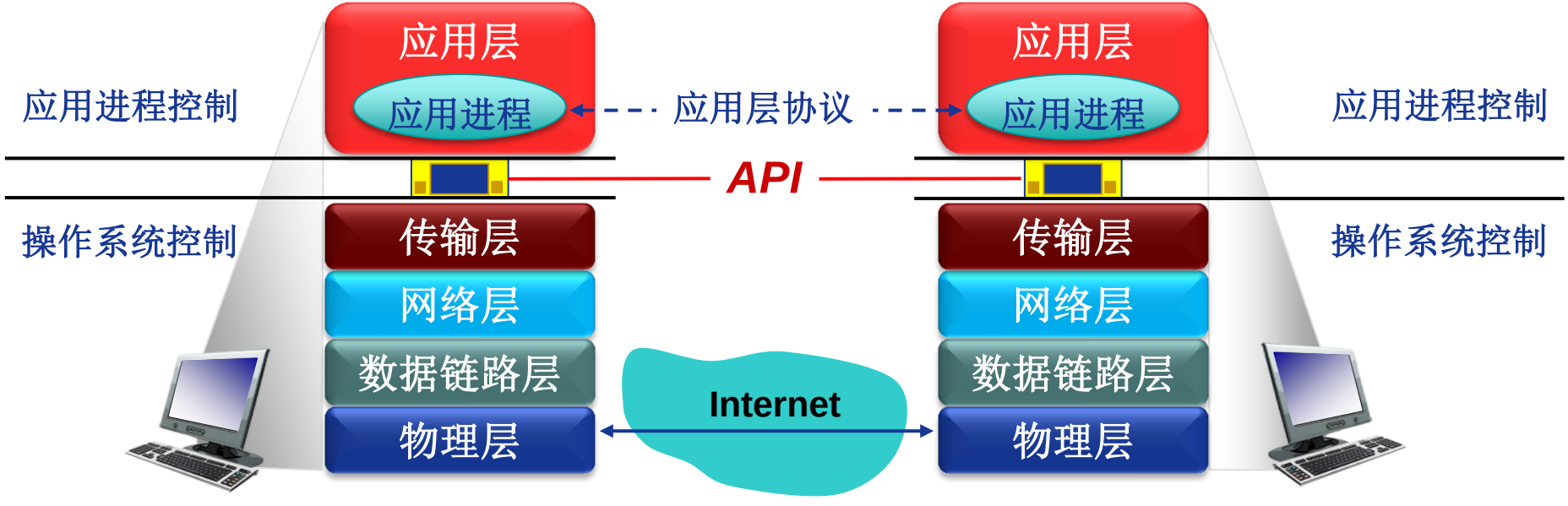
本讲主题

Socket编程-应用编程接口（API）

应用编程接口 API

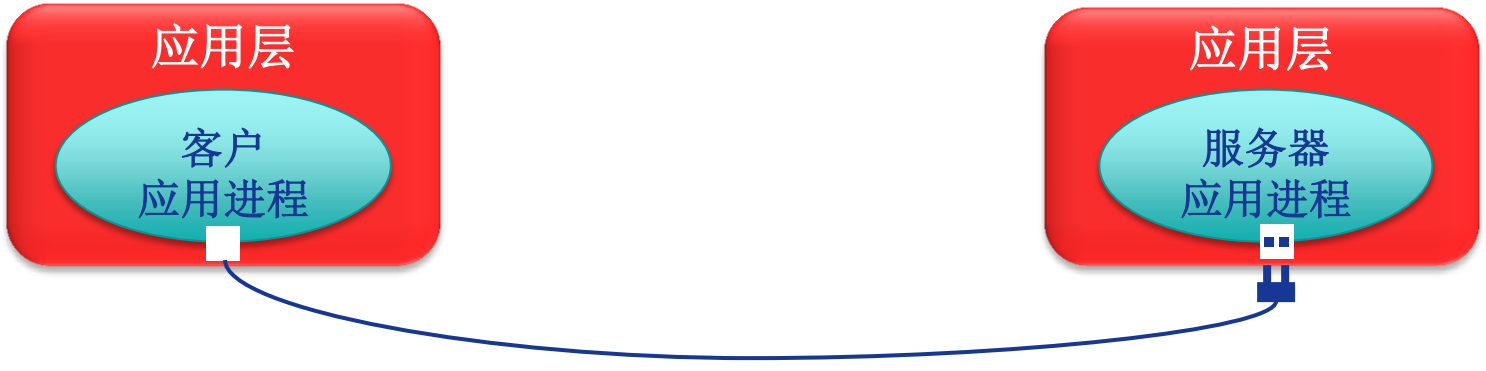
应用编程接口 API (Application Programming Interface)



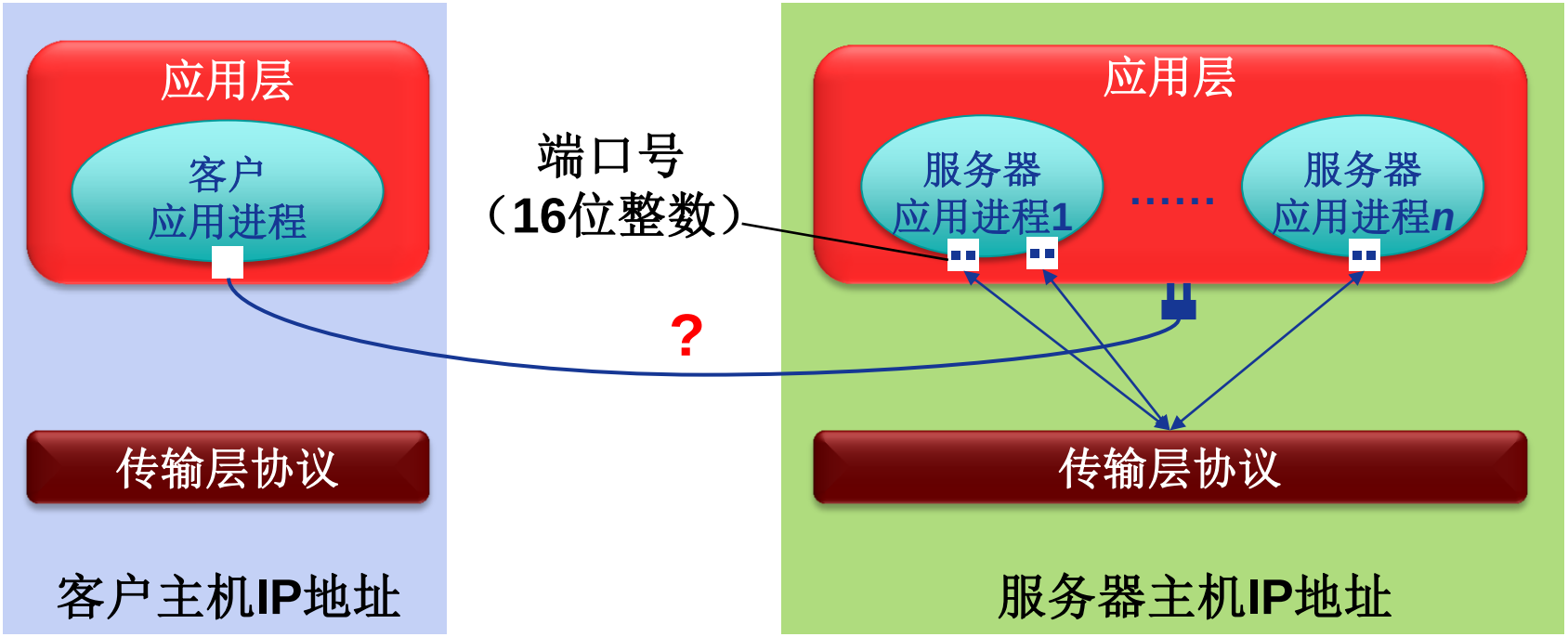
应用编程接口API:就是应用进程的控制权和操作系统的控制权进行转换的一个系统调用接口.

Socket API

- 最初设计
  - 面向BSD UNIX-Berkley
  - 面向TCP/IP协议栈接口
- 目前
  - 事实上的工业标准
  - 绝大多数操作系统都支持
- Internet网络应用最典型的API接口
- 通信模型
  - 客户/服务器（C/S）
- 应用进程间通信的抽象机制



Socket API

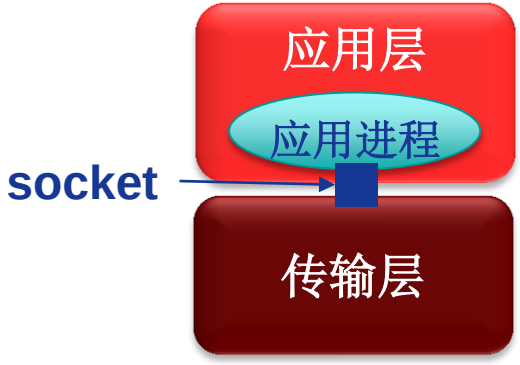


- 标识通信端点（对外）：
  - IP地址+端口号
- 操作系统/进程如何管理套接字（对内）？
  - 套接字描述符（socket descriptor）
    - 小整数

Socket API函数

socket

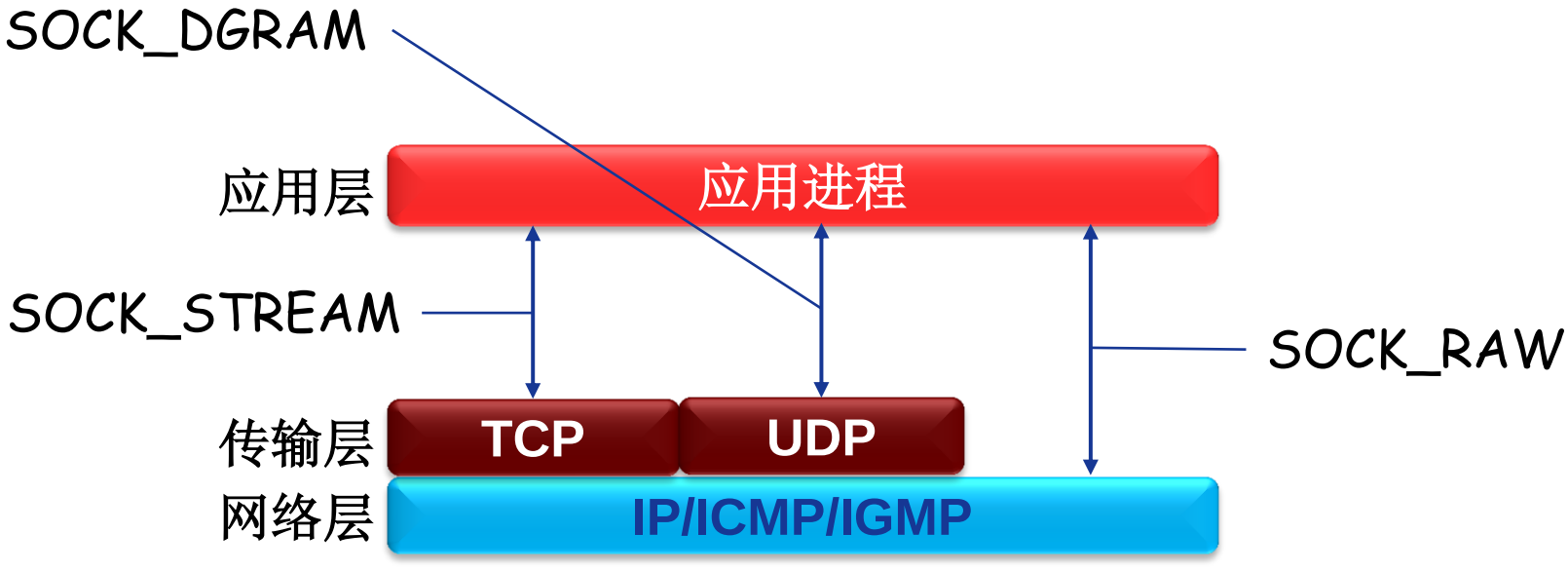
sd = socket(protofamily,type,proto);



- 创建套接字
- 操作系统返回套接字描述符（*sd*）
- 第一个参数(协议族): protofamily = PF_INET（TCP/IP）
- 第二个参数(套接字类型):
  - type = SOCK_STREAM,SOCK_DGRAM or SOCK_RAW（TCP/IP）
- 第三个参数(协议号):0为默认
- 例：创建一个流套接字的代码段

```
struct protoent *p;
p=getprotobyname("tcp");
SOCKET sd=socket(PF_INET,SOCK_STREAM,p->p_proto);
```

Socket面向TCP/IP的服务类型



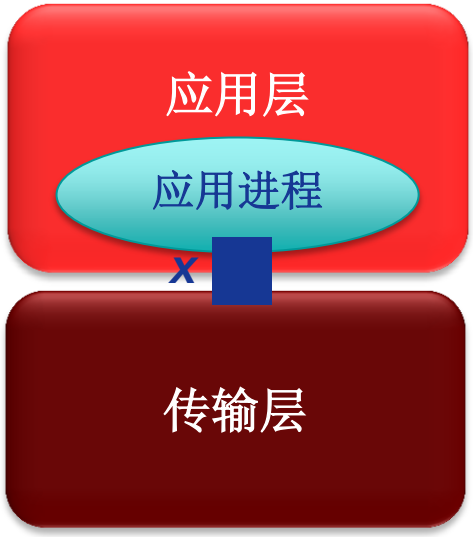
- **TCP**：可靠、面向连接、字节流传输、点对点
- **UDP**：不可靠、无连接、数据报传输

Socket API函数

bind

```
int bind(sd,localaddr,addrlen);
```

- 绑定套接字的本地端点地址
  - IP地址+端口号
- 参数:
  - 套接字描述符（sd）
  - 端点地址（localaddr）
    - 结构*sockaddr_in*
- 客户程序一般不必调用bind函数
- 服务器端？
  - 熟知端口号
  - IP地址？

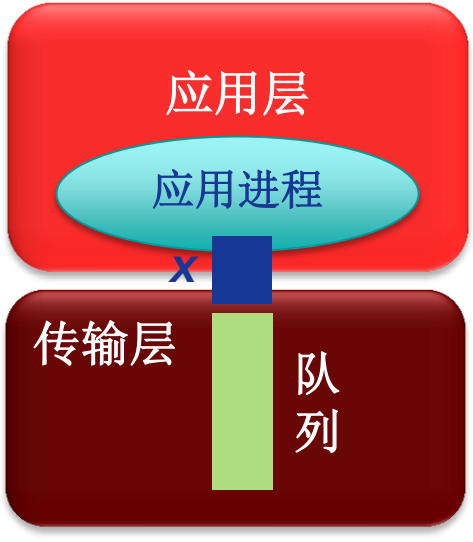


Socket API函数

listen

```
int listen(sd,queuesize);
```

- 置服务器端的流套接字处于监听状态
  - 仅服务器端调用
  - 仅用于面向连接的流套接字
- 设置连接请求队列大小（queuesize）
- 返回值：
  - 0：成功
  - SOCKET_ERROR：失败

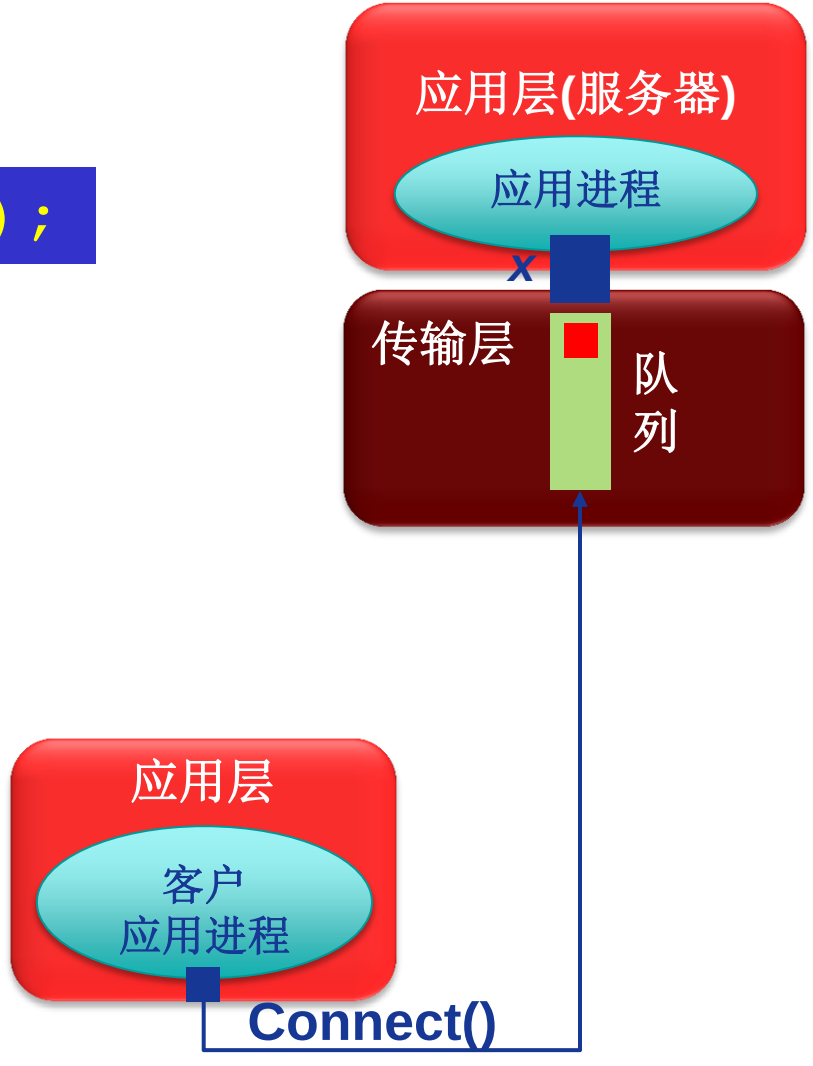


Socket API函数

connect

```
connect(sd,saddr,saddrlen);
```

- 客户程序调用connect函数来使客户套接字（sd）与特定计算机的特定端口（saddr）的套接字（服务）进行连接
- 仅用于客户端
- 可用于TCP客户端也可以用于UDP客户端
  - TCP客户端：建立TCP连接
  - UDP客户端：指定服务器端点地址

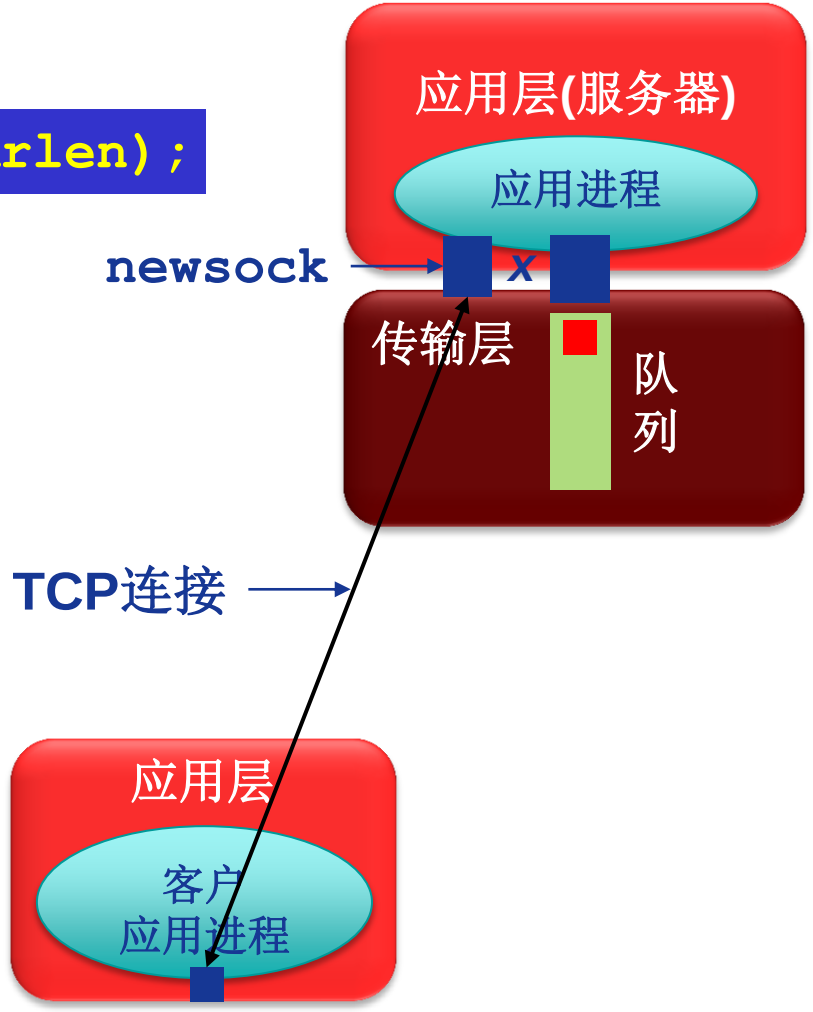


Socket API函数

accept

```
newsock = accept(sd,caddr,caddrlen);
```

- 服务程序调用accept函数从处于监听状态的流套接字sd的客户连接请求队列中取出排在最前的一个客户请求，并且创建一个新的套接字来与客户套接字创建连接通道
  - 仅用于TCP套接字
  - 仅用于服务器
- 利用新创建的套接字（newsock）与客户通信



Socket API函数

send, sendto

```
send(sd,*buf,len,flags);
```

```
sendto(sd,*buf,len,flags,destaddr,addrlen);
```

- send函数TCP套接字（客户与服务器）或调用了connect函数的UDP客户端套接字
- sendto函数用于UDP服务器端套接字与未调用connect函数的UDP客户端套接字

Socket API函数

recv, recvfrom

```
recv(sd,*buffer,len,flags);
```

```
recvfrom(sd,*buf,len,flags,senderaddr,saddrlen);
```

- recv函数从TCP连接的另一端接收数据，或者从调用了connect函数的UDP客户端套接字接收服务器发来的数据
- recvfrom函数用于从UDP服务器端套接字与未调用connect函数的UDP客户端套接字接收对端数据

本讲主题

Socket编程-客户端软件设计

TCP客户端软件流程

1. 确定服务器IP地址与端口号
2. 创建套接字
3. 分配本地端点地址（IP地址+端口号）
4. 连接服务器（套接字）
5. 遵循应用层协议进行通信
6. 关闭/释放连接

UDP客户端软件流程

1. 确定服务器IP地址与端口号
2. 创建套接字
3. 分配本地端点地址（IP地址+端口号）
4. 指定服务器端点地址，构造UDP数据报
5. 遵循应用层协议进行通信
6. 关闭/释放套接字

本讲主题

Socket编程-服务器软件设计

4种类型基本服务器

- 循环无连接(Iterative connectionless)服务器
- 循环面向连接(Iterative connection-oriented)服务器
- 并发无连接(Concurrent connectionless)服务器
- 并发面向连接(Concurrent connection-oriented)服务器

循环无连接服务器基本流程

1. 创建套接字
2. 绑定端点地址（INADDR_ANY+端口号）
3. 反复接收来自客户端的请求
4. 遵循应用层协议，构造响应报文，发送给客户

循环面向连接服务器基本流程

1. 创建（主）套接字，并绑定熟知端口号；
2. 设置（主）套接字为被动监听模式，准备用于服务器；
3. 调用***accept()***函数接收下一个连接请求（通过主套接字），创建新套接字用于与该客户建立连接；
4. 遵循应用层协议，反复接收客户请求，构造并发送响应(通过新套接字)；
5. 完成为特定客户服务后，关闭与该客户之间的连接，返回步骤3.

本讲主题

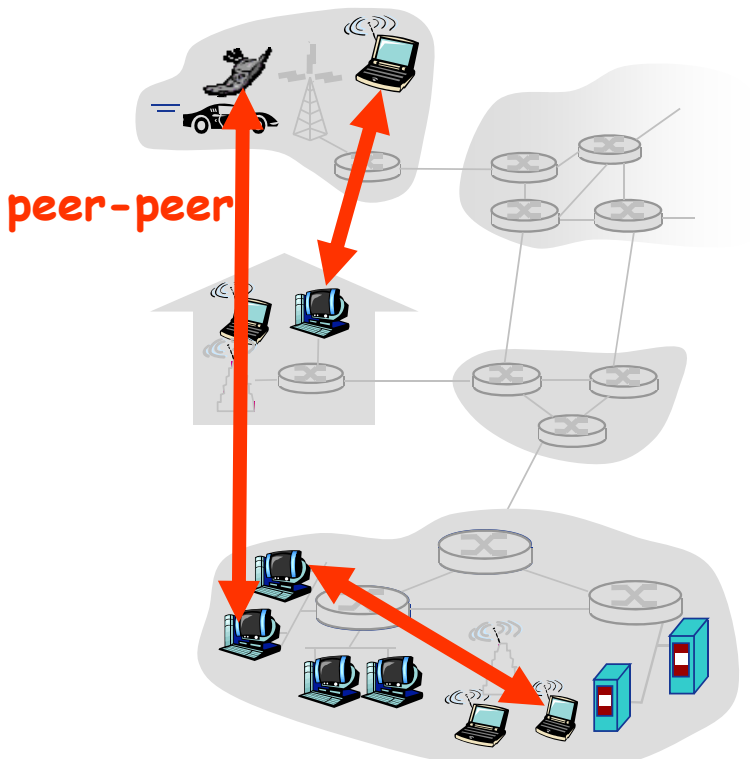
P2P应用：原理与文件分发

纯P2P架构

- Peer-to-peer

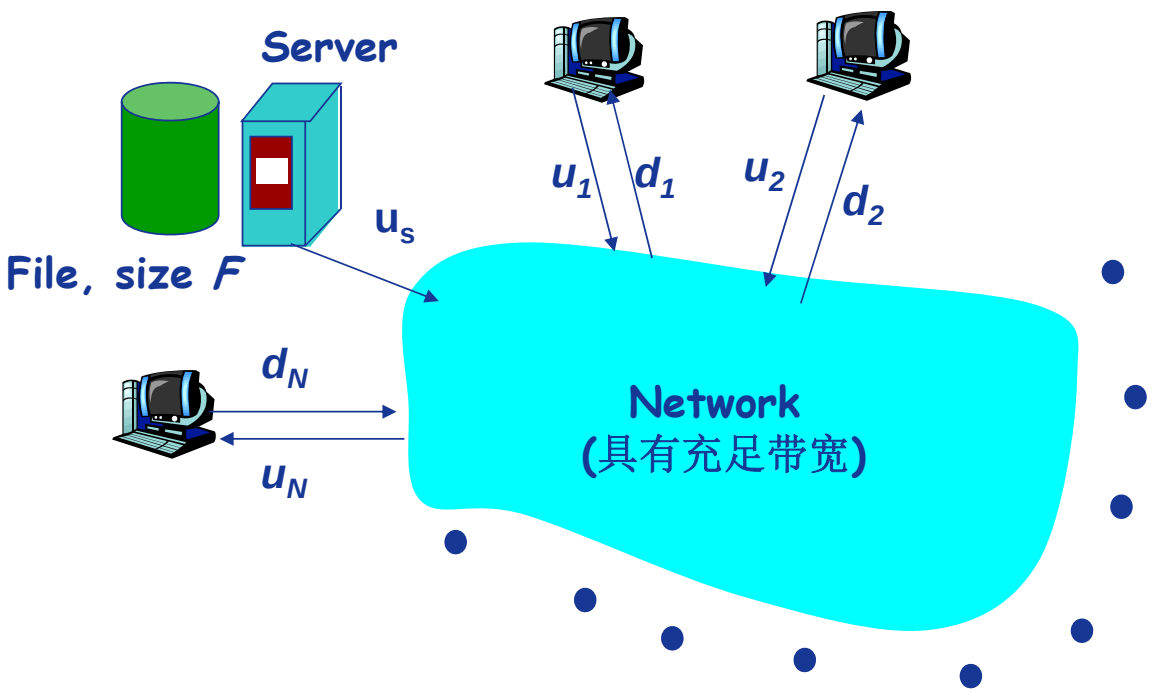
- 没有服务器
- 任意端系统之间直接通信
- 节点阶段性接入Internet
- 节点可能更换IP地址

- 以具体应用为例讲解



文件分发：客户机/服务器 vs. P2P

问题：从一个服务器向N个节点分发一个文件需要多长时间?



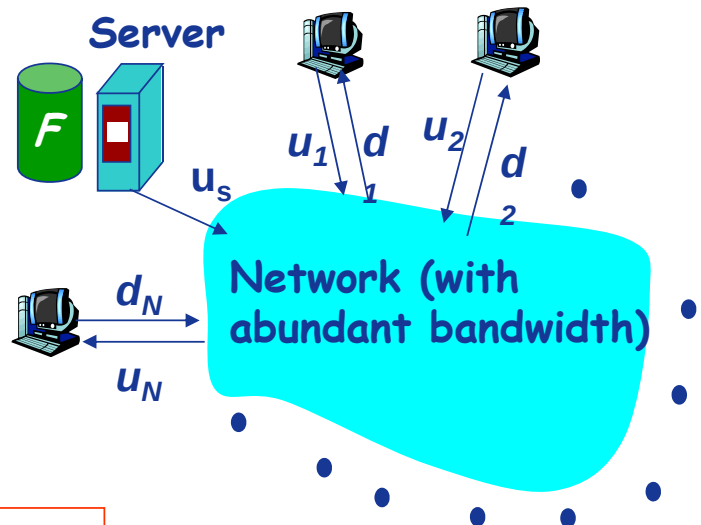
u_s: 服务器上传带宽

u_i: 节点i的上传带宽

d_i: 节点i的下载带宽

文件分发：客户机 / 服务器

- 服务器串行地发送N个副本
  - 时间： NF/u_s
- 客户机i需要F/d_i时间下载

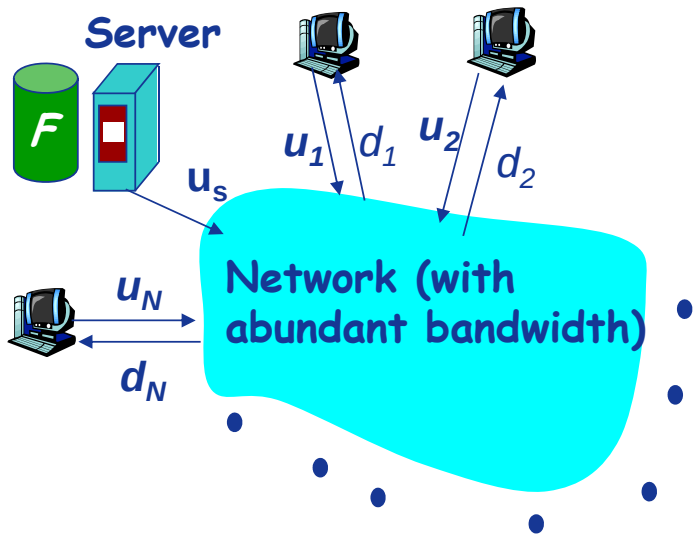


Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

文件分发：P2P

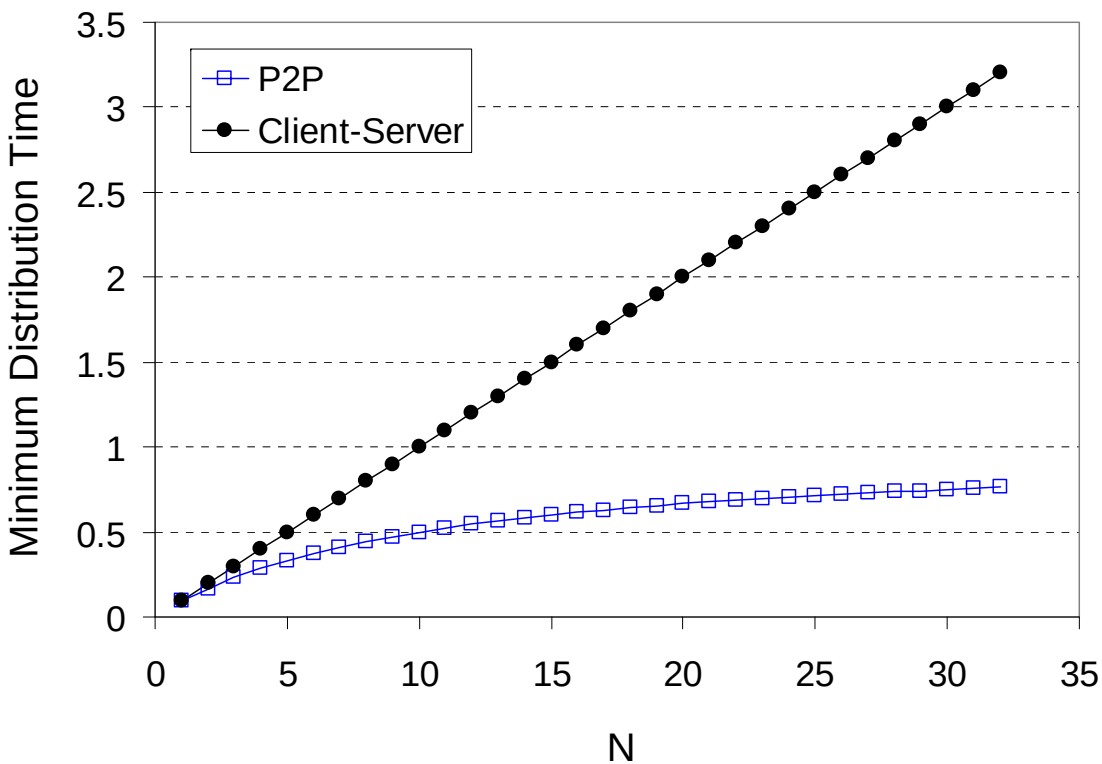
- 服务器必须发送一个副本
  - 时间： F/u_s
- 客户机i需要F/d_i时间下载
- 总共需要下载NF比特
- 最快的可能上传速率： $u_s + \Sigma u_i$



$$d_{P2P} = \max\{ F/u_s, F/\min(d_i)_i, NF/(u_s + \Sigma u_i) \}$$

客户机/服务器 vs. P2P: 例子

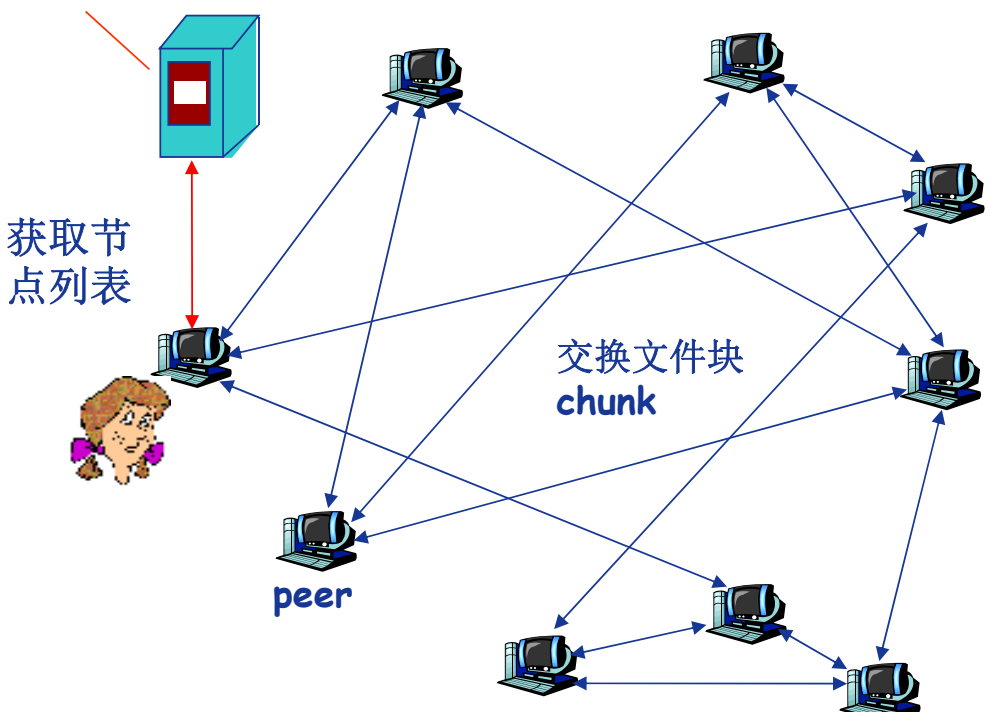
客户端上传速率= u， F/u = 1小时， u_s = 10u， $d_{min} \geq u_s$



文件分发：BitTorrent

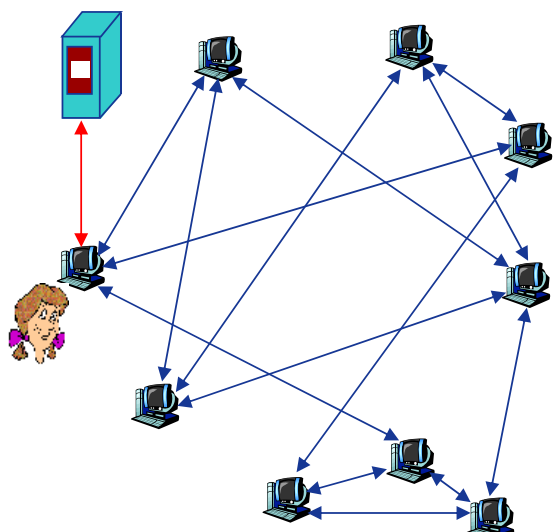
tracker: 跟踪参与torrent的节点

torrent: 交换同一个文件的文件块的节点组



BitTorrent (1)

- 文件划分为256KB的chunk
- 节点加入torrent
  - 没有chunk，但是会逐渐积累
  - 向tracker注册以获得节点清单，与某些节点（“邻居”）建立连接
- 下载的同时，节点需要向其他节点上传chunk
- 节点可能加入或离开
- 一旦节点获得完整的文件，它可能（自私地）离开或（无私地）留下

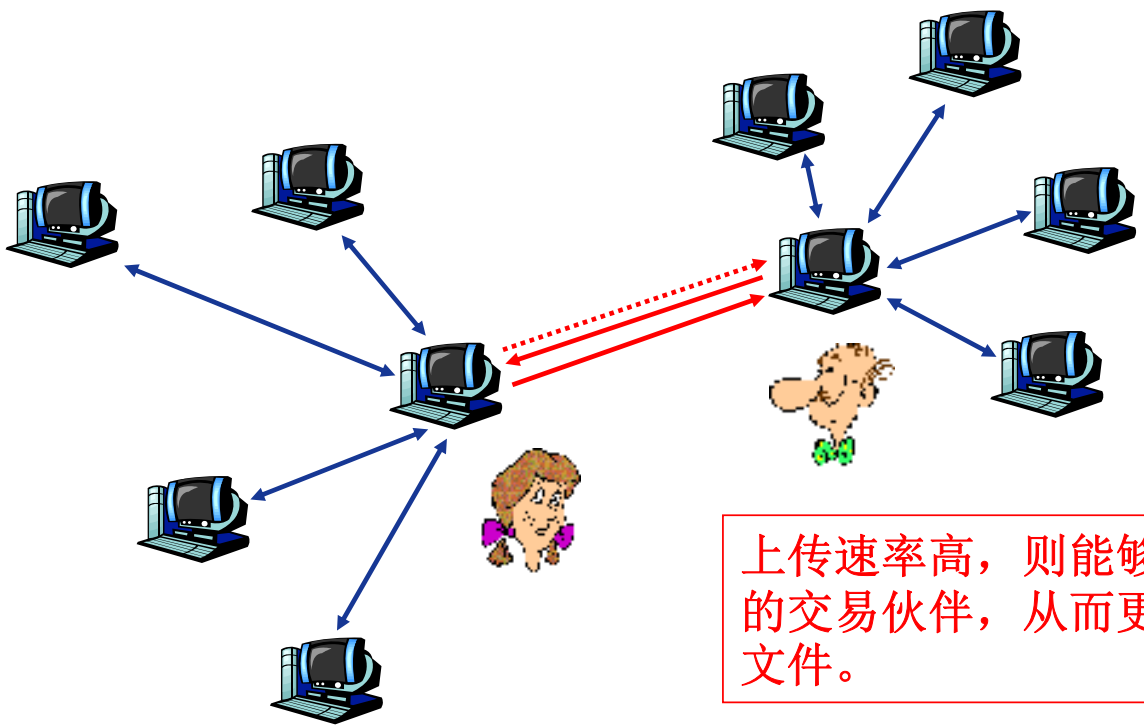


BitTorrent (2)

- 获取chunk
  - 给定任一时刻，不同的节点持有文件的不同chunk集合
  - 节点(Alice)定期查询每个邻居所持有的chunk列表
  - 节点发送请求，请求获取缺失的chunk
    - 稀缺优先
- 发送chunk: tit-for-tat
  - Alice向4个邻居发送chunk：正在向其发送Chunk，速率最快的4个
    - 每10秒重新评估top 4
  - 每30秒随机选择一个其他节点，向其发送chunk
    - 新选择节点可能加入top 4
    - “optimistically unchoke”

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



上传速率高，则能够找到更好的交易伙伴，从而更快地获取文件。

思考题

BitTorrent技术对网络性能有哪些潜在的危害？