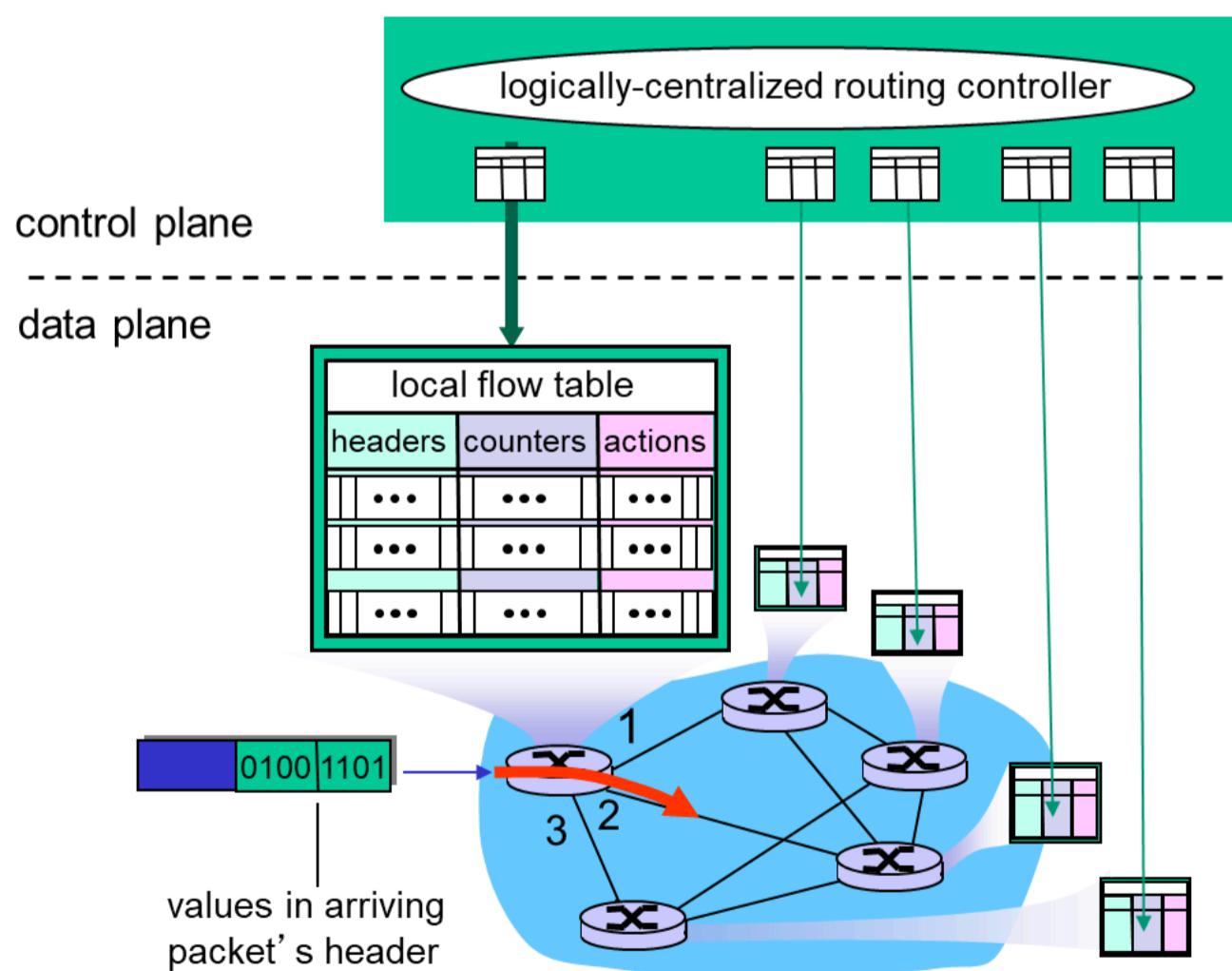
使用隧道（Tunnels）



## 通用转发和流表 Generalized Forwarding and Flow Table

### 通用转发

每个路由器都包含一个路由控制器 (routing controller) 赋予的流表

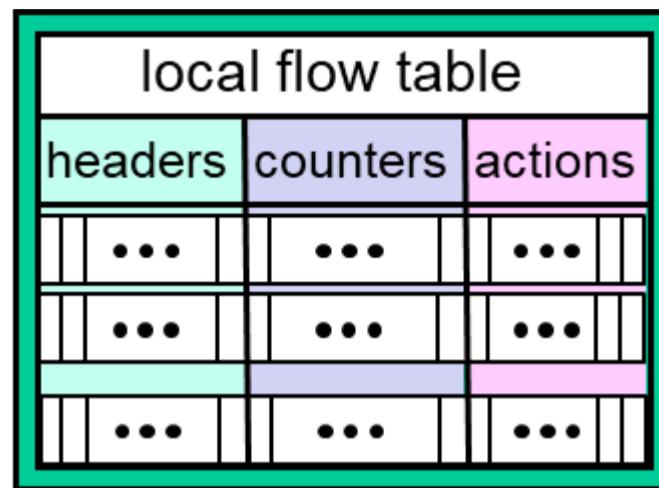


通用转发相比起只根据目的IP选择输出端口的基于目的地转发，其额外考虑更多匹配因素例如协议类型，端口号等，更复杂但更灵活，其动作也能包括例如重写和阻拦等。

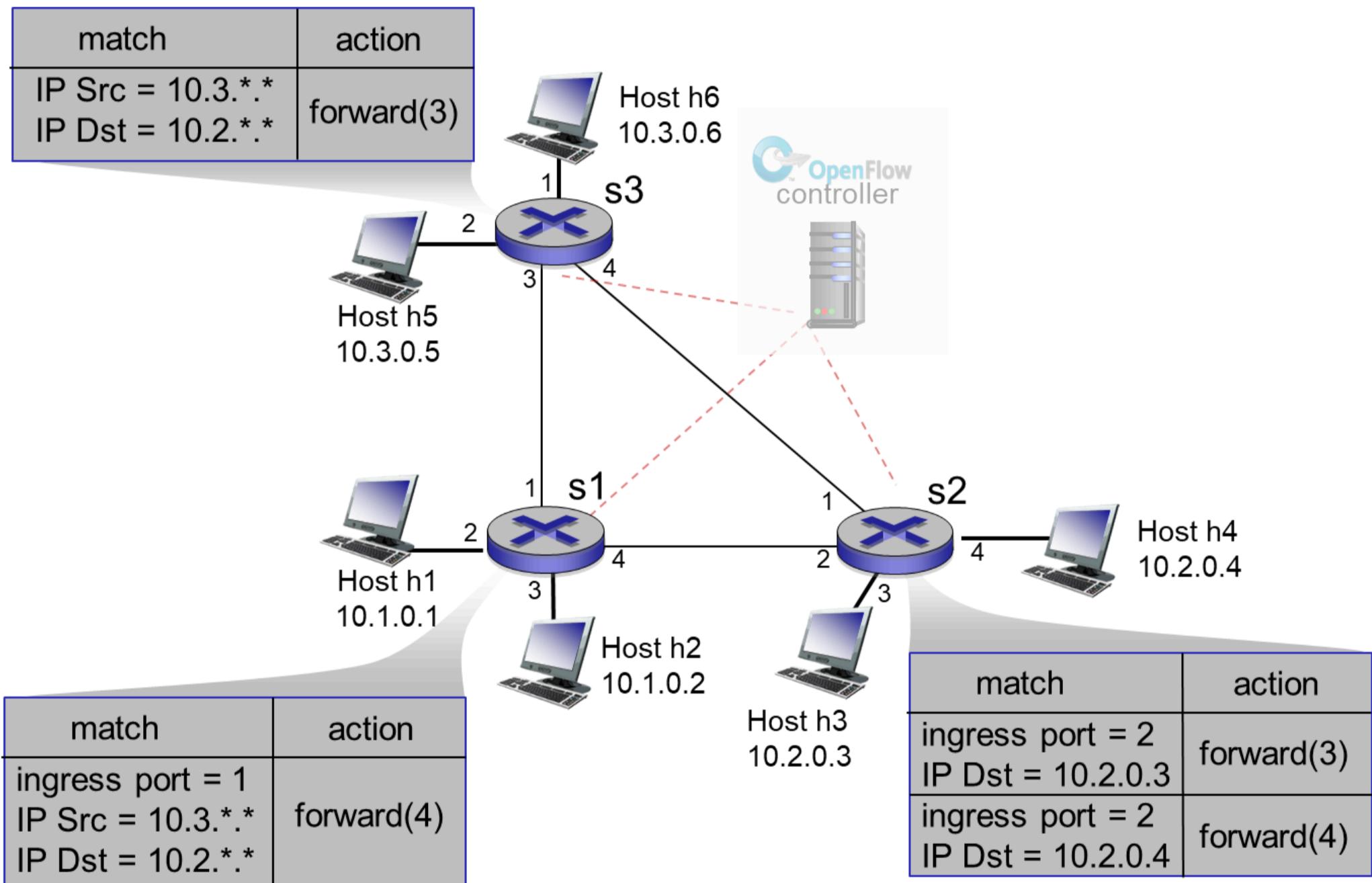
## OpenFlow

OpenFlow标准定义了流表，其内包含了：

- 匹配的首部字段值
- 计数器
- 动作



OpenFlow抽象出了一种match+action的模式，由此适配多种情况和功能。可用于路由器，交换机，防火墙等



## 5. Network Layer: Control Plane

### 概述

控制平面的功能是路由：确定数据包从源到目的地的路径

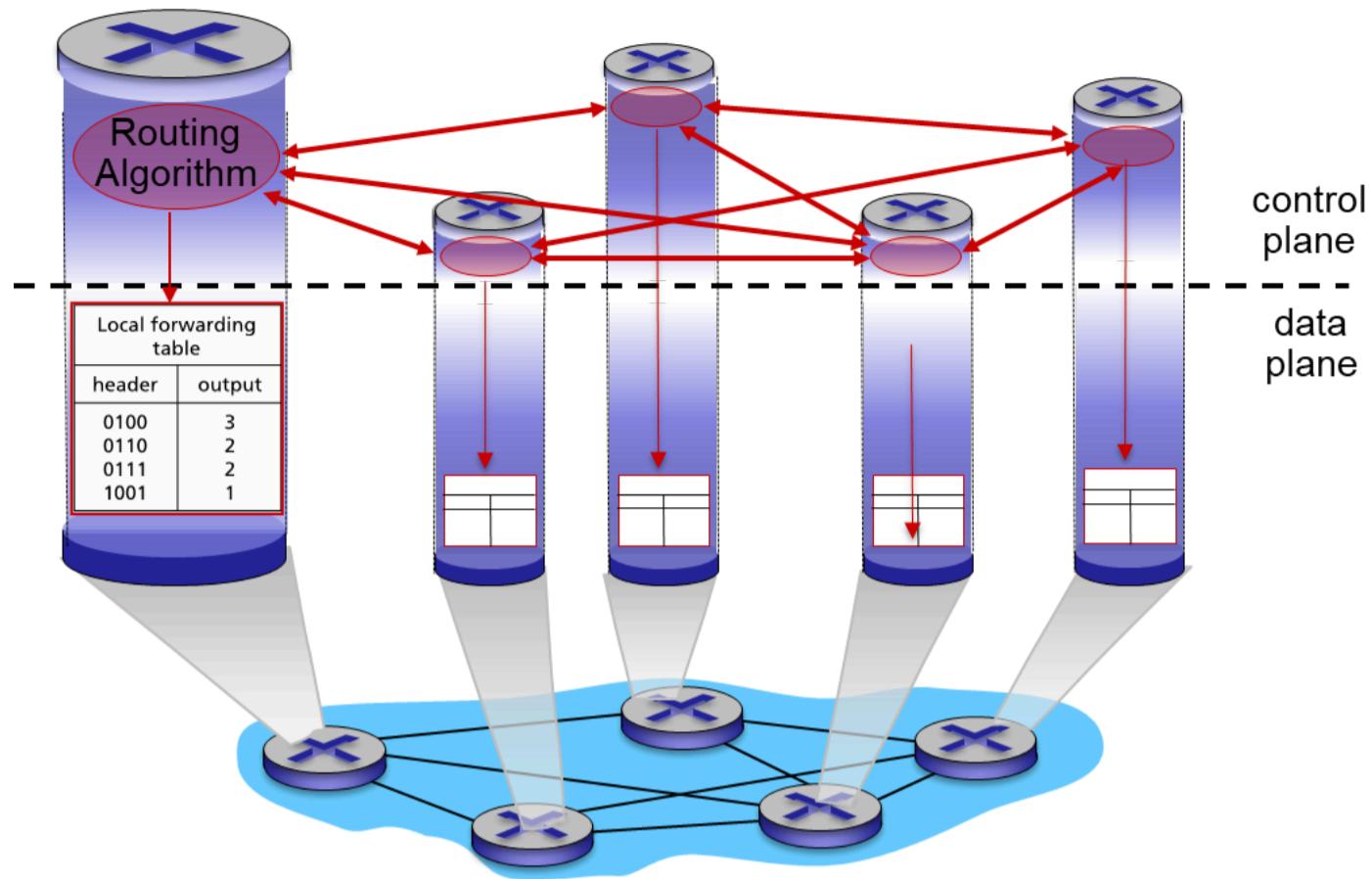
两种方法：

1. 路由器独立控制 per-router control (传统)

## 2. 逻辑集中控制 logically centralized control (软件实现)

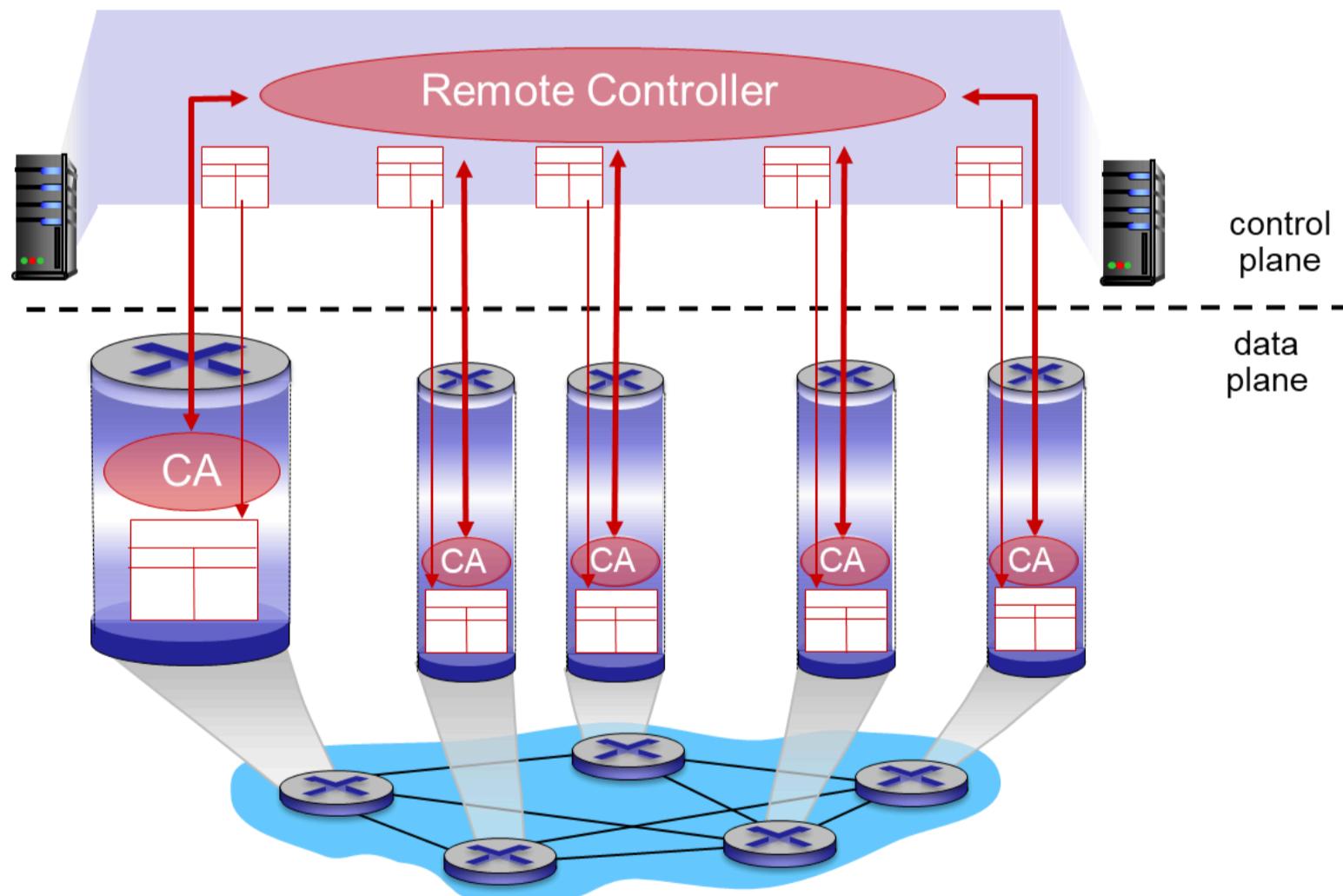
### 路由独立控制

各个路由在控制平面中相互交互以计算路由表



### 逻辑集中控制

远程控制器与本地控制代理CA交互，以计算路由表



## 路由协议 Routing Protocols

目的是从发送方到接收方的过程中确定一条通过路由器网络的具有最低开销的路径

### 链路状态路由选择算法 Link-State Routing Algorithm

在链路状态算法中，网络拓扑和所有的链路开销都是已知的，在实践中通常通过链路状态广播 (link state broadcast) 完成

## Dijkstra算法

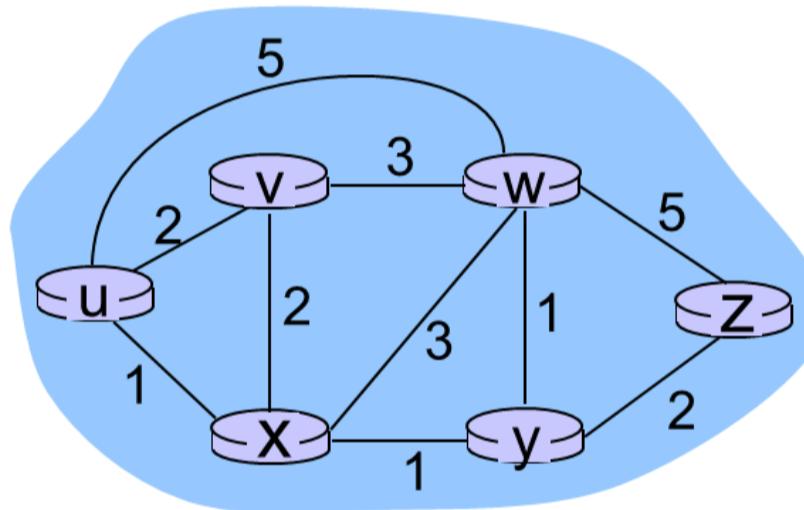
计算从某节点（源节点，我们称之为u）到网络中所有其他节点的最低开销路径。经过k次迭代后，可知道到k个目的节点的最低开销路径

定义：

- $c(x,y)$ : 从节点x到y的开销
- $D(v)$ : 目前从源节点到目标节点v的已知最低开销
- $p(v)$ : 从源到v的沿着最低开销路径的前一节点 (v的邻居)
- $N'$ : 最低成本路径明确已知的节点集

算法的核心思想是不断检查是否可用经过目标节点的具有最短路径的邻居节点来缩短最低路径，以此循环

Step	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	$\infty$	$\infty$
1	ux	2,u	4,x		2,x	$\infty$
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



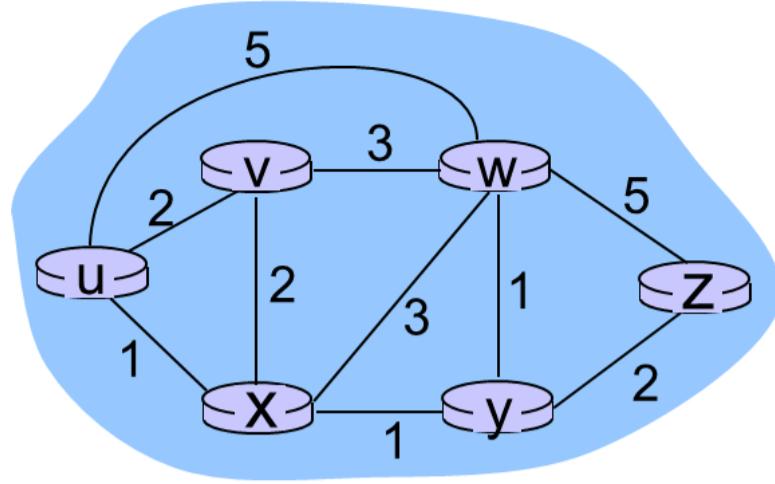
5-15

## 距离向量算法 Distance-Vector

Bellman-Ford方程：令 $d_x(y)$ 表示从x到y的最短开销路径， $c(x, v)$ 表示从x到邻居v的开销

则有：

$$d_x(y) = \min\{c(x, v) + d_v(y)\}$$



Given  $d_v(z) = 5$ ,  $d_x(z) = 3$ ,  $d_w(z) = 3$

B-F equation says:

$$\begin{aligned} d_u(z) &= \min \{ c(u,v) + d_v(z), \\ &\quad c(u,x) + d_x(z), \\ &\quad c(u,w) + d_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

DV算法基于BF方程，每个节点都要维护：

- 其到达每个邻居的开销 $c(x, y)$
- 其自身的距离向量 $D_x$
- 其邻居的距离向量 $D_v = [D_v(y) : y \in N]$

各节点时不时互相分享其距离向量，并根据BF方程互相更新

## OSPF

### AS 路由

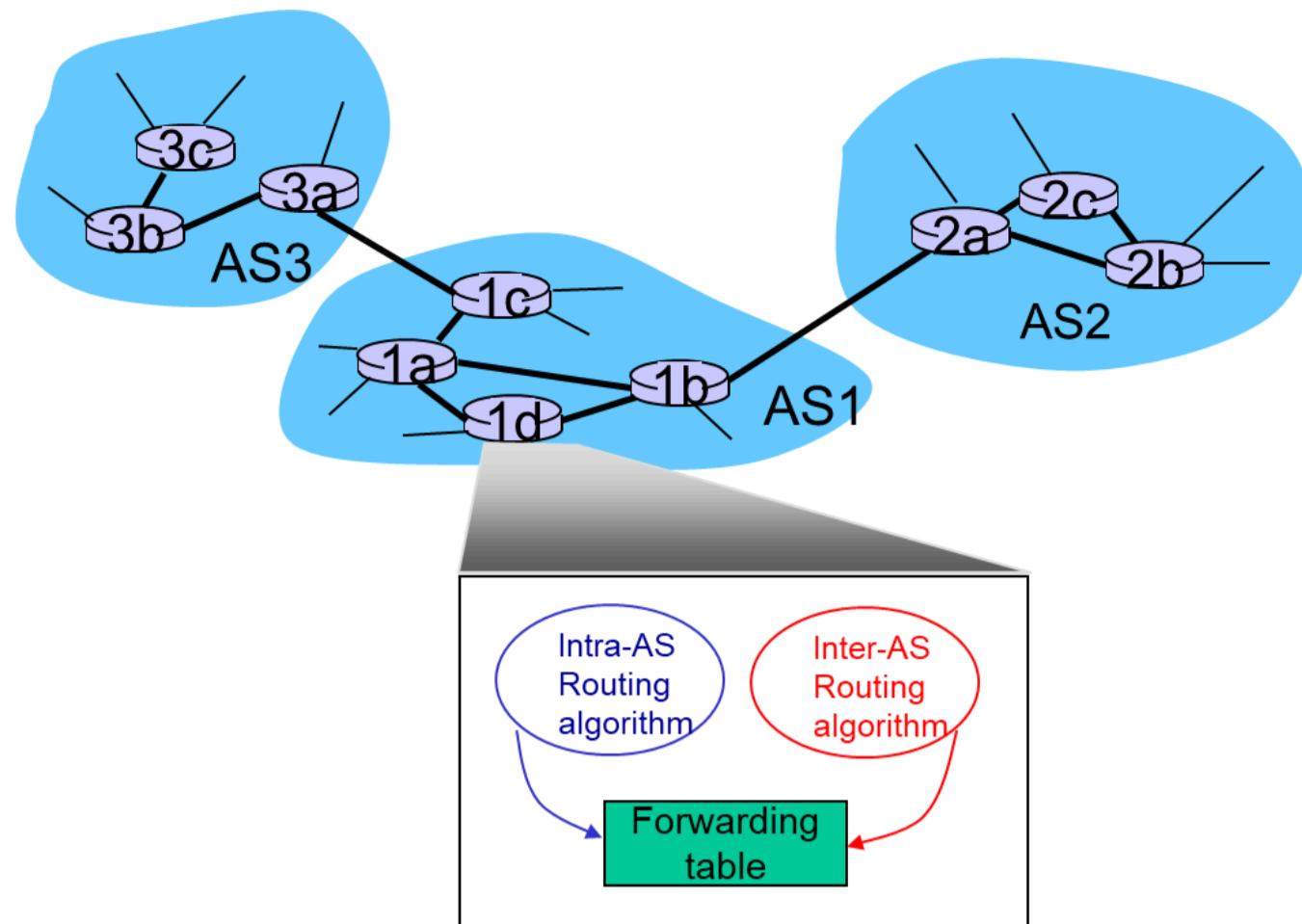
在实际的路由计算中，路由器的规模十分庞大，因此将其组织进AS (Autonomous System) 来减小路由计算的复杂性

内部AS路由：

- 主机以及同一AS内的路由
- 所有路由都要运行相同的域内协议
- 不同AS可以运行不同的域内协议
- 网关路由器：位于AS边缘，与其他AS链路相连

AS间路由：

- AS之间的路由
- 网关路由器执行域间路由（也执行域内路由）



## OSPF

内部AS路由也被称为内部网关协议 (Interior Gateway Protocols IGP) , 常见的有OSPF, RIP

OSPF (Open Shortest Path First)

1. 使用泛洪链路状态信息 (Flood Link State Information)
2. 通过泛洪得到整个AS的拓扑图 (topology map)
3. 运行Dijkstra算法, 以自身为根节点确定最短路径

## BGP

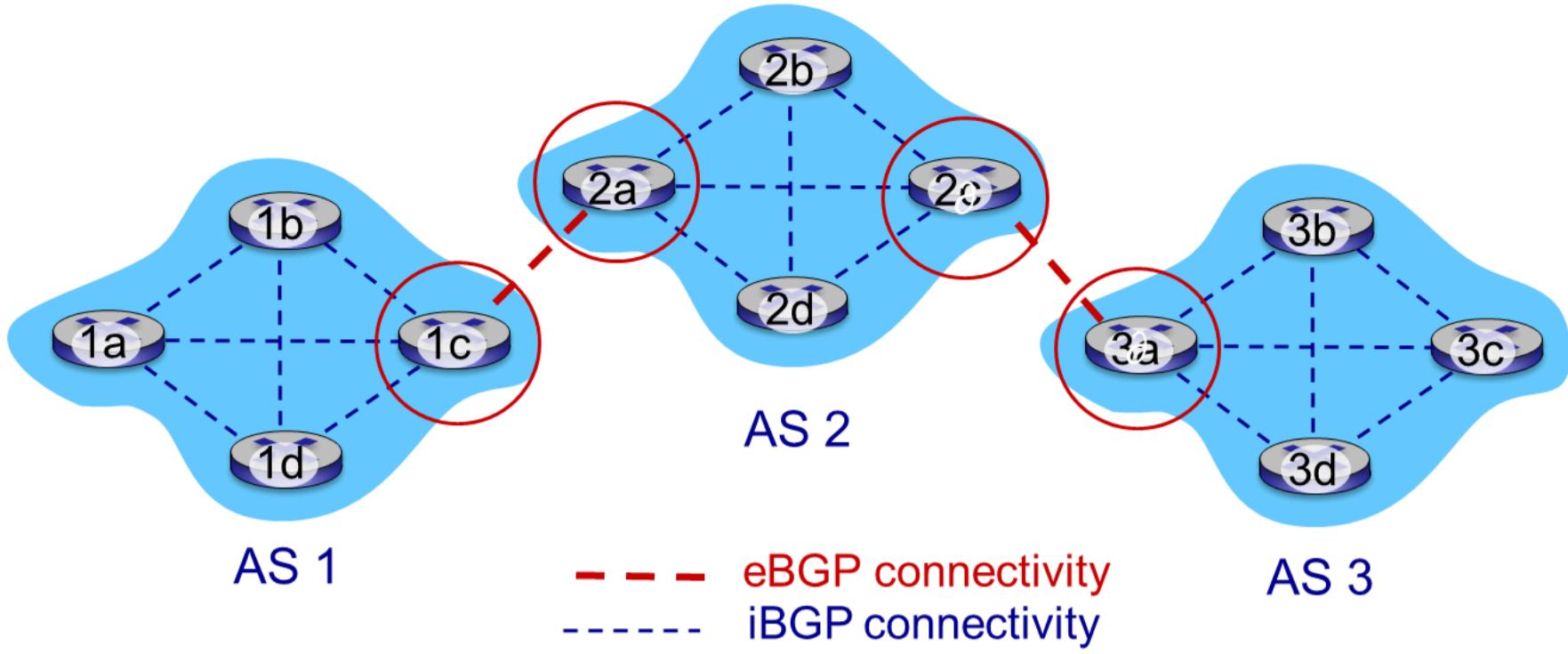
边界网关协议用于使得AS了解其他AS的信息以及如何到达, 并告知此AS内的所有路由器

BGP:

- 外部BGP (External BGP, eBGP) : 从相邻AS获取子网访问信息
- 内部BGP (Internal BGP, iBGP) : 将访问信息传播到AS内部所有路由器

两个BGP路由器通过半永久TCP连接来交换BGP信息

BGP将各个AS连接在了一起



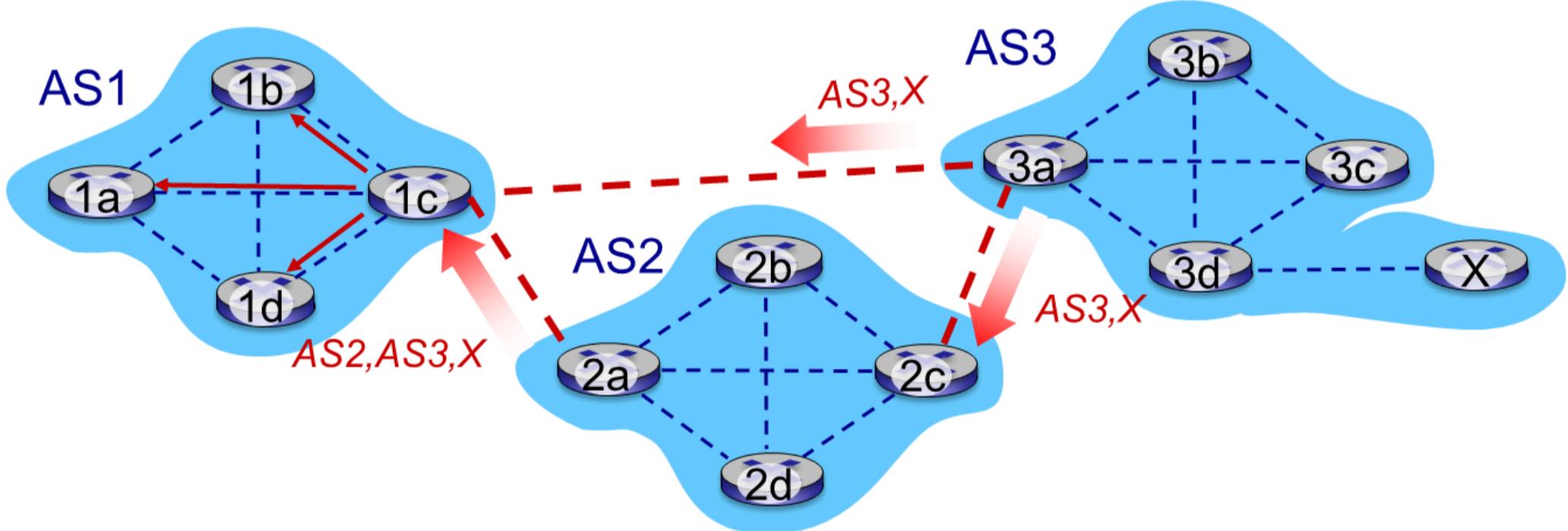
gateway routers run both eBGP and iBGP protocols

在BGP中, route = prefix+attributes

比较重要的属性包括:

- AS-PATH: 包含前缀通告通过其的AS列表
- NEXT-HOP: 沿AS-PATH起始的路由器接口的IP地址

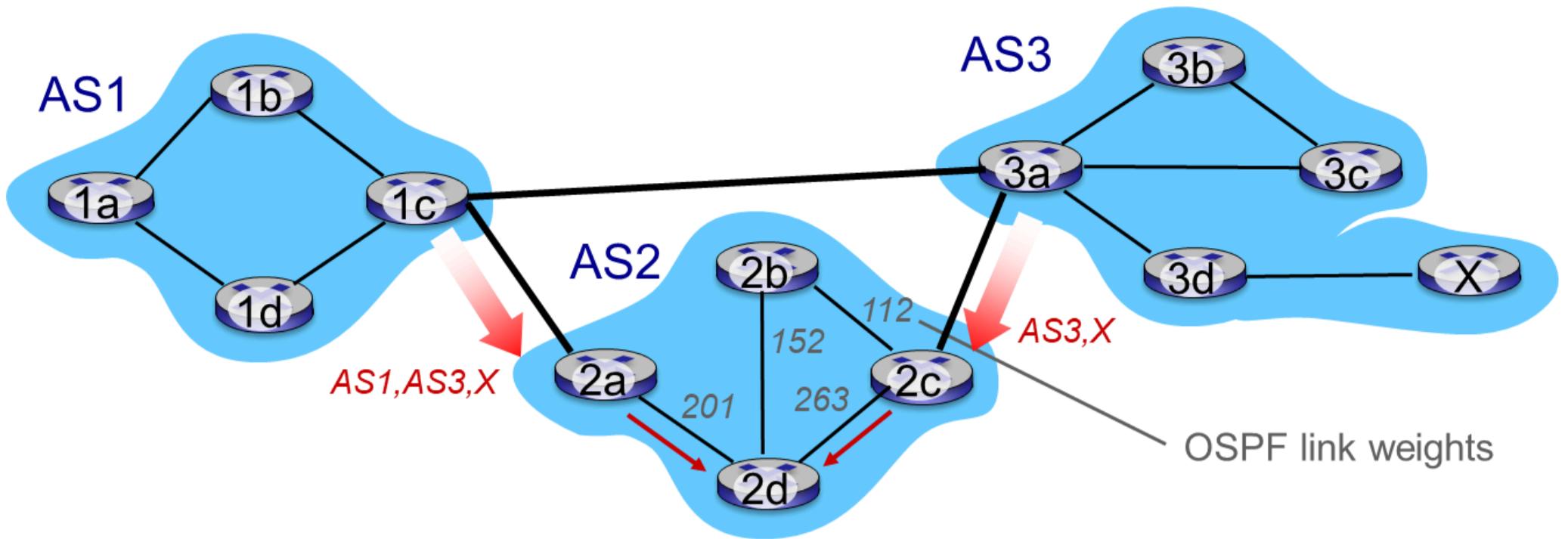
如下图, 1c可接收到两个到X的路由, 它应该如何选择



## 热土豆路由 (Hot Potato Routing)

选择具有最低开销的域内路由

2d得知其可用通过2c或2a到达X, 使用热土豆路由, 其选择2a



## ICMP

internet control message protocol

由主机和路由器用来传达网络级信息，进行差错报告等

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

## Consideration of Routing protocols

**政策:** 在 AS 中，政策问题占主导地位。源自给定 AS 的流量无法通过另一个特定 AS 可能很重要。同样，给定的 AS 可能希望控制它在其他 AS 之间传输的传输流量。在 AS 中，所有内容名义上都处于相同的管理控制之下，因此在 AS 中选择路由时，策略问题的作用要小得多。

**规模:** 路由算法及其数据结构扩展以处理到大量网络/在大量网络之间进行路由的能力是 AS 间路由中的一个关键问题。在 AS 中，可扩展性不太重要。首先，如果单个管理域变得太大，则始终可以将其划分为两个 AS，并在两个新 AS 之间执行 AS 间路由。

**性能:** 由于 AS 间路由非常面向策略，因此所用路由的质量（例如，性能）通常是次要的（即，满足某些策略条件的更长或成本更高的路由很可能被较短但不满足该条件的路由采用）。事实上，我们看到，在 AS 中，甚至没有与路由相关的成本（AS 跳点计数除外）的概念。然而，在单个 AS 中，此类策略问题不太重要，允许路由更多地关注路由上实现的性能级别。

- **Policy:** Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS not be able to pass through another specific AS. Similarly, a given AS may want to control what transit traffic it carries between other ASs. Within an AS, everything is nominally under the same administrative control and thus policy issues a much less important role in choosing routes within AS.

- Scale: The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if a single administrative domain becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs.

- Performance: Because inter-AS routing is so policy oriented, the quality (for example, performance) of the routes used is often of secondary concern (that is, a longer or more costly route that satisfies certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of cost (other than AS hop count) associated with routes. Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on the level of performance realized on a route.

## 6. Link Layer

### 概述

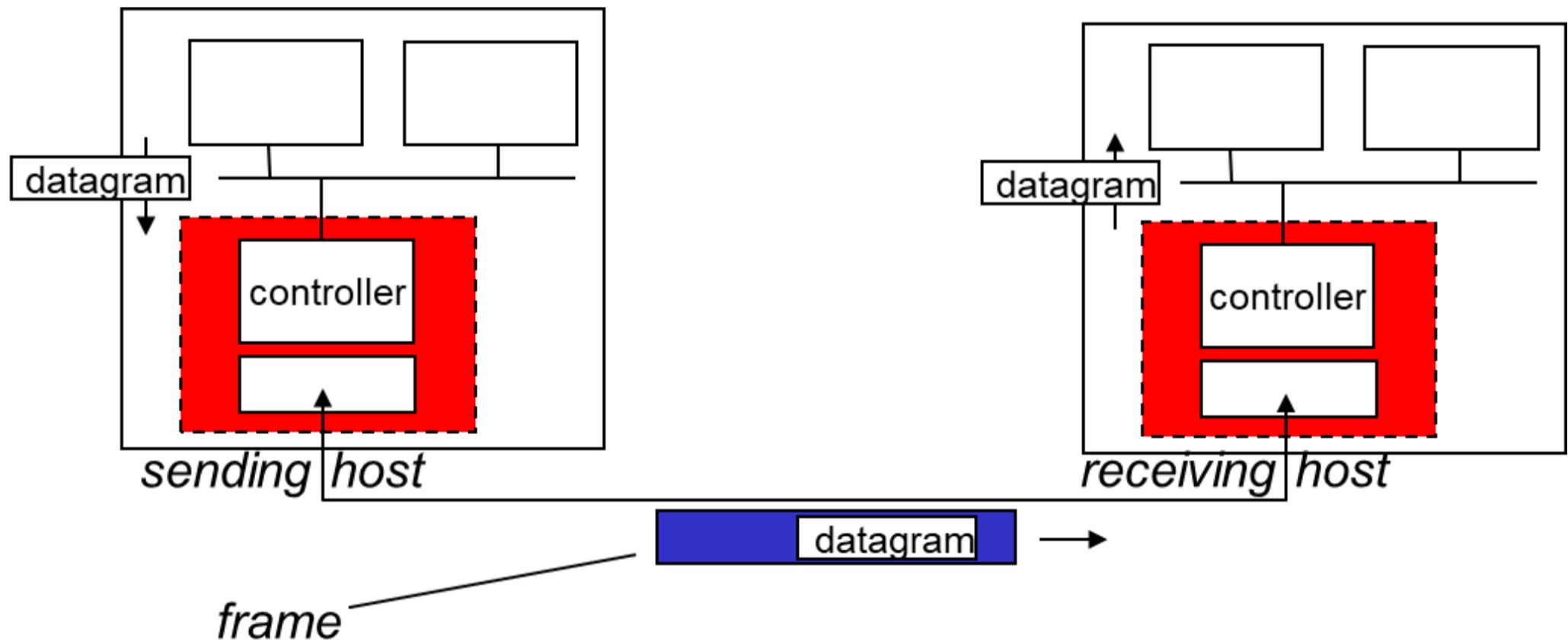
在链路层中，主机和路由器称为节点（node），节点之间相连的通道称为链路(link)，数据包称为帧(frame)

链路层负责从一个节点传输数据报到链路上物理相邻的节点

数据链路层将数据封装成帧，使用MAC地址来标记源和目的地（与IP不同）

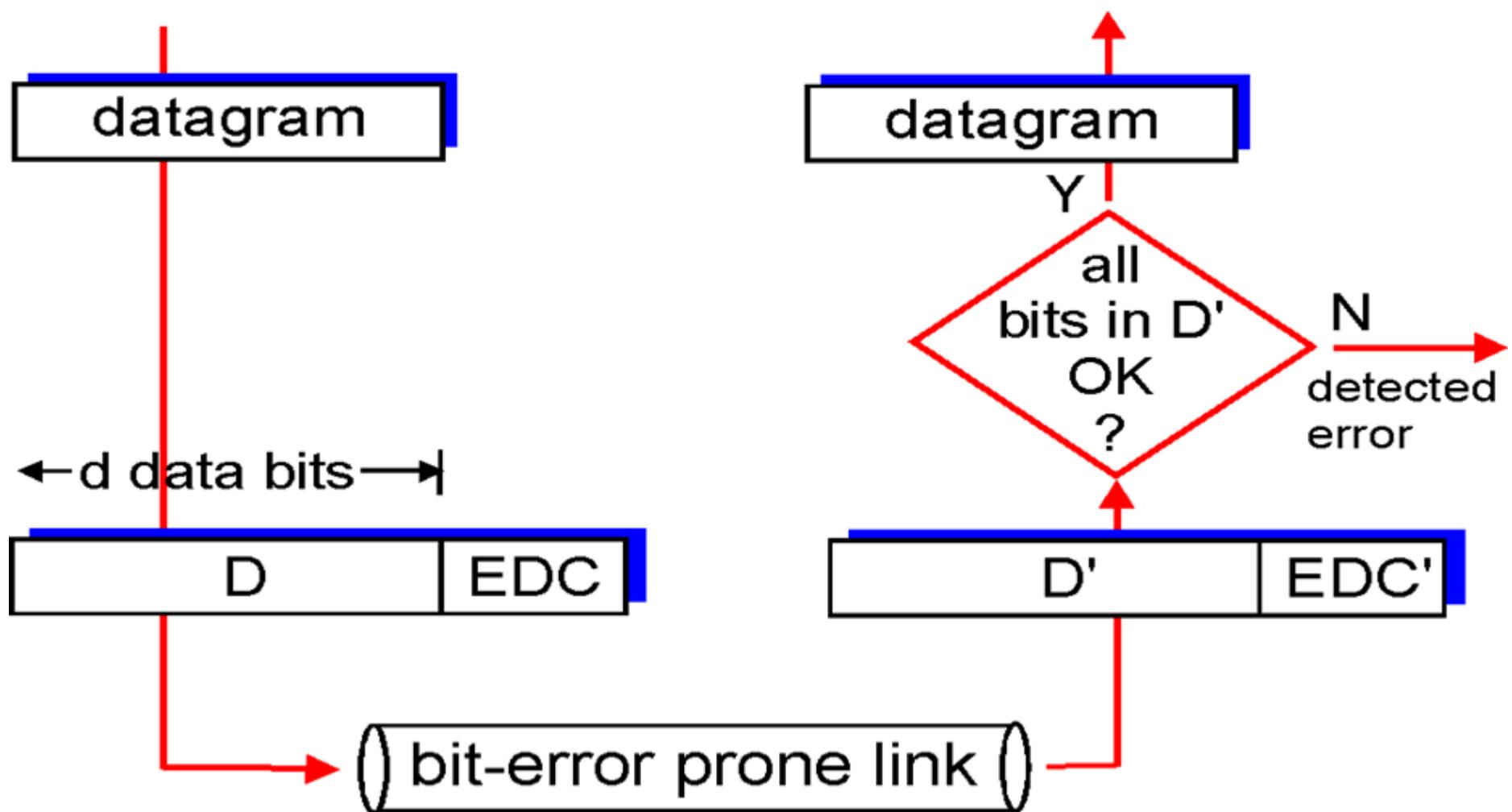
该层还提供了差错检测和纠正

在任意主机和适配器中都涉及到链路层，可看作是硬件和软件的结合



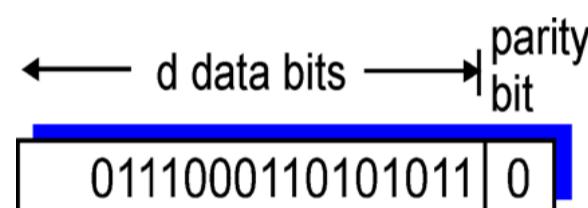
### 差错检测

差错检测的实现依赖于D (数据) + EDC (冗余的检错位)

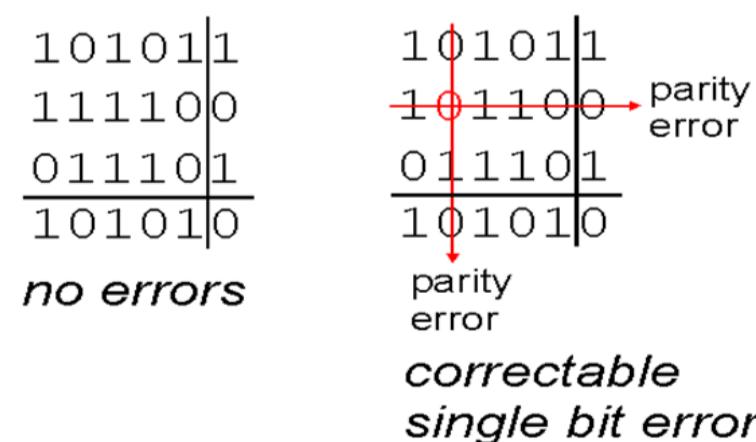
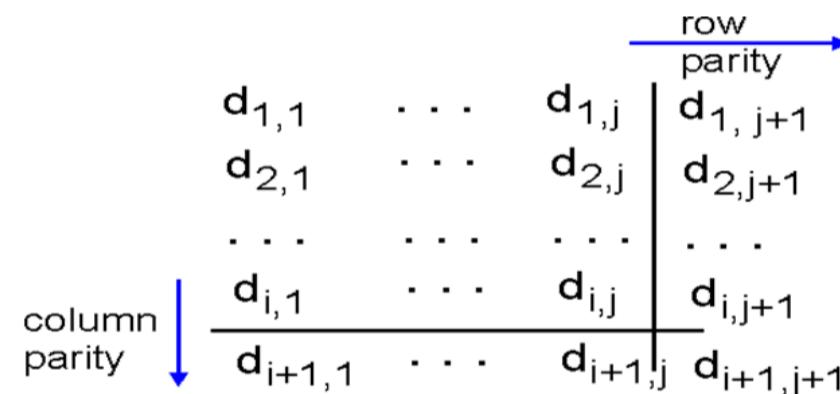


## 奇偶校验

单比特奇偶校验：检测单比特错误



二维奇偶校验：检测和纠正单比特错误：



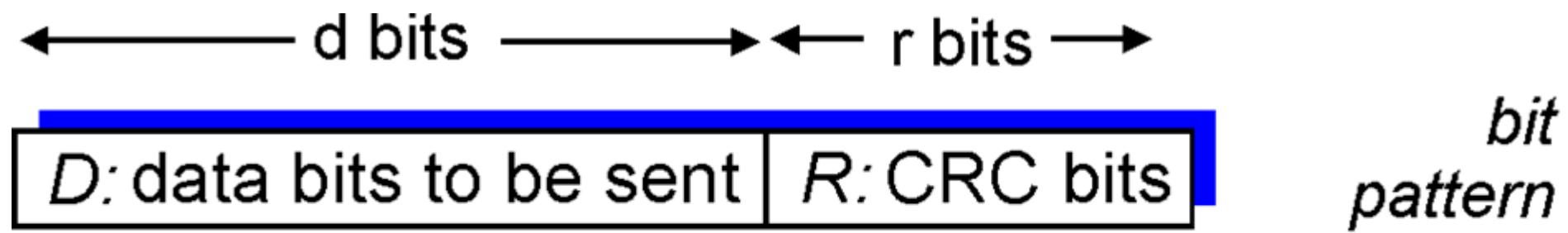
## 循环冗余检测CRC

广泛使用

D: 数据位, 01多项式形式

G: 生成多项式, 具有r+1位长

R: r位冗余检测位, 可检出不超过r+1位错

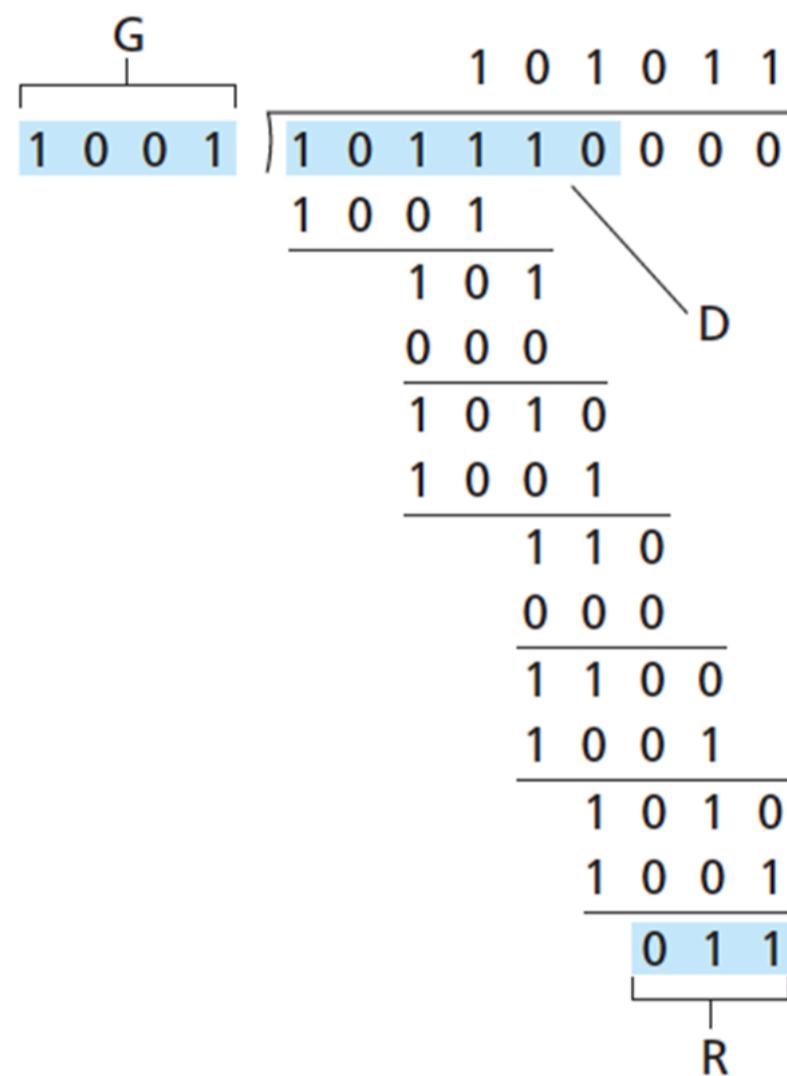


$D * 2^r \text{ XOR } R$       *mathematical formula*

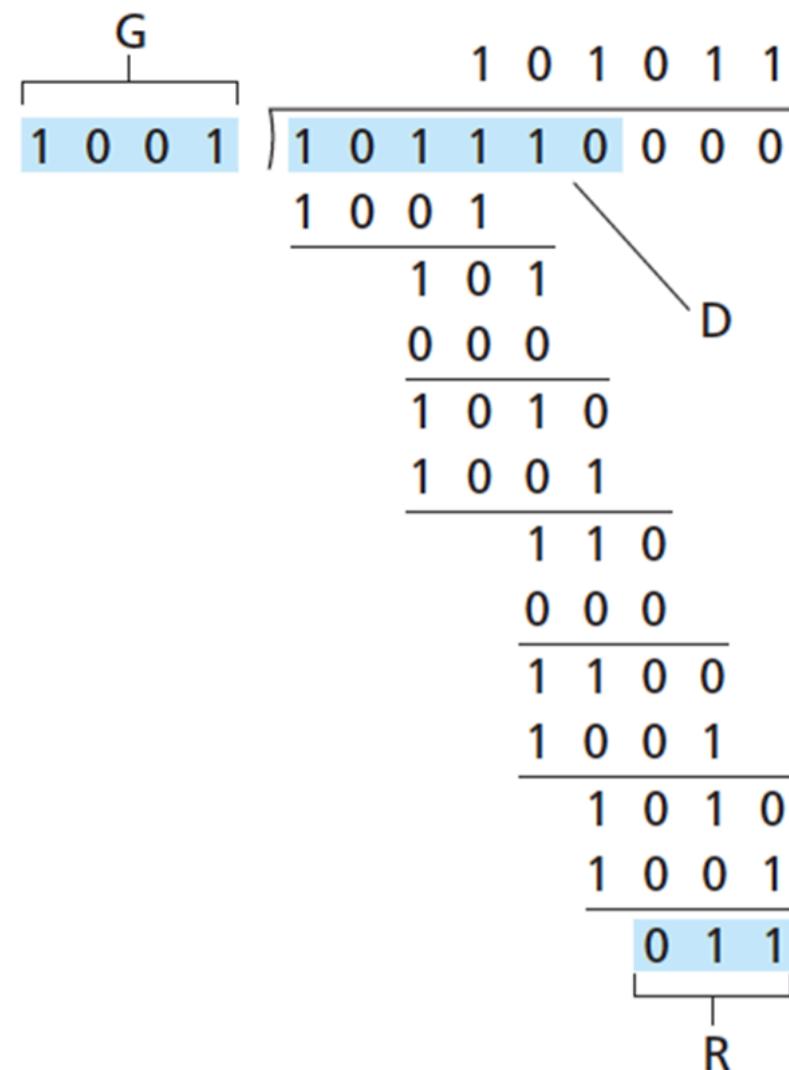
检错过程:

将 $\langle D, R \rangle$ 除以G, 若余数为0则无差错

编码: 将D写成 $D+r$ 位0的形式再除 (异或)



解码: 将D加上余数R再做除 (异或)



# 多路访问协议

点对点链路很可靠，例如以太网主机和交换机之间的链路

但更多的是广播链路，共享线路或媒介

这种情况下如何可靠地共享线路很重要，因为如果同时发送，碰撞会导致帧无法被成功接收。

通过协议的方式来协商如何共享，即多路访问协议（multiple access protocol）

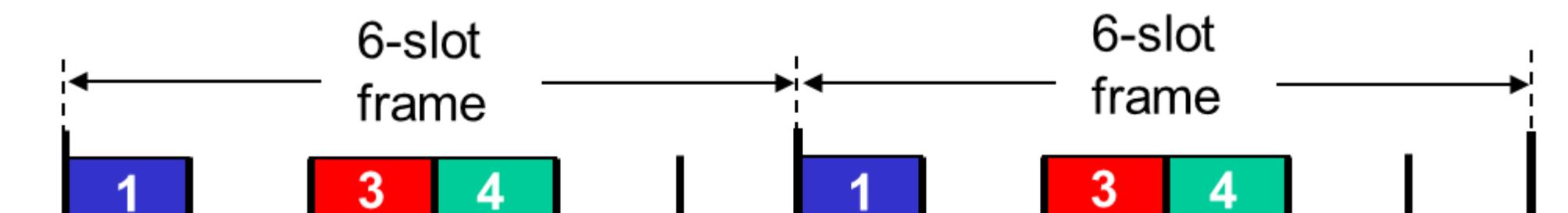
一个理想的协议满足：

1. 一个节点运输时有速率R
  2. M个节点同时运输有 $R/M$
  3. 协议去中心化
  4. 协议简单

## 信道划分协议 (Channel partitioning MAC protocols)

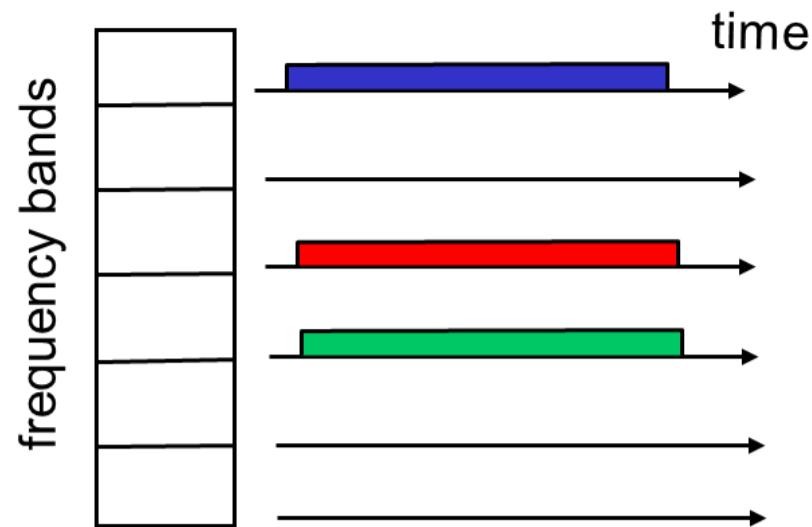
TDMA (time division multiple access) :

轮流访问，每个节点使用固定长时间



FDMA(frequency division multiple access):

线路被划分成固定频段



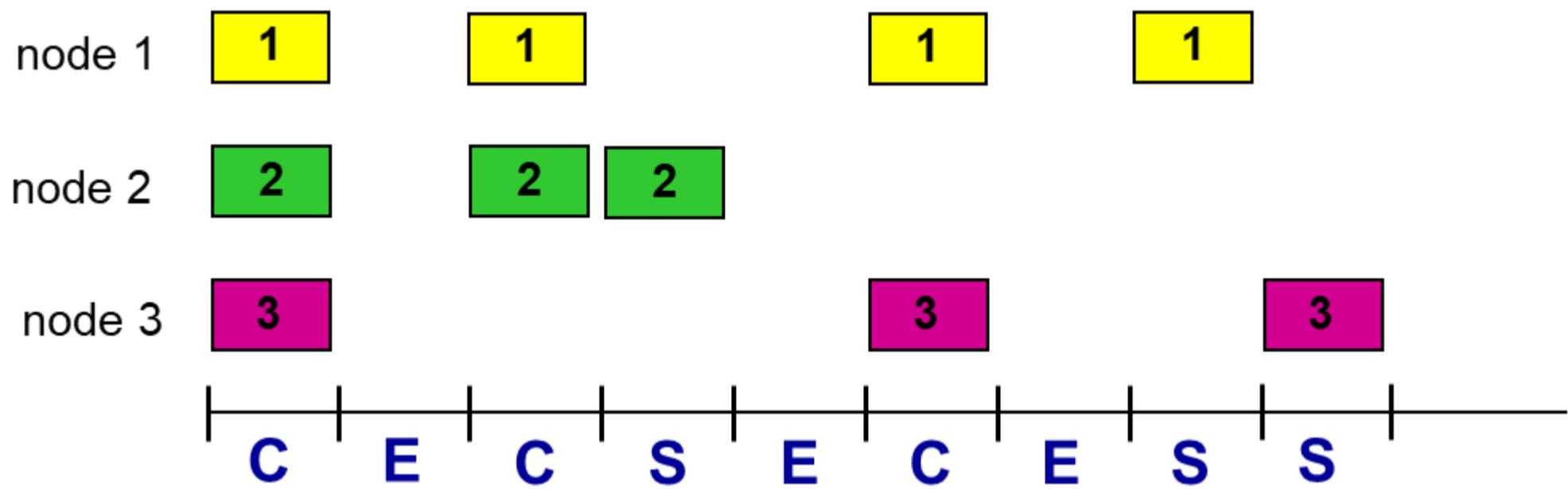
## 随机接入协议 (Random access protocols)

节点全力接入，当发生碰撞时稍等一定时延后重新发送直到发送完成

时隙 ALOHA (Slotted ALOHA)

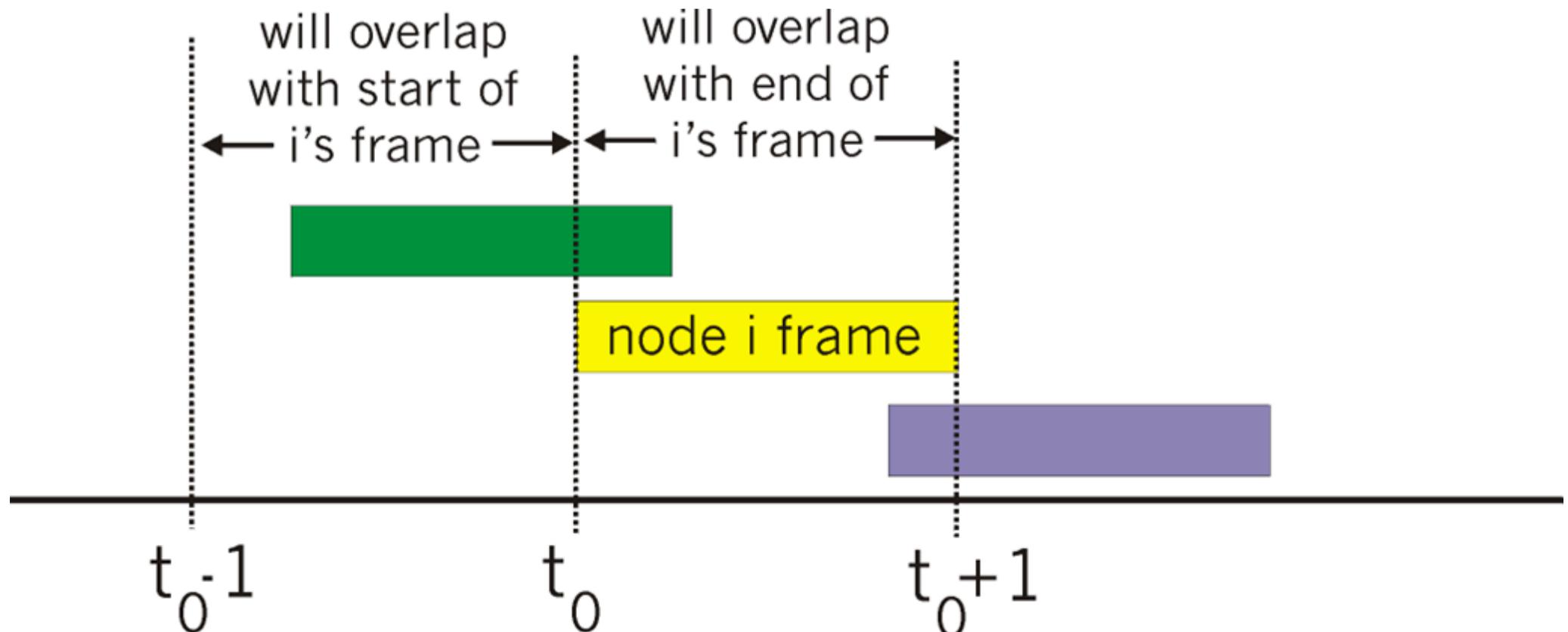
所有帧由L比特组成，时间被划分成长度为L/R秒的时隙，节点在时隙起点开始尝试传输帧。如果发送碰撞，则以概率p在后续的每个时隙中重传它的帧，直到该帧被无碰撞地传输出去。

该方法简单但最好情况下也只有37%的效率



ALOHA

更简单，不需要时间同步，当收到帧后直接传输，但碰撞概率增加，表现甚至不如时隙ALOHA



载波侦听多路访问 (carrier sense multiple access, CSMA)

在传输前先侦听，若通道空闲则传输，否则等待片刻

但这不意味着CSMA不会碰撞，传输时延的存在导致两个不同节点可能会碰巧接近决定在此刻使用目前空闲的信道

具有碰撞检测的载波侦听多路访问 (CSMA/CD)

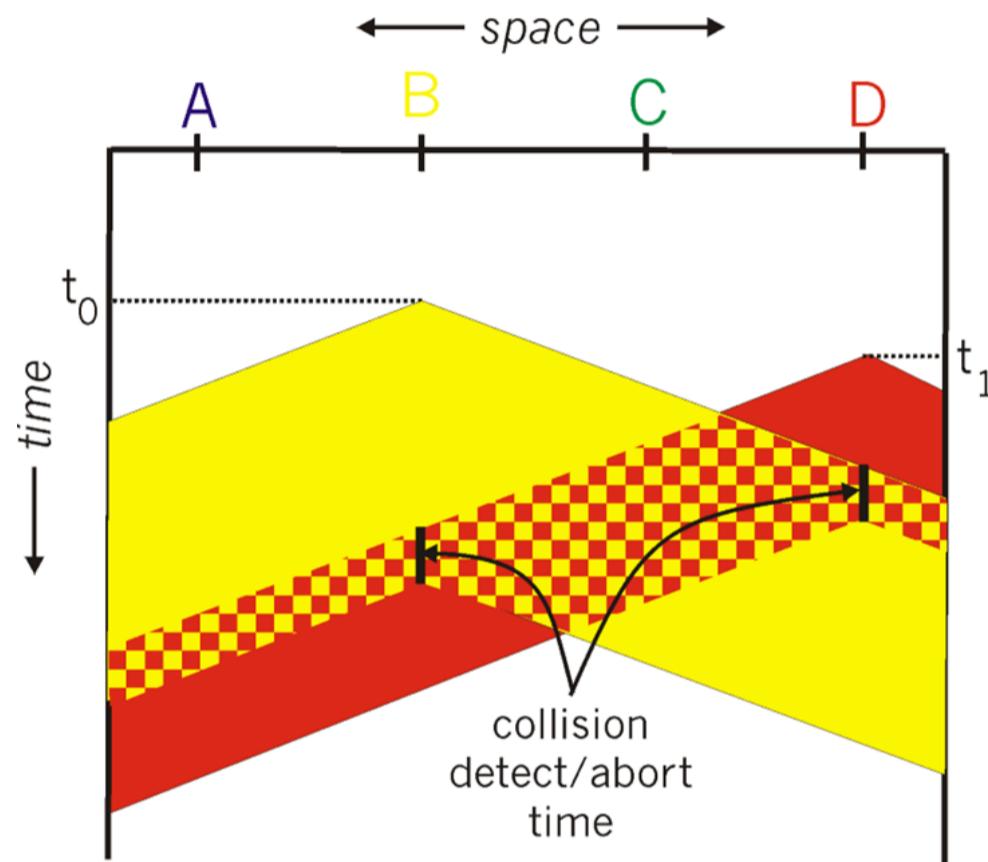
以太网采用

在传输前先侦听，若通道空闲则传输，否则等待片刻。开始传输后检测是否有碰撞，如果有则停止传输。

这一方法使得注定无用的（碰撞的）帧会及时停止传输，提高了通道可用效率。

以太网中，在第m次碰撞后，随机从 $\{0,1,2,\dots,2^m - 1\}$ 中选择一个K值，随后等待 $K \times 512$ 比特时间

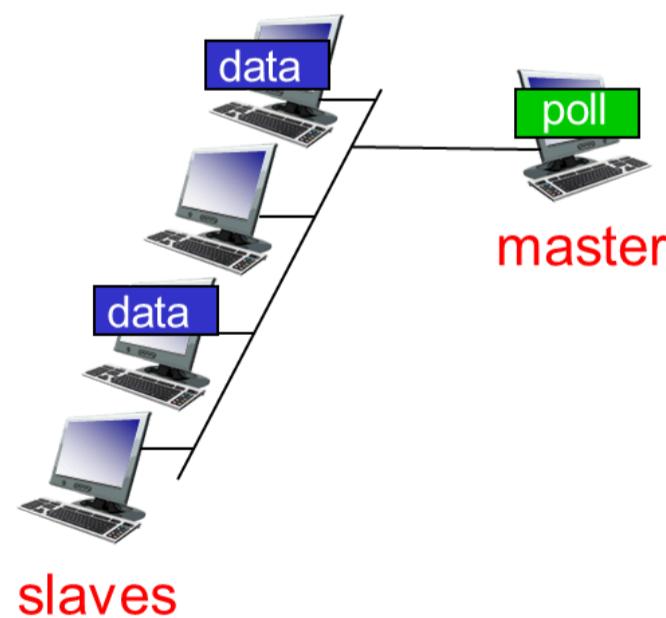
比ALOHA更简单更高效



## 轮流协议 Taking turns protocols

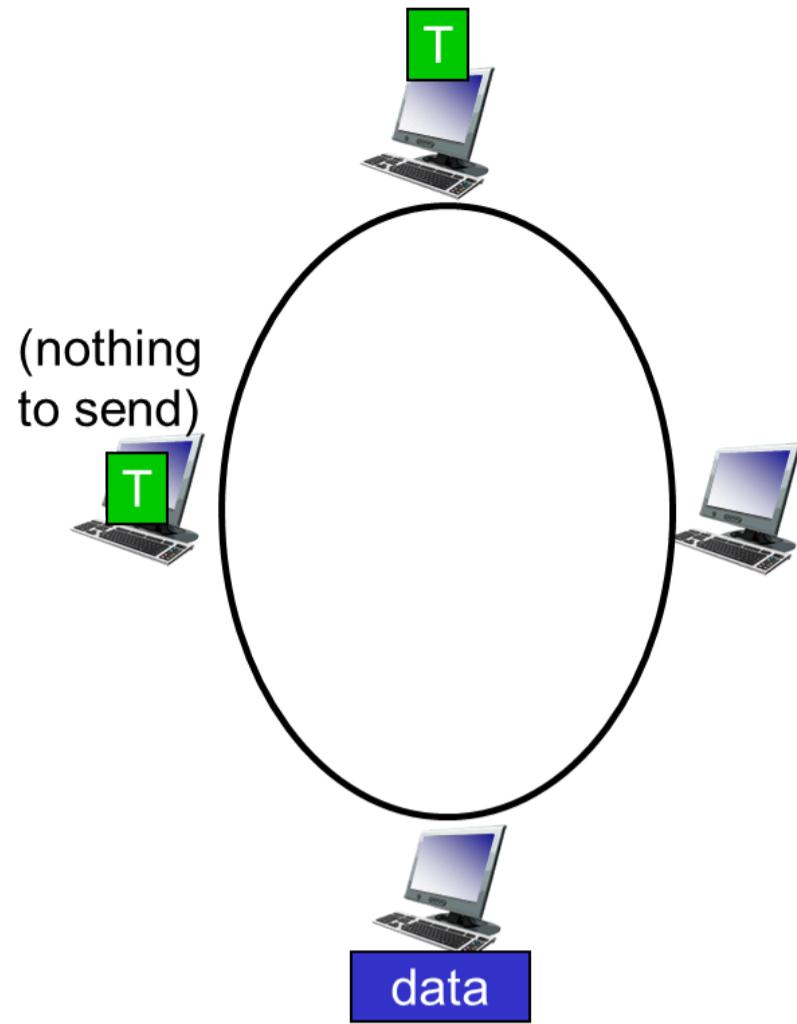
轮询协议

指定一个主节点，以循环的方式轮询 (poll) 每个节点，告诉它们可传输的帧数量



令牌传递协议 (token-passing protocol)

以某种规则传递令牌，收到令牌并需要传输的节点进行传输，否则继续传递



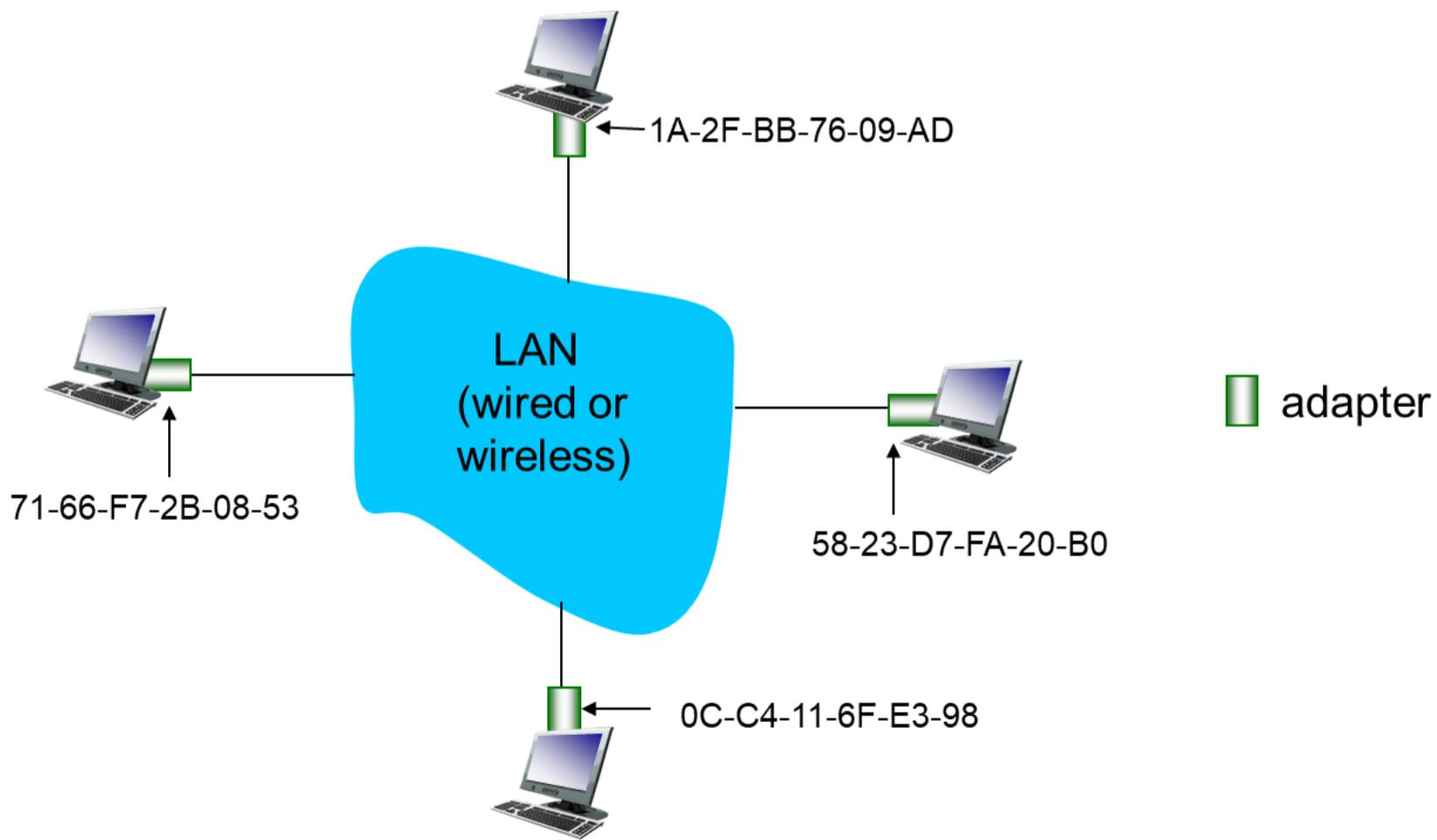
## 局域网

### addressing and ARP

#### MAC地址

6字节,  $2^{48}$ 种可能, 16进制表示 e.g.: 1A-2F-BB-76-09-AD

主机或路由器的适配器 (即网络接口) 具有独一无二的链路层MAC地址



### ARP协议 address resolution protocol

地址解析协议, 用于IP地址与MAC地址的转换。仅可用于同子网内, 不像DNS可在互联网任何地方使用

每台主机或路由器在其内存中具有一个ARP表 (ARP table), 这张表包含IP地址到MAC地址的映射关系

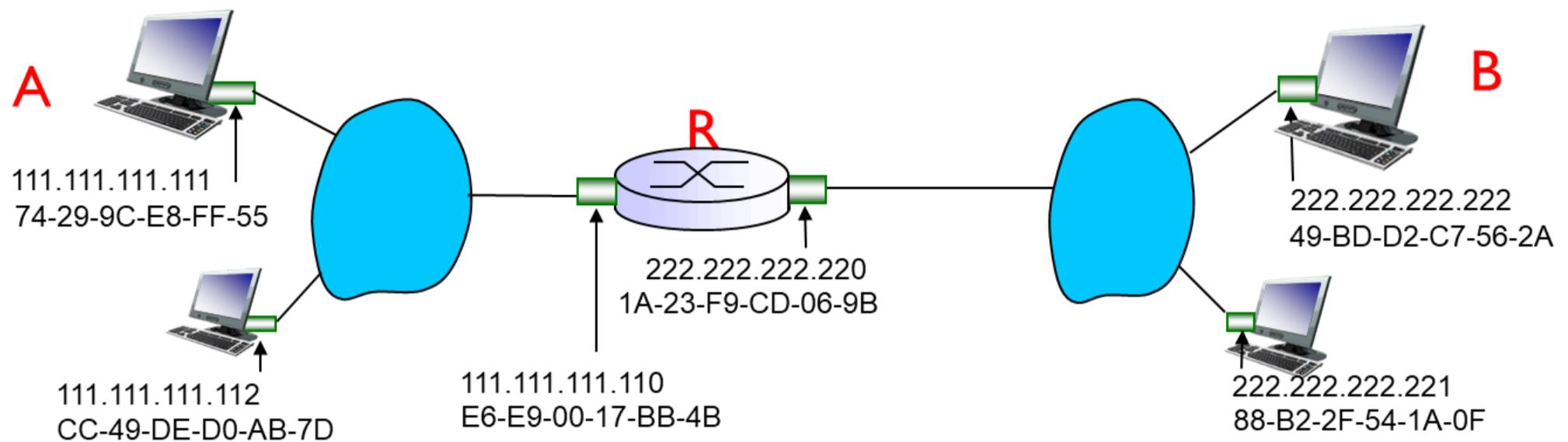
如果A已知B的IP地址, 其ARP表内不存在B的MAC地址, 想要得知其MAC地址以发送数据, 则:

1. 广播包含B的IP的ARP查询报文, 广播的目的MAC为FF-FF-FF-FF-FF-FF

2. B接收到后将其MAC地址返回给A

ARP协议不适用于跨局域网的情况

跨局域网的情况下需要连接两个子网的路由器协助



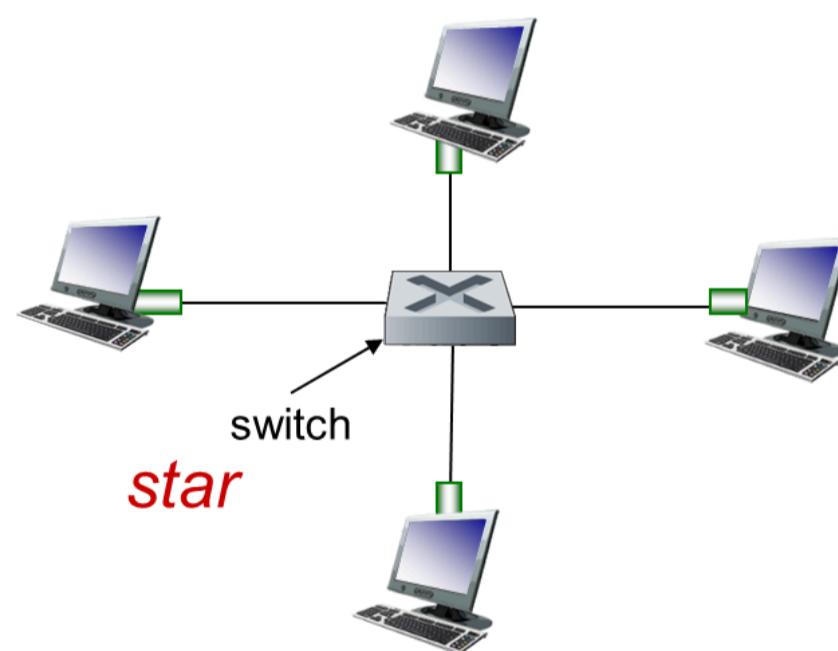
假设A需要发送到B，则：

1. A创建IP数据报，源IP为A，目标IP为B。再创建链路层帧，源MAC为A，目标MAC为R1（使用ARP）
2. R接收，丢弃链路层帧内容，将剩余内容上交给网络层
3. R转发数据包（从R1到R2），源IP为A，目标IP为B，再创建链路层帧，源MAC为R2，目标MAC为B（使用ARP）
4. B成功接收

## Ethernet

现在的局域网通常通过以太网技术实现

中心使用交换机，周围的节点不会相互碰撞



以太网帧：



前同步码（Preamble）8字节：用于同步时钟

源MAC地址，目的MAC地址各6字节：若适配器adapter接收到了不以自身为目标的帧，则丢弃（广播除外）

类型 2字节：指示更高层的协议

CRC：检测是否有差错

以太网具有以下特点：

- 无连接：不需要握手
- 不可靠：不发送ACK，只有更高层使用可靠发送协议比如TCP时，才会尝试恢复数据，否则直接丢弃出错帧

以太网使用无时隙CSMA/CD协议及二进制退避机制（unslotted CSMA/CD with binary backoff）

## 交换机

交换机是链路层设备，发挥主动作用，存储和转发以太网帧（局域网内部）

透明：主机不知道交换机存在

自学习：不需要配置

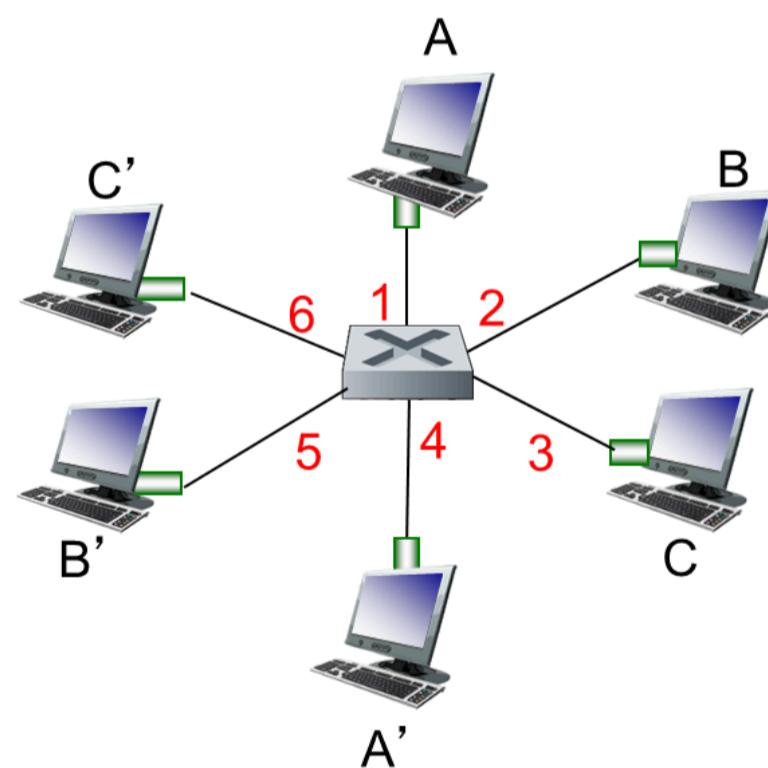
### 多个同步传输

各个主机具有专用的直接连接到交换机

各个传入链路使用无冲突的以太网协议

全双工：各交换机接口既可接收也可发送

切换：A-to-A和B-to-B可同时进行，无冲突



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

### 交换机表

交换机内部维护一个交换机表，以此可以得知通过哪个接口可以到达哪个主机

地址	接口	时间
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
...	...	...

## 自学习

交换机不需要维护，它在接收到主机发送的帧时自行记录主机的MAC地址对应哪个接口

## 转发和过滤

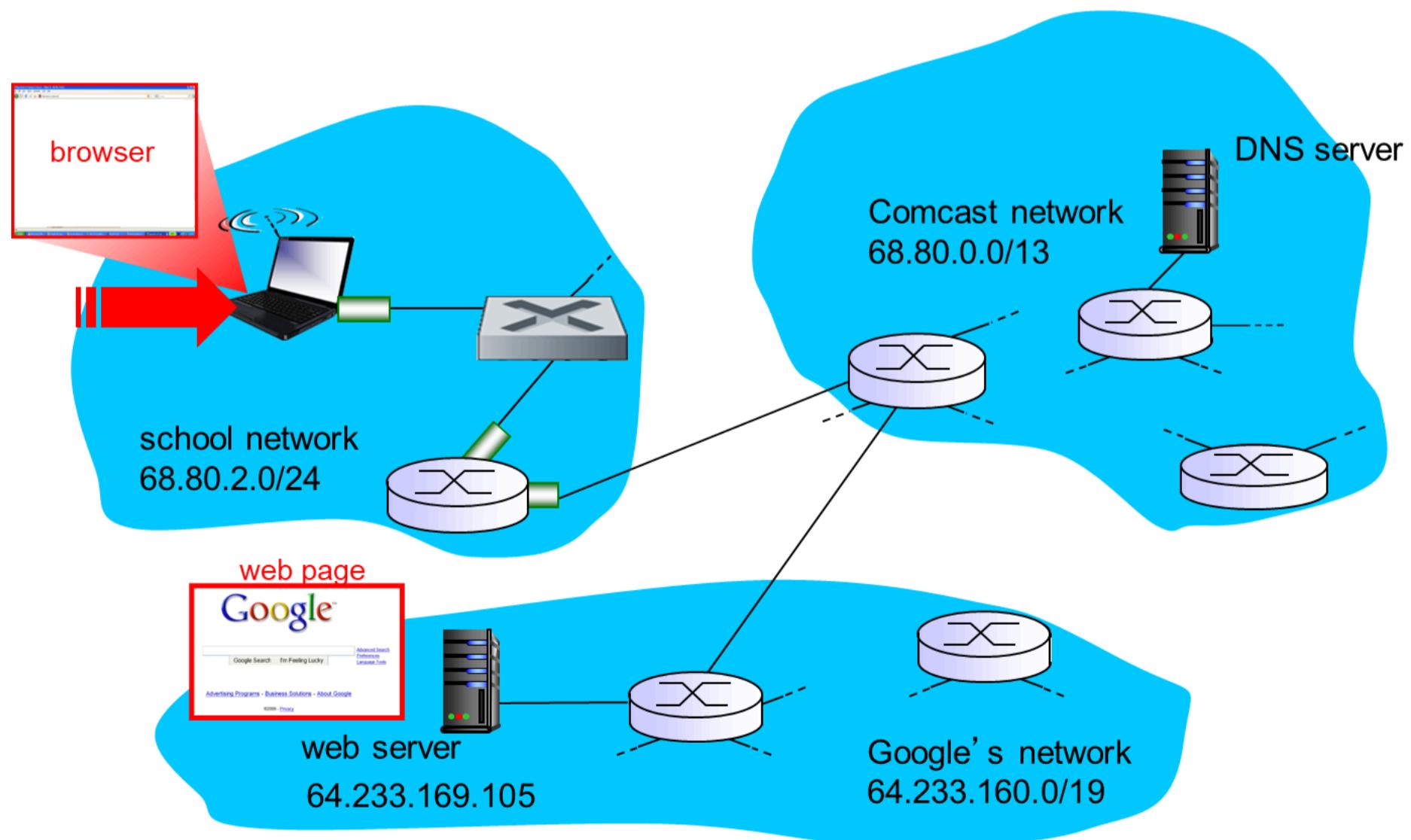
当交换机接收到帧时，其

1. 首先会记录下发送主机的MAC地址和对应接口x
2. 随后查询交换机表
3. 若帧的目标地址在交换机表内，则：
  1. 若表内记录的接口y=x，丢弃以执行过滤功能
  2. 若y与x不等，则转发到对应接口y
4. 若目标地址不在表内，则广播除x之外的接口

## 网络请求的完整过程

假想一个学术将笔记本电脑连入校园网，请求[www.google.com](http://www.google.com)的过程

整个过程涉及应用层，传输层，网络层，链路层



## 准备工作：DHCP, UDP, IP, Ethernet

1. 尝试使用DHCP获取自身IP, DNS地址, 第一跳路由器地址
2. 生成的DHCP请求报文被层层封装, UDP, IP, 以太网
3. 在局域网内通过交换机广播该报文 (目标FFFFFFFFFFFF) , 被运行DHCP服务的路由器接收
4. 层层分解
5. DHCP服务器生成DHCP ACK报文, 包含IP, DNS地址, 第一跳路由器地址
6. 封装并通过交换机在局域网中发送 (这一过程中交换机自学习)
7. 用户分解并接收, 由此得知自身IP, DNS地址, 第一跳路由器地址

## 准备工作：ARP

首先需要通过DNS得知google的IP地址

1. 创建DNS查询, 但尽管已知DNS的IP, 首先需要发送给路由器, 而并不知道路由器的MAC地址, 需要ARP协议
2. 广播ARP请求, 路由器接收到后回复自身的MAC地址
3. 用户得知第一跳路由器的MAC地址后, 可以正式进行DNS查询

## 准备工作：DNS

正式发起DNS查询

1. 通过交换机发送DNS查询数据包到达第一跳路由器
2. 路由器根据路由表发送到Comcast网络的网关路由器
3. 网关路由器接收到后, 查看目标IP地址, 根据其内的路由表 (由域内协议如OSPF, RIP, ISIS或域间协议BGP得到) 转发到DNS服务器
4. DNS服务器回复google的IP地址

## 正式连接：TCP

要发送HTTP请求, 首先需要建立连接

1. 用户生成连接Socket, 设置SYN=1 (握手第一步)
2. google服务器回复ACK (握手第二步)
3. 用户回复ACK (握手第三步)
4. 连接建立

## 正式连接：HTTP

1. 将HTTP请求放入TCP Socket, 生成HTTP请求IP数据包
2. google服务器生成并发送HTTP回复
3. 用户从HTTP回复中读取页面数据, 页面终于得到显示

# 7. Wireless and Mobile Networks

目前无线移动用户越来越多, 如何支持此类设备不受限制地访问互联网

# Wireless Link

相比起有线连接：

- 信号弱
- 干扰多
- 多路径传播

由此产生隐藏的终端问题 *Hidden terminal problem*, 信号衰减 *Signal attenuation*:

## CSMA/CA

传输前感应，繁忙时避免传输

但不检测碰撞，由于信号弱，无法及时接收碰撞信息，因此尽力避免碰撞

1. 检测是否空闲，如果不空闲则等待，如果空闲，仍等待一个随机时间后再发送

2. 发送后等待接收方的ACK，若没有收到则认为丢失，重新开始上述过程

此外，可以通过RTS（短请求发送）控制帧以及CTS（短允许请求）控制帧来控制，用于指示A可以发送，B不可以发送

