哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY
立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：聂兰顺

本讲主题

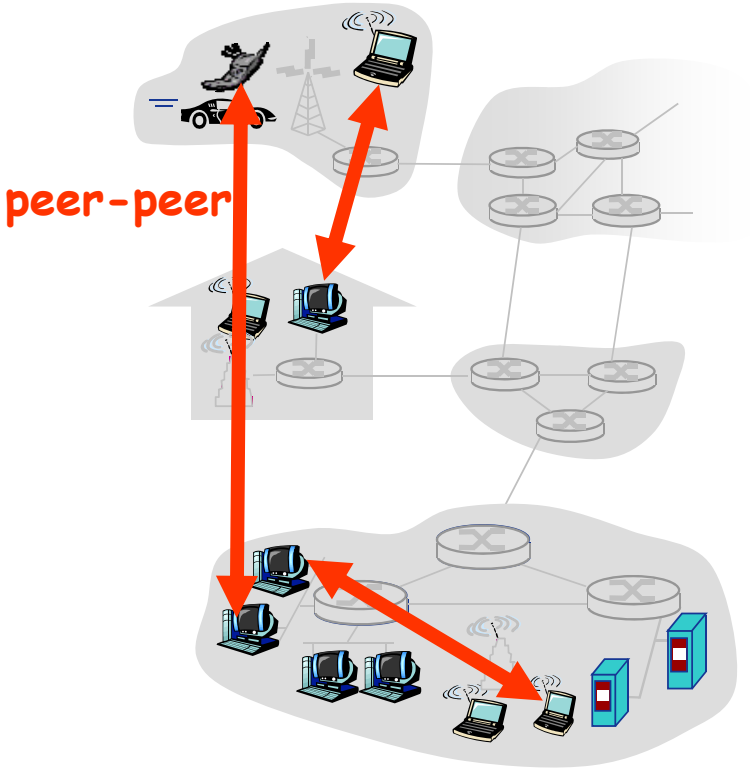
P2P应用：原理与文件分发

纯P2P架构

- Peer-to-peer

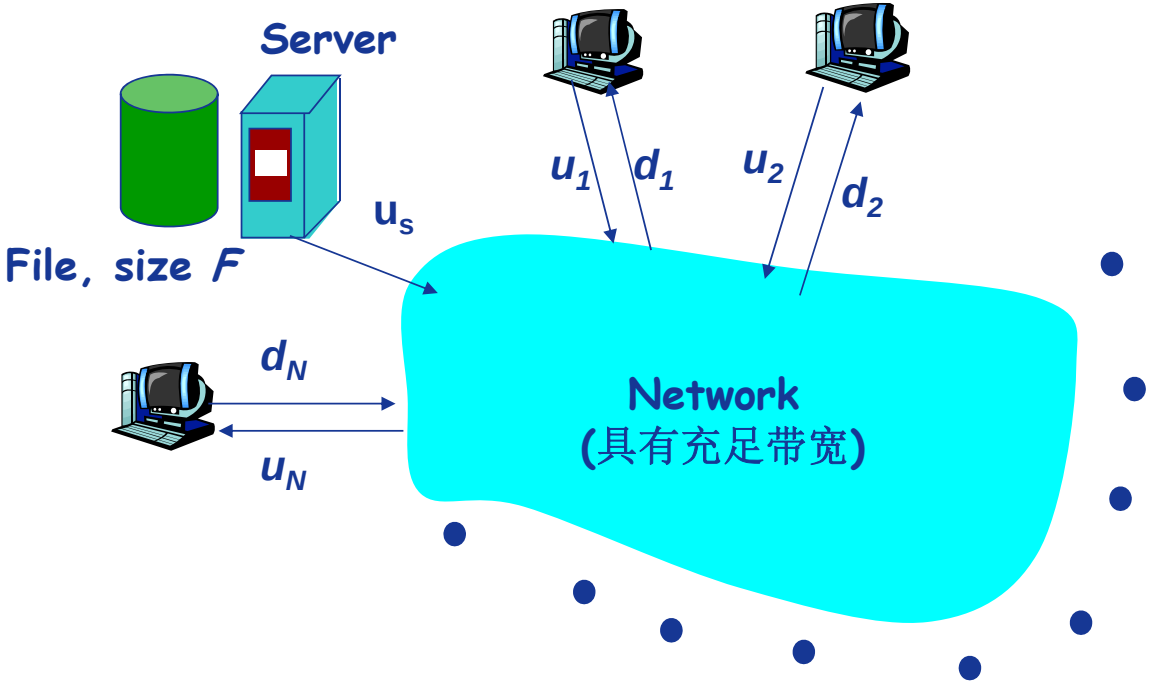
- 没有服务器
- 任意端系统之间直接通信
- 节点阶段性接入Internet
- 节点可能更换IP地址

- 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?



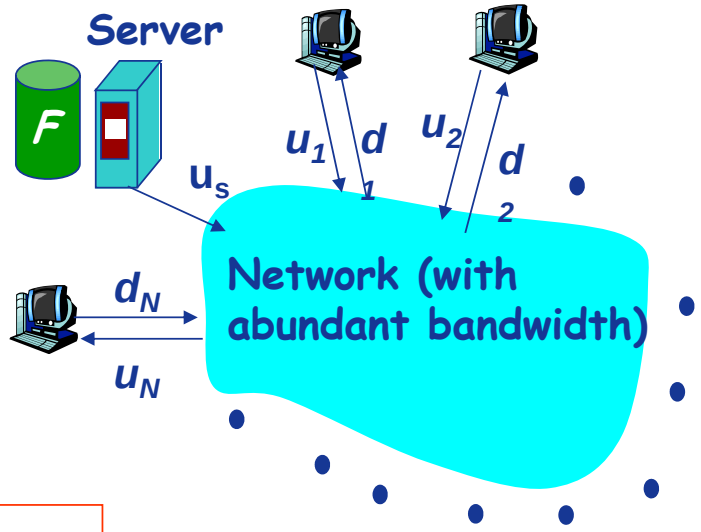
u_s: 服务器上传带宽

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

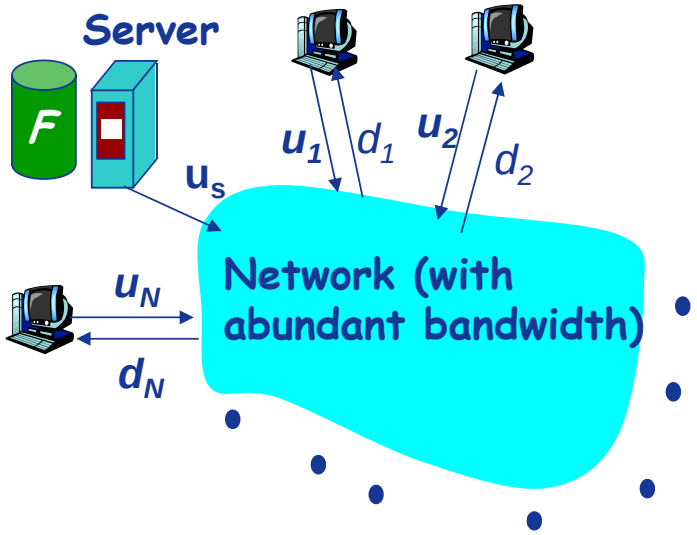


Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

文件分发：P2P

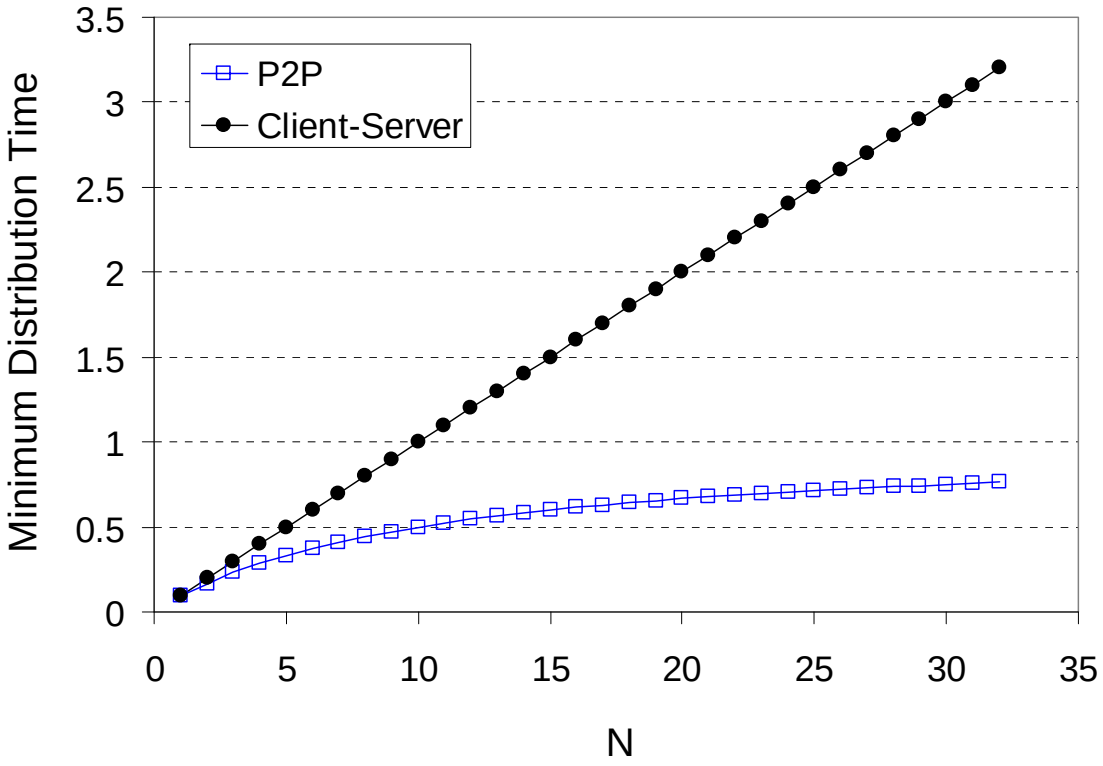
- 服务器必须发送一个副本
  - 时间： F/u_s
- 客户机i需要F/d_i时间下载
- 总共需要下载NF比特
- 最快的可能上传速率： $u_s + \Sigma u_i$



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

客户机/服务器 vs. P2P: 例子

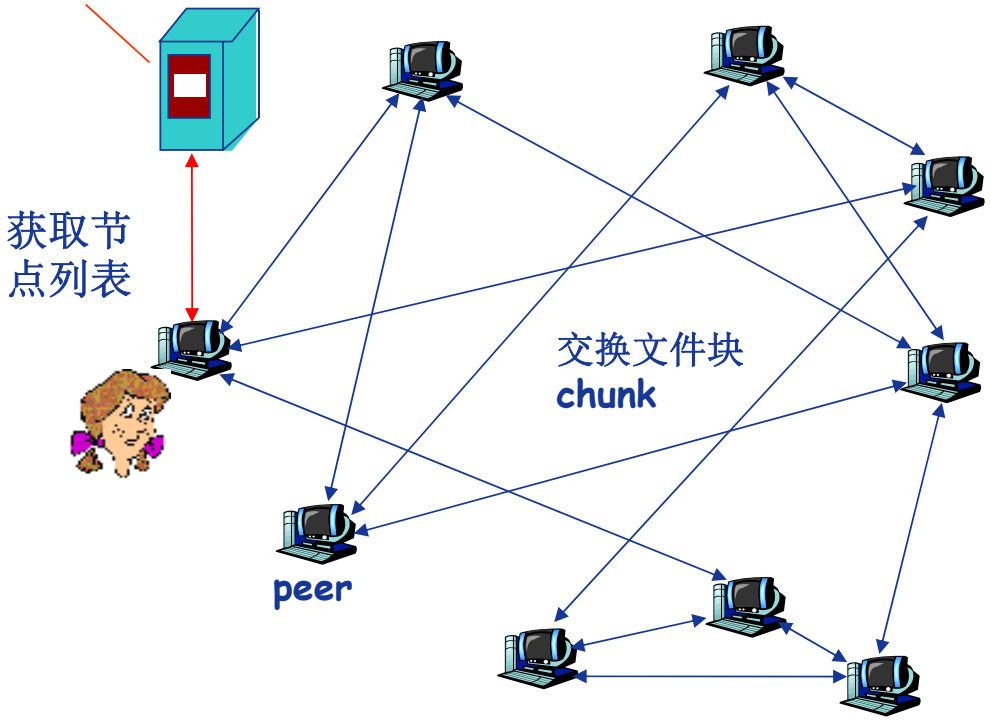
客户端上传速率= u， F/u = 1小时， u_s = 10u， $d_{min} \geq u_s$



文件分发：BitTorrent

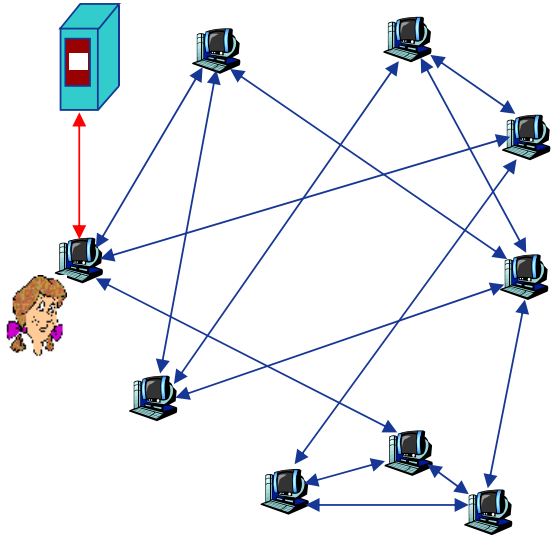
tracker: 跟踪参与torrent的节点

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

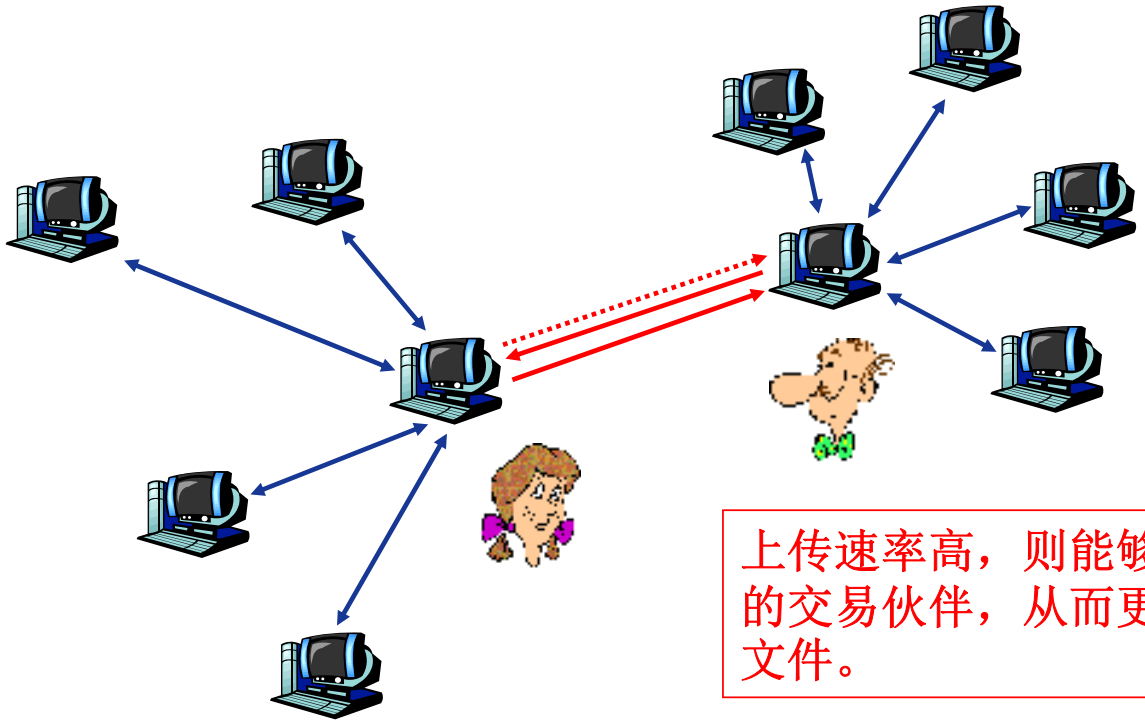


BitTorrent (2)

- 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？

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY
立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY
立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：聂兰顺

本讲主题

P2P应用：索引技术

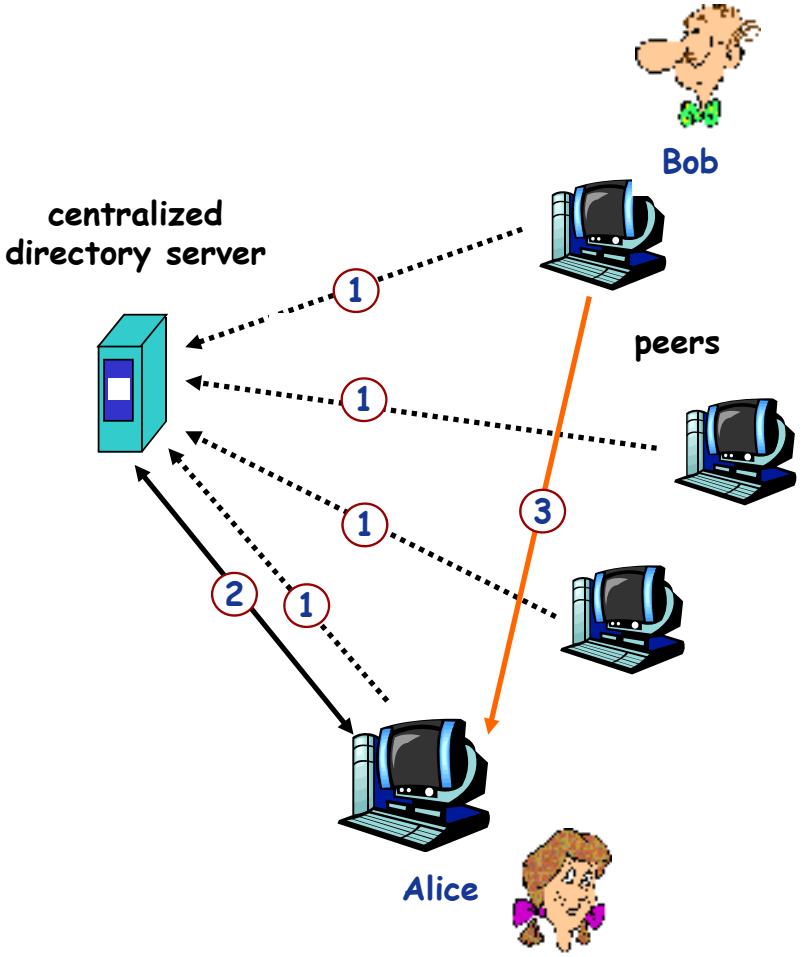
P2P: 搜索信息

- P2P系统的索引：信息到节点位置(IP地址+端口号)的映射
- 文件共享(电驴)
  - 利用索引动态跟踪节点所共享的文件的位置
  - 节点需要告诉索引它拥有哪些文件
  - 节点搜索索引，从而获知能够得到哪些文件

- 即时消息(QQ)
  - 索引负责将用户名映射到位置
  - 当用户开启IM应用时，需要通知索引它的位置
  - 节点检索索引，确定用户的IP地址

集中式索引

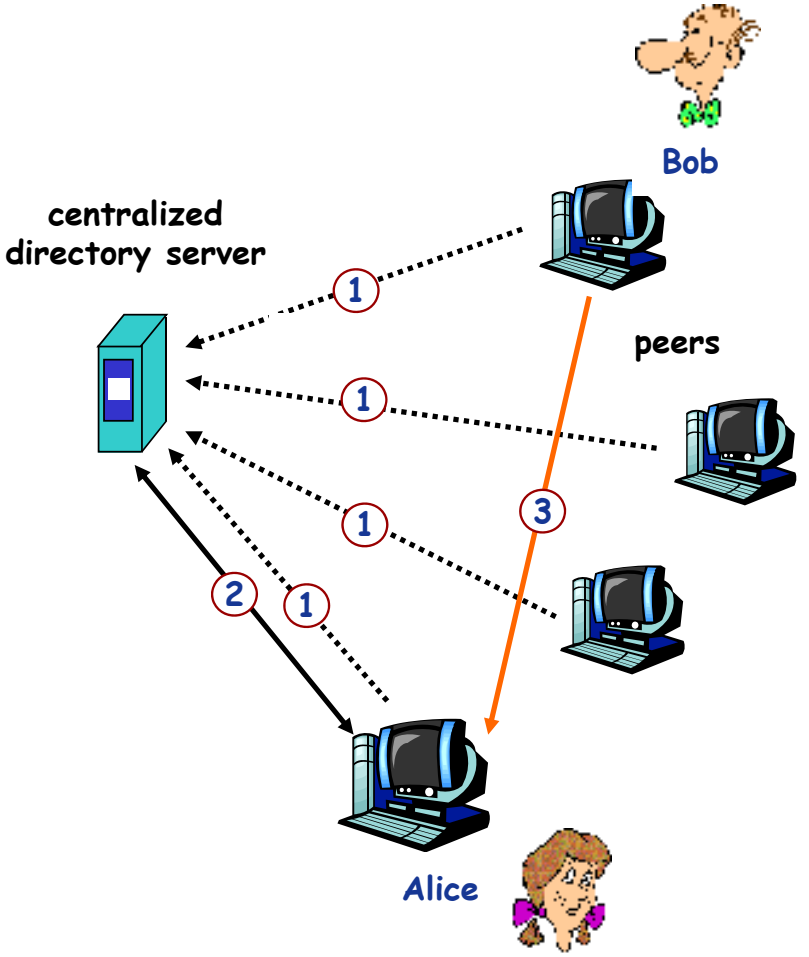
- Napster最早采用这种设计
  - 1) 节点加入时，通知中央服务器：
    - IP地址
    - 内容
  - 2) Alice查找“Hey Jude”
  - 3) Alice从Bob处请求文件



集中式索引的问题

内容和文件传输是分布式的，
但是内容定位是高度集中式的

- 单点失效问题
- 性能瓶颈
- 版权问题



洪泛式查询: Query flooding

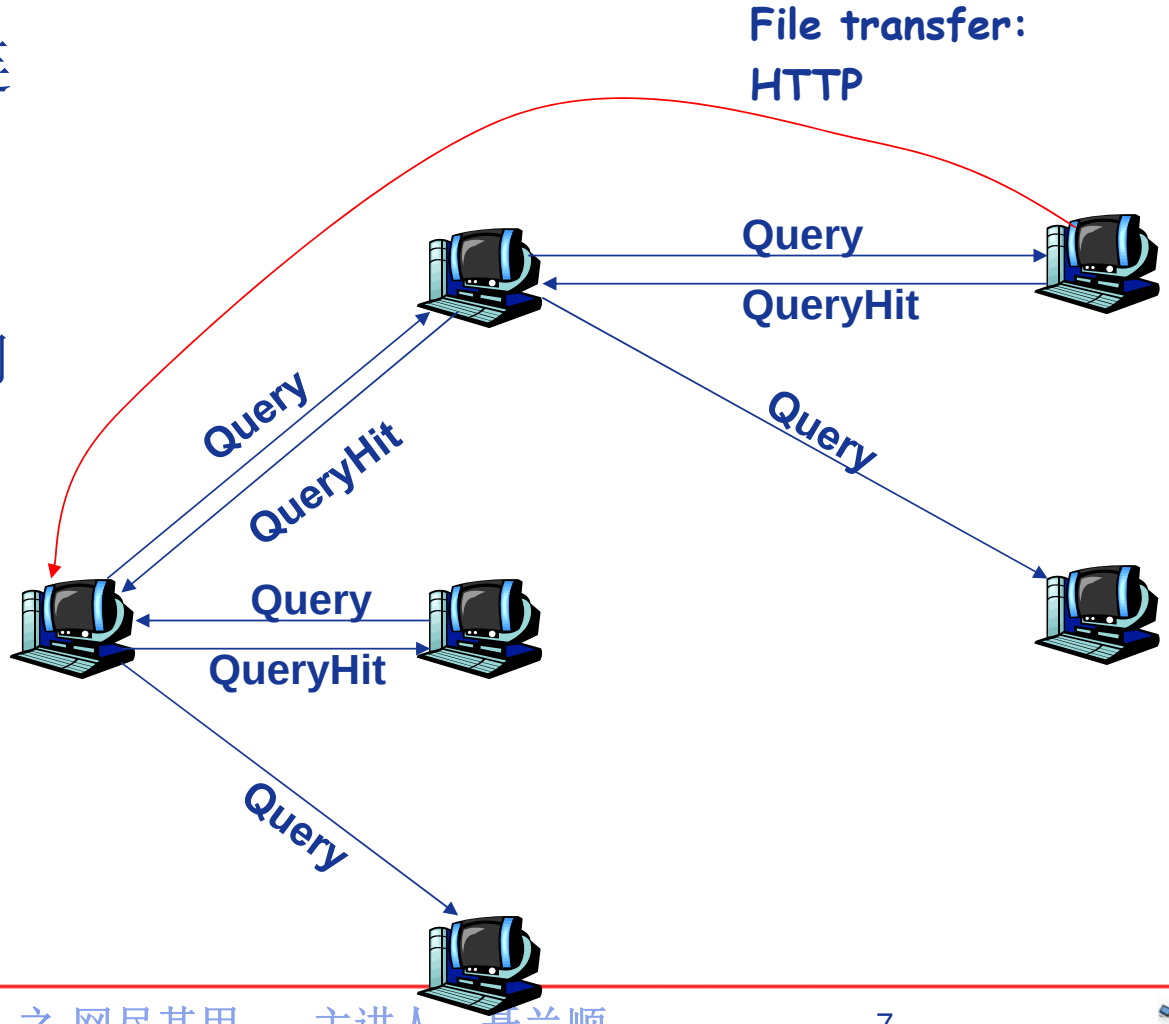
- 完全分布式架构
- Gnutella采用这种架构
- 每个节点对它共享的文件进行索引，且只对它共享的文件进行索引

覆盖网络(overlay network): Graph

- 节点X与Y之间如果有TCP连接，那么构成一个边
- 所有的活动节点和边构成覆盖网络
- 边：虚拟链路
- 节点一般邻居数少于10个

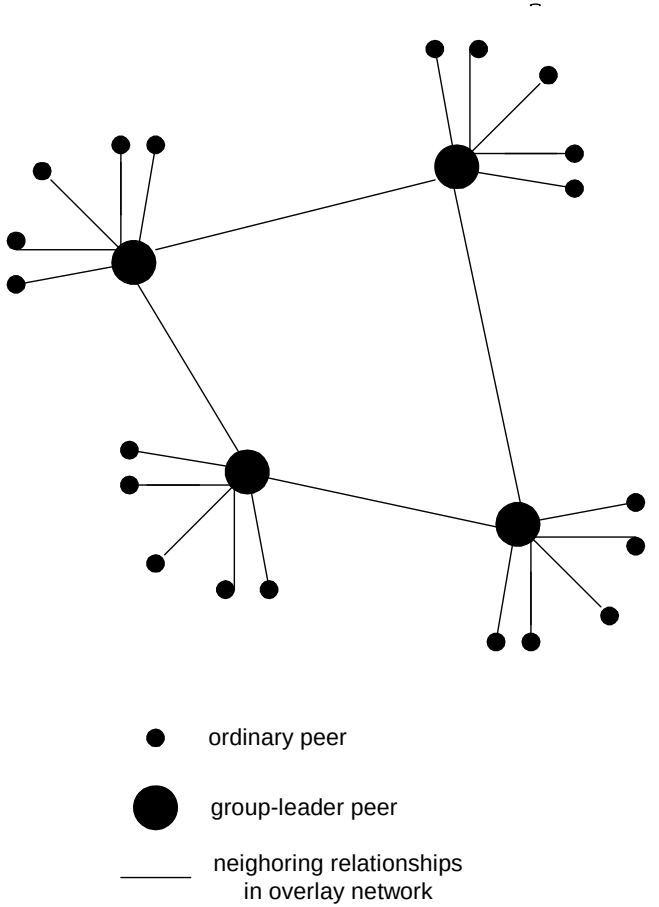
洪泛式查询: Query flooding

- 查询消息通过已有的TCP连接发送
- 节点转发查询消息
- 如果查询命中，则利用反向路径发回查询节点



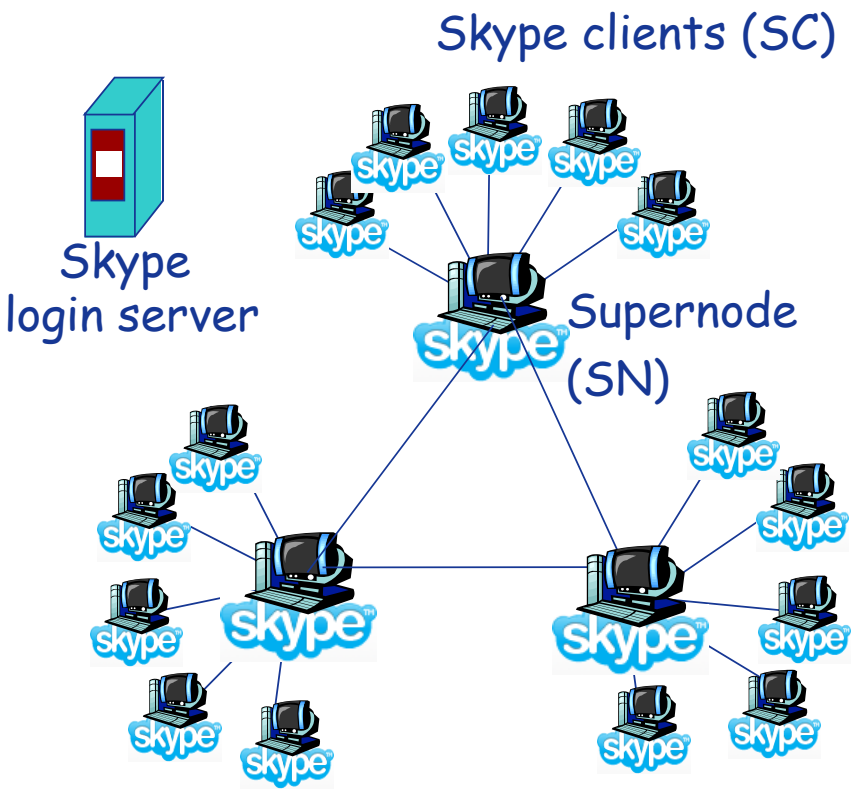
层次式覆盖网络

- 介于集中式索引和洪泛查询之间的方法
- 每个节点或者是一个超级节点，或者被分配一个超级节点
  - 节点和超级节点间维持TCP连接
  - 某些超级节点对之间维持TCP连接
- 超级节点负责跟踪子节点的内容



P2P案例应用：Skype

- 本质上是P2P的：用户/节点对之间直接通信
- 私有应用层协议
- 采用层次式覆盖网络架构
- 索引负责维护用户名与IP地址间的映射
- 索引分布在超级节点上



课后作业

查阅**Skype**应用的相关资料，就其架构、协议、算法等撰写一篇调研报告，长度在**5000**字以上。

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY
立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

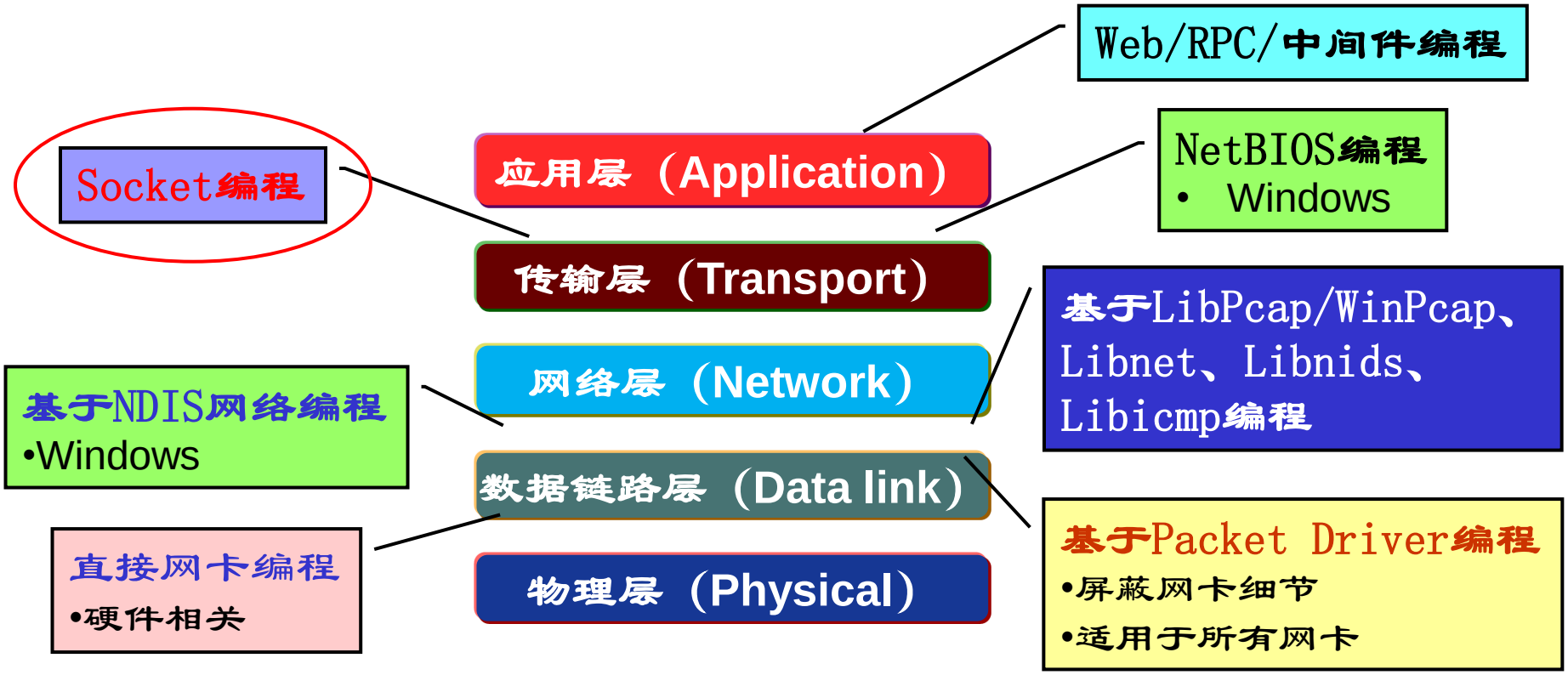
计算机网络之网尽其用

主讲人：李全龙

本讲主题

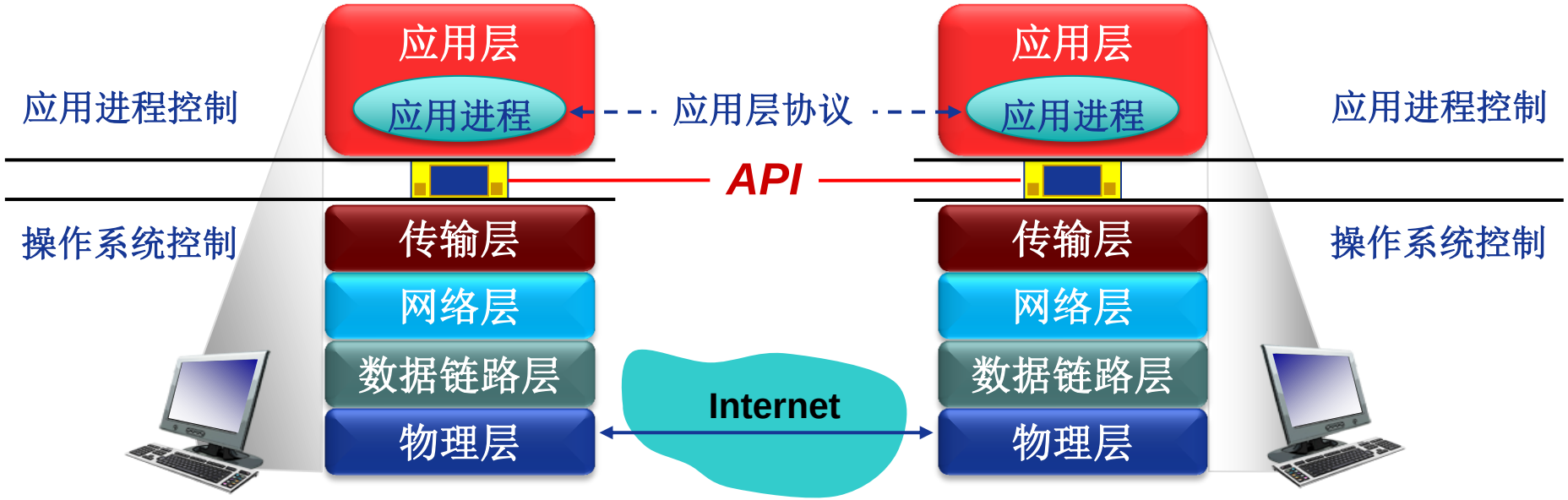
Socket编程-应用编程接口（API）

网络程序设计接口


Socket编程
应用层（Application）
传输层（Transport）
网络层（Network）
数据链路层（Data link）
物理层（Physical）
Web/RPC/中间件编程
NetBIOS编程
• Windows
基于LibPcap/WinPcap、Libnet、Libnids、Libicmp编程
基于Packet Driver编程
•屏蔽网卡细节
•适用于所有网卡
基于NDIS网络编程
•Windows
直接网卡编程
•硬件相关

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.

几种典型的应用编程接口

- Berkeley UNIX 操作系统定义了一种 API，称为套接字接口(socket interface)，简称套接字（socket）。
- 微软公司在其操作系统中采用了套接字接口 API，形成了一个稍有不同的 API，并称之为 Windows Socket Interface，WINSOCK。
- AT&T 为其 UNIX 系统 V 定义了一种 API，简写为 TLI (Transport Layer Interface)。

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

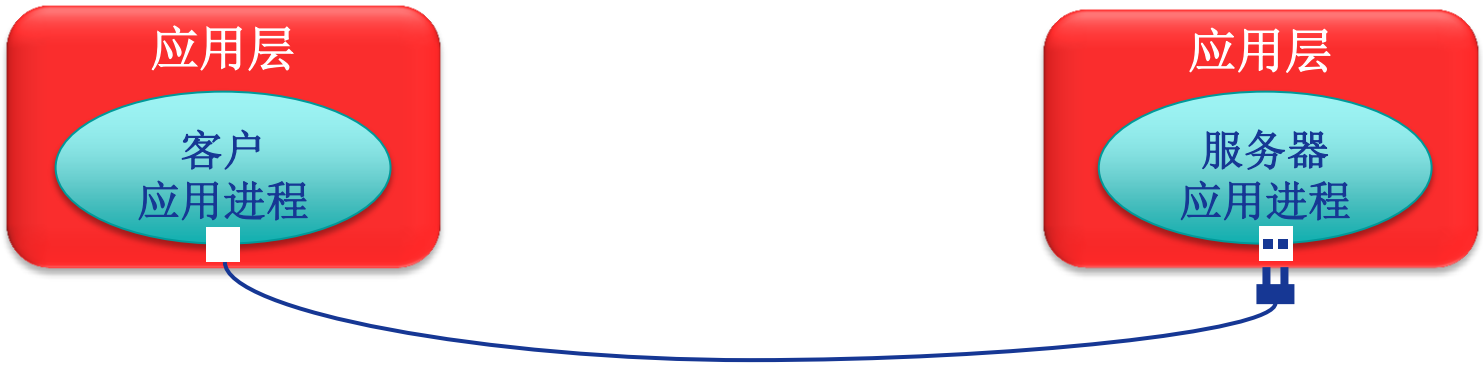
主讲人：李全龙

本讲主题

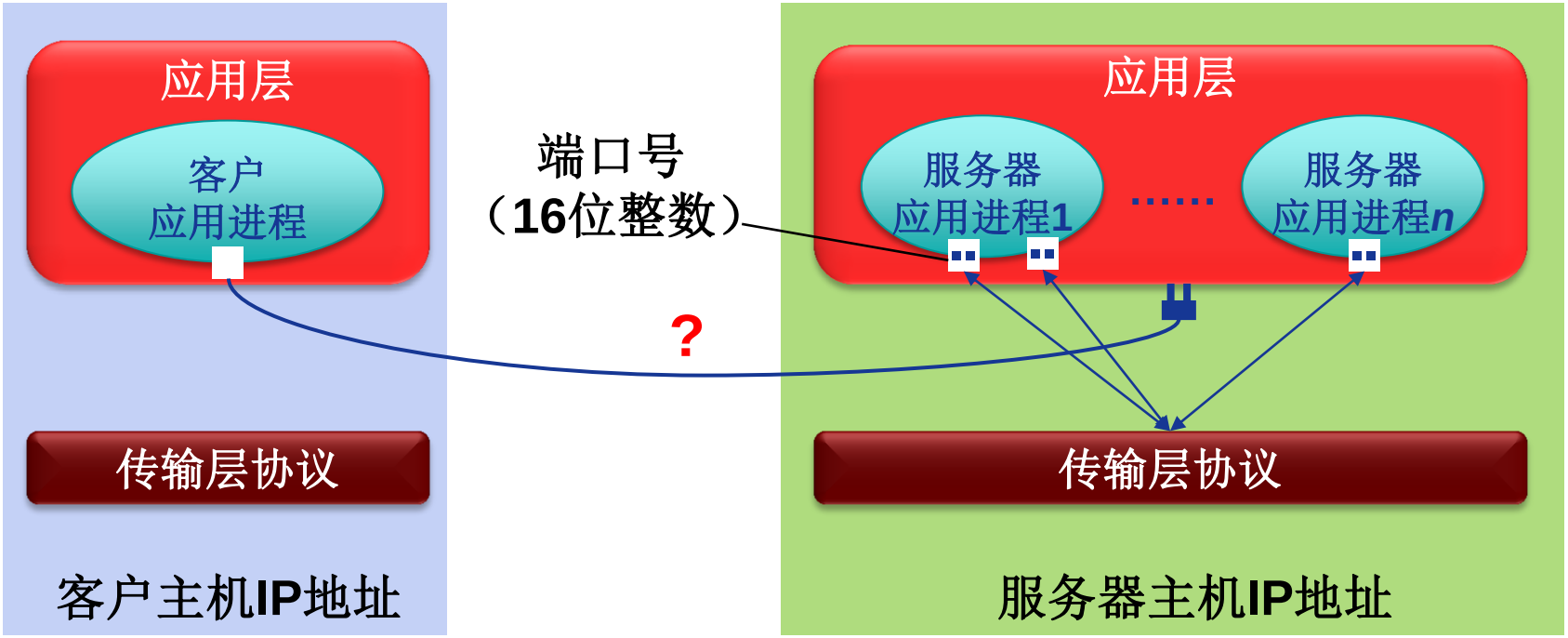
Socket编程-Socket API概述

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



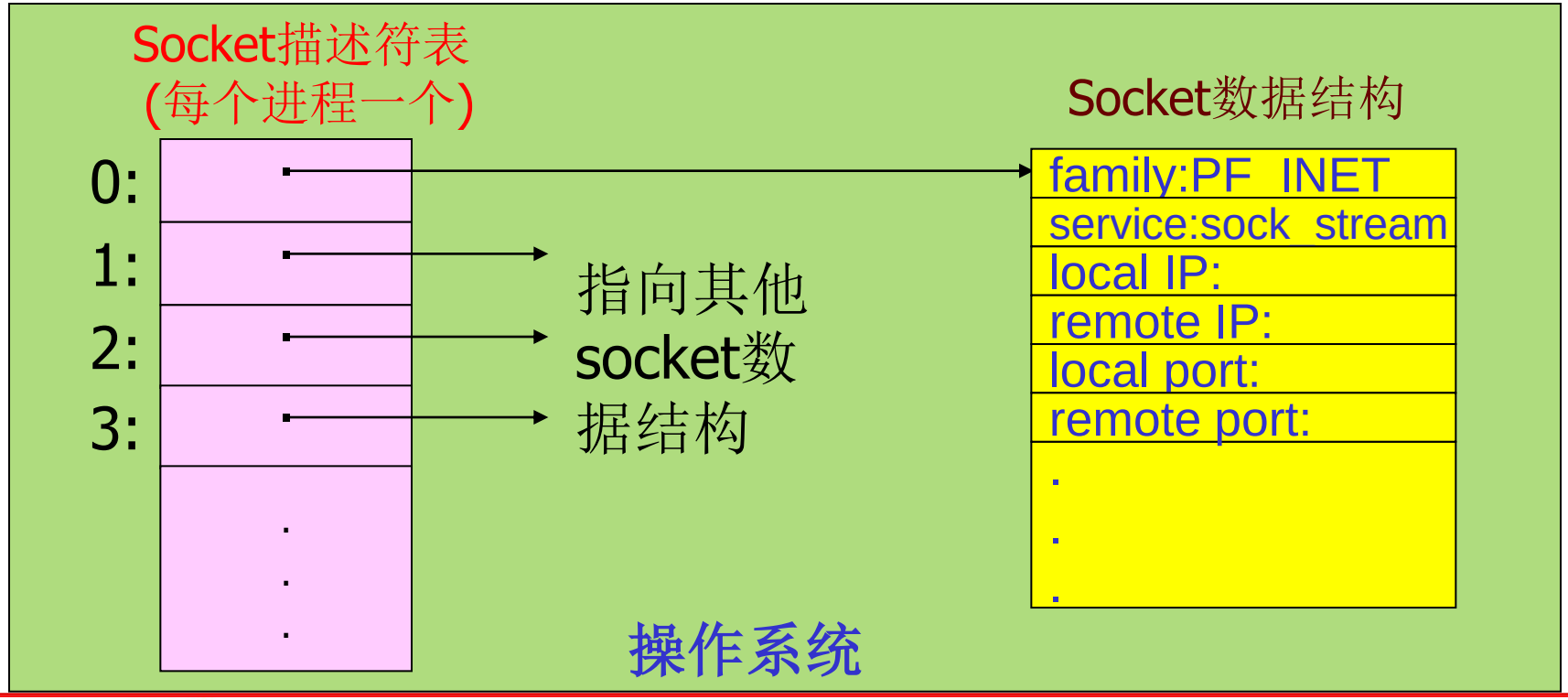
Socket API



- 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抽象

- 类似于文件的抽象
- 当应用进程创建套接字时，操作系统分配一个数据结构存储该套接字相关信息
- 返回套接字描述符



地址结构

- 已定义结构*sockaddr_in*:

```
struct sockaddr_in
{
   u_char sin_len;          /*地址长度                          */
   u_char sin_family;       /*地址族(TCP/IP：AF_INET)           */
   u_short sin_port;        /*端口号                            */
   struct in_addr sin_addr; /*IP地址                            */
   char sin_zero[8];        /*未用(置0)                         */
}
```

- 使用TCP/IP协议簇的网络应用程序声明端点地址变量时，使用结构*sockaddr_in*

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：李全龙

本讲主题

Socket编程-Socket API函数（1）

Socket API函数（WinSock）

WSAStartup
（初始化Windows Sockets API）

应用程序

WSACleanup
（释放所使用的Windows Sockets DLL）

Socket API函数

WSAStartup

int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

- 使用Socket的应用程序在使用Socket之前必须首先调用*WSAStartup*函数
- 两个参数:
  - 第一个参数指明程序请求使用的WinSock版本，其中高位字节指明副版本、低位字节指明主版本.
    - 十六进制整数，例如*0x102*表示*2.1*版
  - 第二个参数返回实际的WinSock的版本信息
    - 指向*WSADATA*结构的指针
- 例：使用2.1版本的WinSock的程序代码段

```
wVersionRequested = MAKEWORD( 2, 1 );
err = WSAStartup( wVersionRequested, &wsaData );
```

Socket API函数

WSACleanup

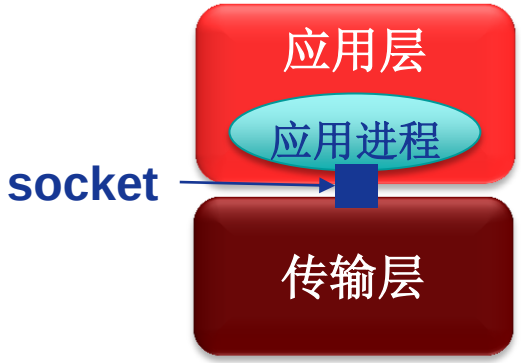
```
int WSACleanup (void);
```

- 应用程序在完成对请求的Socket库的使用，最后要调用*WSACleanup*函数
- 解除与Socket库的绑定
- 释放Socket库所占用的系统资源

Socket API函数

socket

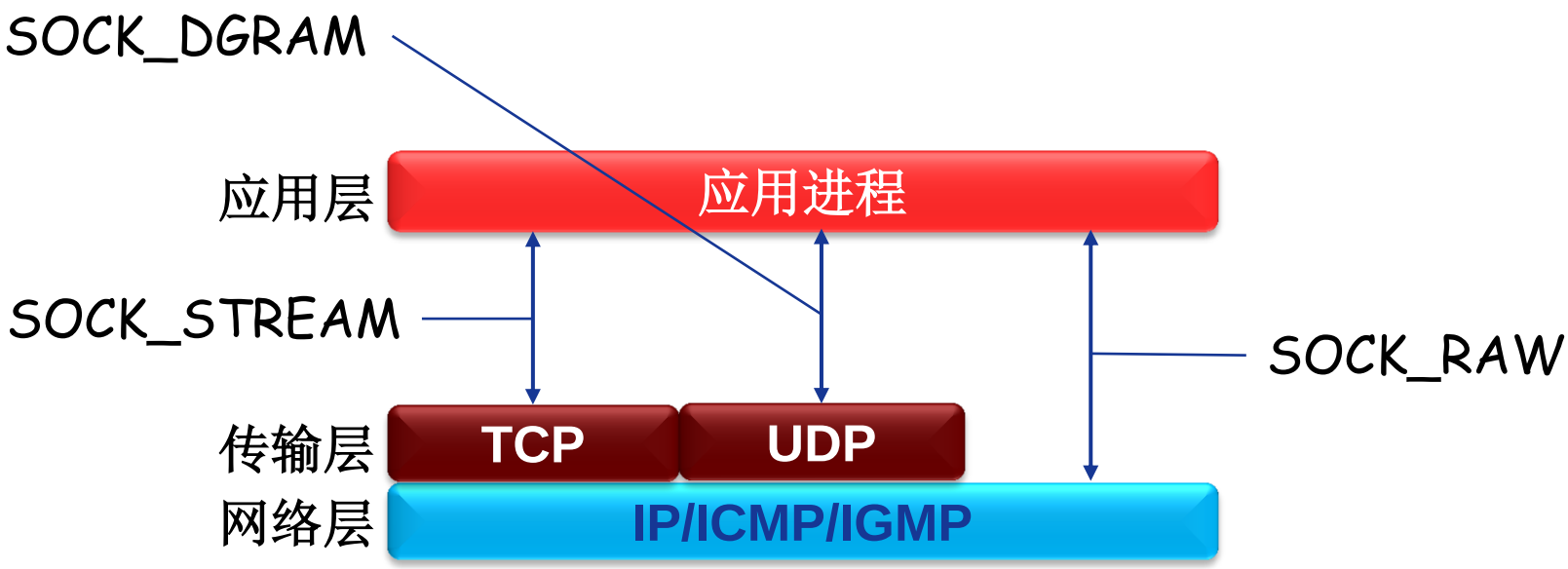
sd = socket(protofamily,type,proto);



- 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

Closesocket

```
int closesocket(SOCKET sd);
```

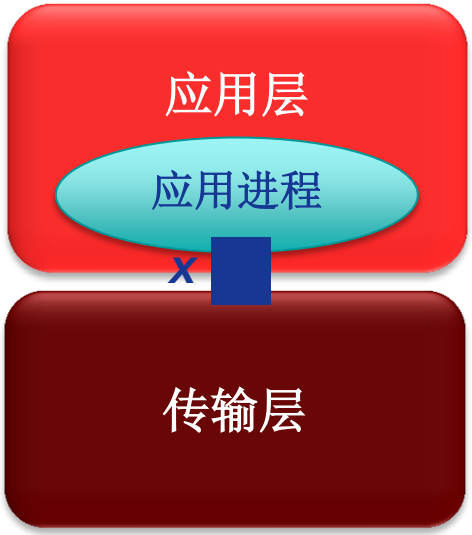
- 关闭一个描述符为sd的套接字
- 如果多个进程共享一个套接字，调用*closesocket*将套接字引用计数减1，减至0才关闭
- 一个进程中的多线程对一个套接字的使用无计数
  - 如果进程中的一个线程调用*closesocket*将一个套接字关闭，该进程中的其他线程也将不能访问该套接字
- 返回值：
  - 0：成功
  - SOCKET_ERROR：失败

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

- 考虑如下情形：

- 服务器应该绑定哪个地址？
- 问题？
- 解决方案
  - 地址通配符：INADDR_ANY

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：李全龙

本讲主题

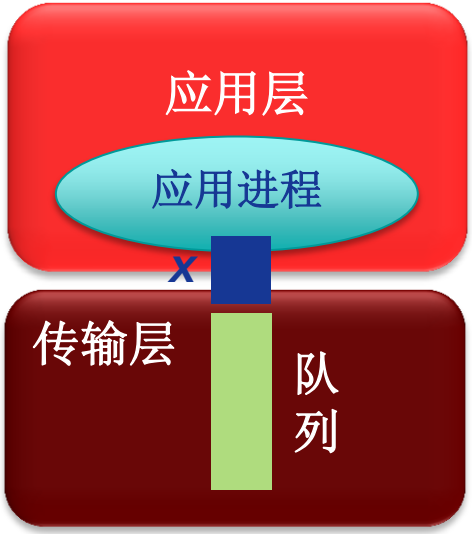
Socket编程-Socket API函数（2）

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

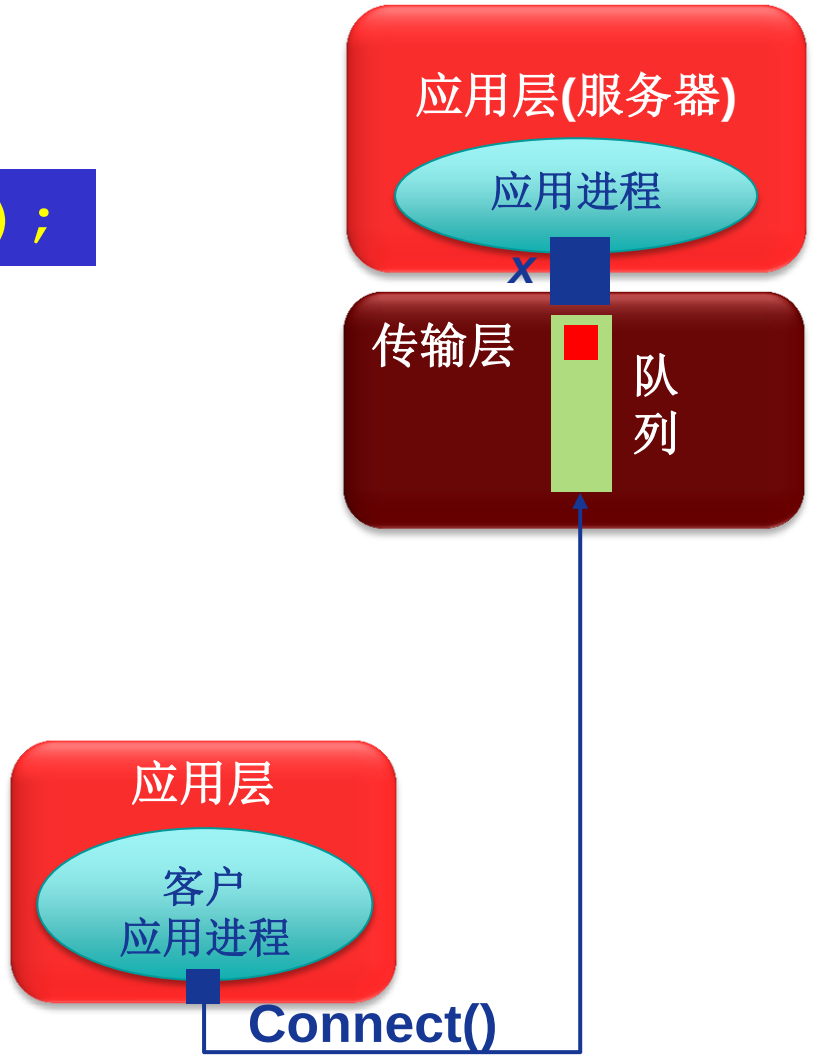


Socket API函数

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

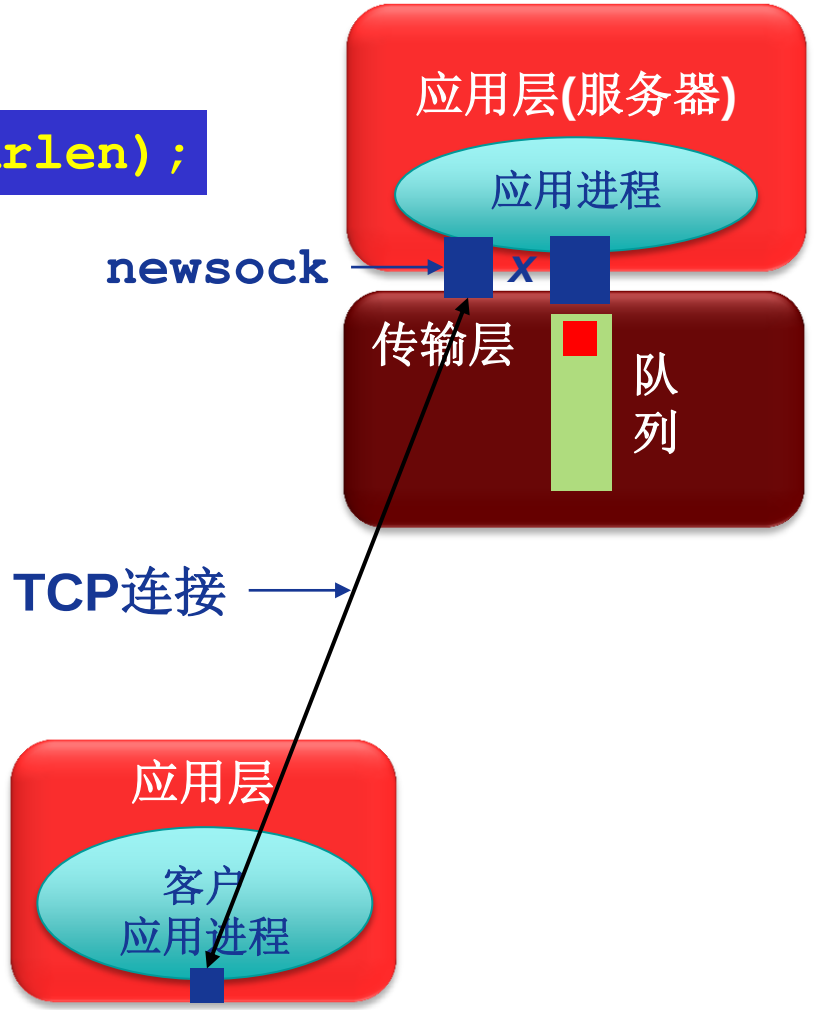


Socket API函数

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

Socket API函数

setsockopt, getsockopt

```
int setsockopt(int sd, int level, int optname, *optval, int optlen);
```

```
int getsockopt(int sd, int level, int optname, *optval, socklen_t *optlen);
```

- setsockopt()函数用来设置套接字sd的选项参数
- getsockopt()函数用于获取任意类型、任意状态套接口的选项当前值，并把结果存入optval

Socket API函数小结

- ***WSAStartup***: 初始化socket库(**仅对WinSock**)
- ***WSACleanup***: 清楚/终止socket库的使用 (**仅对WinSock**)
- ***socket***: 创建套接字
- ***connect***:“连接”远端服务器 (**仅用于客户端**)
- ***closesocket***: 释放/关闭套接字
- ***bind***: 绑定套接字的本地IP地址和端口号（**通常客户端不需要**）
- ***listen***: 置服务器端TCP套接字为监听模式，并设置队列大小 (**仅用于服务器端TCP套接字**)
- ***accept***: 接受/提取一个连接请求，创建新套接字，通过新套接 (**仅用于服务器端的TCP套接字**)
- ***recv***: 接收数据（**用于TCP套接字或连接模式的客户端UDP套接字**）

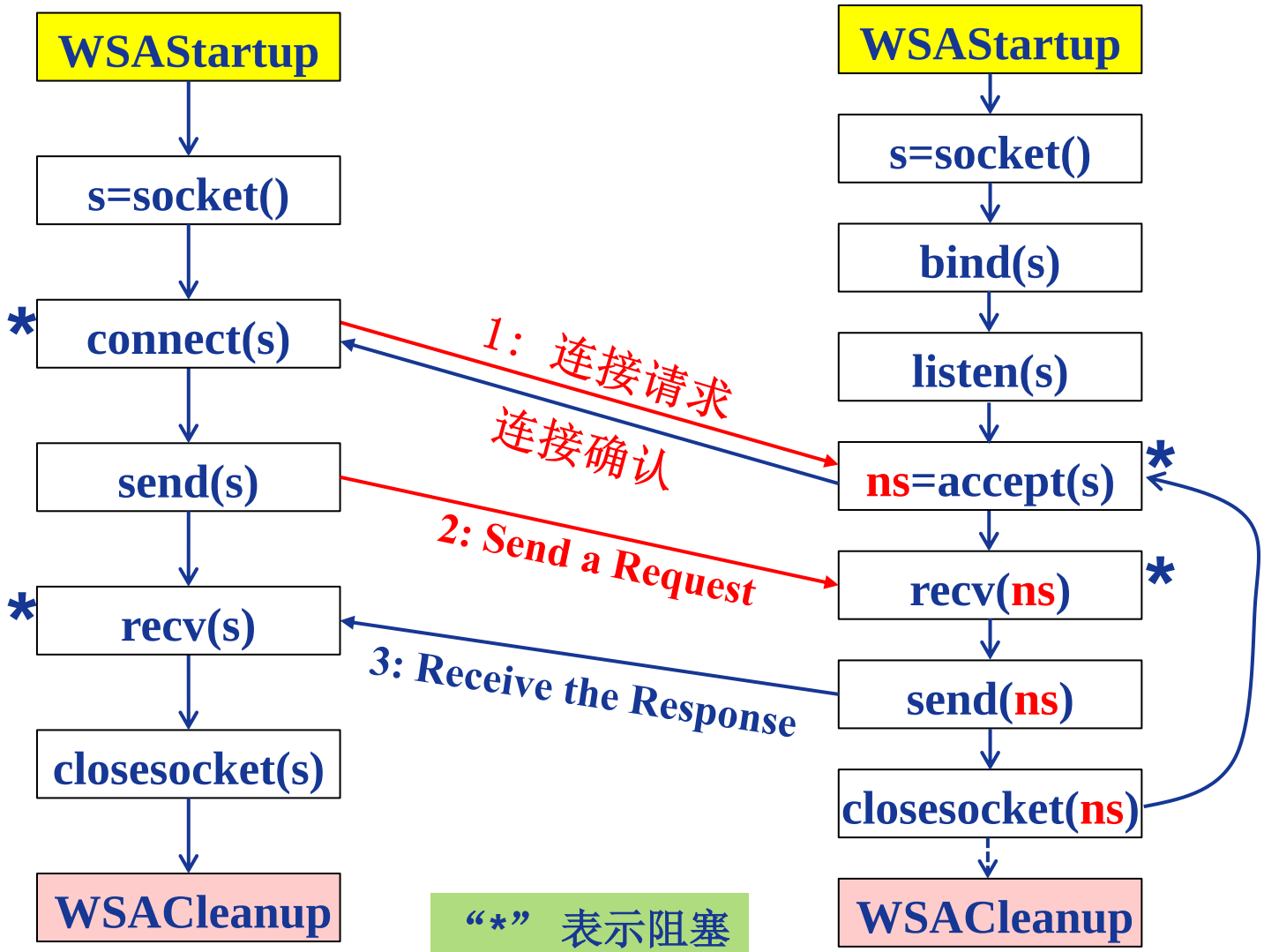
Socket API函数小结

- ***recvfrom***: 接收数据报（**用于非连接模式的UDP套接字**）
- ***send***: 发送数据（**用于TCP套接字或连接模式的客户端UDP套接字**）
- ***sendto***:发送数据报（**用于非连接模式的UDP套接字**）
- ***setsockopt***: 设置套接字选项参数
- ***getsockopt***: 获取套接字选项参数

关于网络字节顺序

- TCP/IP定义了标准的用于协议头中的二进制整数表示：网络字节顺序（network byte order）
- 某些Socket API函数的参数需要存储为网络字节顺序（如IP地址、端口号等）
- 可以实现本地字节顺序与网络字节顺序间转换的函数
  - *htons*: 本地字节顺序→网络字节顺序(16bits)
  - *ntohs*: 网络字节顺序→本地字节顺序(16bits)
  - *htonl*: 本地字节顺序→网络字节顺序(32bits)
  - *ntohl*: 网络字节顺序→本地字节顺序(32bits)

网络应用的Socket API(TCP)调用基本流程



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：李全龙

本讲主题

Socket编程-客户端软件设计

解析服务器IP地址

- 客户端可能使用域名（如:study.163.com）或IP地址（如：123.58.180.121）标识服务器
- IP协议需要使用32位二进制IP地址
- 需要将域名或IP地址转换为32位IP地址
  - 函数*inet_addr( )* 实现点分十进制IP地址到32位IP地址转换
  - 函数*gethostbyname( )* 实现域名到32位IP地址转换
    - 返回一个指向结构*hostent* 的指针

```
struct hostent {
    char FAR*       h_name;         /*official host name    */
    char FAR* FAR*  h_aliases;      /*other aliases         */
    short           h_addrtype;     /*address type          */
    short           h_lengty;       /*address length */
    char FAR* FAR*  h_addr_list;    /*list of address   */
};
#define h_addr h_addr_list[0]
```

解析服务器（熟知）端口号

- 客户端还可能使用服务名（如HTTP）标识服务器端口
- 需要将服务名转换为熟知端口号
  - 函数*getservbyname( )*
    - 返回一个指向结构*servent*的指针

```
struct servent {
    char FAR*       s_name;     /*official service name  */
    char FAR* FAR*  s_aliases;  /*other aliases          */
    short           s_port;     /*port for this service  */
    char FAR*       s_proto;    /*protocol to use        */
};
```

解析协议号

- 客户端可能使用协议名（如:TCP）指定协议
- 需要将协议名转换为协议号（如：6）
  - 函数*getprotobyname ( )* 实现协议名到协议号的转换
    - 返回一个指向结构*protoent*的指针

```
struct protoent {
    char FAR*       p_name;     /*official protocol name */
    char FAR* FAR*  p_aliases;  /*list of aliases  allowed*/
    short           p_proto;    /*official protocol number*/
    };
```

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

客户端软件的实现- connectsock()

- 设计一个*connectsock*过程封装底层代码

```
/* consock.cpp - connectsock */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <winsock.h>

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif  /* INADDR_NONE */

void    errexit(const char *, ...);
/*------------------------------------------------------------------------
 * connectsock - allocate & connect a socket using TCP or UDP
 *------------------------------------------------------------------------
 */
```

客户端软件的实现- connectsock()

```
SOCKET connectsock(const char *host, const char *service, const char
                   *transport )
{
    struct hostent  *phe;  /* pointer to host information entry      */
    struct servent  *pse;  /* pointer to service information entry   */
    struct protoent *ppe;  /* pointer to protocol information entry  */
    struct sockaddr_in sin;/* an Internet endpoint address           */
    int   s, type;         /* socket descriptor and socket type      */


    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
```

客户端软件的实现- connectsock()

```
/* Map service name to port number */
   if ( pse = getservbyname(service, transport) )
        sin.sin_port = pse->s_port;
   else if ( (sin.sin_port = htons((u_short)atoi(service))) == 0 )
        errexit("can't get \"%s\" service entry\n", service);

    /* Map host name to IP address, allowing for dotted decimal */
   if ( phe = gethostbyname(host) )
        memcpy(&sin.sin_addr, phe->h_addr, phe->h_length);
   else if ( (sin.sin_addr.s_addr = inet_addr(host))==INADDR_NONE)
        errexit("can't get \"%s\" host entry\n", host);

    /* Map protocol name to protocol number */
   if ( (ppe = getprotobyname(transport)) == 0)
        errexit("can't get \"%s\" protocol entry\n", transport);
```

客户端软件的实现- connectsock()

```
/* Use protocol to choose a socket type */
    if (strcmp(transport, "udp") == 0)
          type = SOCK_DGRAM;
    else
          type = SOCK_STREAM;
     /* Allocate a socket */
    s = socket(PF_INET, type, ppe->p_proto);
    if (s == INVALID_SOCKET)
          errexit("can't create socket: %d\n", GetLastError());
     /* Connect the socket */
    if (connect(s, (struct sockaddr *)&sin, sizeof(sin))==SOCKET_ERROR)
          errexit("can't connect to %s.%s: %d\n", host, service,
                GetLastError());
    return s;
}
```

客户端软件的实现-UDP客户端

- 设计*connectUDP*过程用于创建连接模式客户端UDP套接字

```
/* conUDP.cpp - connectUDP */
#include <winsock.h>
SOCKET connectsock(const char *, const char *, const char *);
/*------------------------------------------------------------
  * connectUDP - connect to a specified UDP service
  * on a specified host
  *------------------------------------------------------------
  */
SOCKET connectUDP(const char *host, const char *service )
{
    return connectsock(host, service, "udp");
}
```

客户端软件的实现-TCP客户端

- 设计*connectTCP*过程，用于创建客户端TCP套接字

```
/* conTCP.cpp - connectTCP */
#include <winsock.h>
SOCKET connectsock(const char *, const char *, const char *);
/*------------------------------------------------------------
  * connectTCP - connect to a specified TCP service
  * on a specified host
  *------------------------------------------------------------
  */
SOCKET connectTCP(const char *host, const char *service )
{
    return connectsock( host, service, "tcp");
}
```

客户端软件的实现-异常处理

```
/* errexit.cpp - errexit */
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
/*------------------------------------------------------------
 * errexit - print an error message and exit
 *------------------------------------------------------------
 */
/*VARARGS1*/
void errexit(const char *format, ...)
{   va_list args;
    va_start(args, format);
    vfprintf(stderr, format, args);
    va_end(args);
    WSACleanup();
    exit(1);}
```

例1：访问DAYTIME服务的客户端（TCP）

- DAYTIME服务
  - 获取日期和时间
  - 双协议服务（TCP、 UDP），端口号13
  - TCP版利用TCP连接请求触发服务
  - UDP版需要客户端发送一个请求

例1：访问DAYTIME服务的客户端（TCP）

```
/* TCPdtc.cpp - main, TCPdaytime */
#include <stdlib.h>
#include <stdio.h>
#include <winsock.h>

void      TCPdaytime(const char *, const char *);
void      errexit(const char *, ...);
SOCKET connectTCP(const char *, const char *);

#define LINELEN 128
#define WSVERS MAKEWORD(2, 0)
/*------------------------------------------------------------
 * main - TCP client for DAYTIME service
 *------------------------------------------------------------
 */
```

例1：访问DAYTIME服务的客户端（TCP）

```
int main(int argc, char *argv[])
{
    char *host = "localhost";      /* host to use if none supplied  */
    char *service = "daytime";     /* default service port          */
    WSADATA wsadata;
    switch (argc) {
    case 1:
          host = "localhost";
          break;
    case 3:
          service = argv[2];
          /* FALL THROUGH */
    case 2:
          host = argv[1];
          break;
```

例1：访问DAYTIME服务的客户端（TCP）

```
default:
        fprintf(stderr, "usage: TCPdaytime [host [port]]\n");
        exit(1);
    }
if (WSAStartup(WSVERS, &wsadata) != 0)
        errexit("WSAStartup failed\n");
    TCPdaytime(host, service);
    WSACleanup();
    return 0;     /* exit */
}

/*------------------------------------------------------
 * TCPdaytime - invoke Daytime on specified host and print results
 *------------------------------------------------------
 */
```

例1：访问DAYTIME服务的客户端（TCP）

```
void TCPdaytime(const char *host, const char *service)
{
    char buf[LINELEN+1];        /* buffer for one line of text   */
    SOCKET s;                   /* socket descriptor             */
    int cc;                     /* recv character count          */
    s = connectTCP(host, service);

    cc = recv(s, buf, LINELEN, 0);
    while( cc != SOCKET_ERROR && cc > 0)
    {
          buf[cc] = '\0';       /* ensure null-termination       */
          (void) fputs(buf, stdout);
         cc = recv(s, buf, LINELEN, 0);
    }
    closesocket(s);
}
```

例2：访问DAYTIME服务的客户端（UDP）

```
/* UDPdtc.cpp - main, UDPdaytime */
#include <stdlib.h>
#include <stdio.h>
#include <winsock.h>

void      UDPdaytime(const char *, const char *);
void      errexit(const char *, ...);
SOCKET connectUDP(const char *, const char *);

#define LINELEN 128
#define WSVERS MAKEWORD(2, 0)
#define MSG  "what daytime is it?\n"

/*------------------------------------------------------------------------
 * main - UDP client for DAYTIME service
 *------------------------------------------------------------------------
 */
```

例2：访问DAYTIME服务的客户端（UDP）

```
int main(int argc, char *argv[])
{
    char *host = "localhost";      /* host to use if none supplied  */
    char *service = "daytime";     /* default service port          */
    WSADATA wsadata;
    switch (argc) {
    case 1:
          host = "localhost";
          break;
    case 3:
          service = argv[2];
          /* FALL THROUGH */
    case 2:
          host = argv[1];
          break;
```

例2：访问DAYTIME服务的客户端（UDP）

```
default:
        fprintf(stderr, "usage: UDPdaytime [host [port]]\n");
        exit(1);
    }
if (WSAStartup(WSVERS, &wsadata) != 0)
        errexit("WSAStartup failed\n");
    UDPdaytime(host, service);
    WSACleanup();
    return 0;     /* exit */
}

/*------------------------------------------------
 * UDPdaytime - invoke Daytime on specified host and print results
 *------------------------------------------------
 */
```

例2：访问DAYTIME服务的客户端（UDP）

```
void UDPdaytime(const char *host, const char *service)
{
    char   buf[LINELEN+1];    /* buffer for one line of text */
    SOCKET  s;                /* socket descriptor           */
    int  n;                   /* recv character count        */

    s = connectUDP(host, service);
    (void) send(s, MSG, strlen(MSG), 0);

    /* Read the daytime */
    n = recv(s, buf, LINELEN, 0);
    if (n == SOCKET_ERROR)
        errexit("recv failed: recv() error %d\n", GetLastError());
    else
    {
        buf[cc] = '\0';       /* ensure null-termination     */
        (void) fputs(buf, stdout);
    }
    closesocket(s);
    return 0;                 /* exit                         */
}
```

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

计算机网络之网尽其用

主讲人：李全龙

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

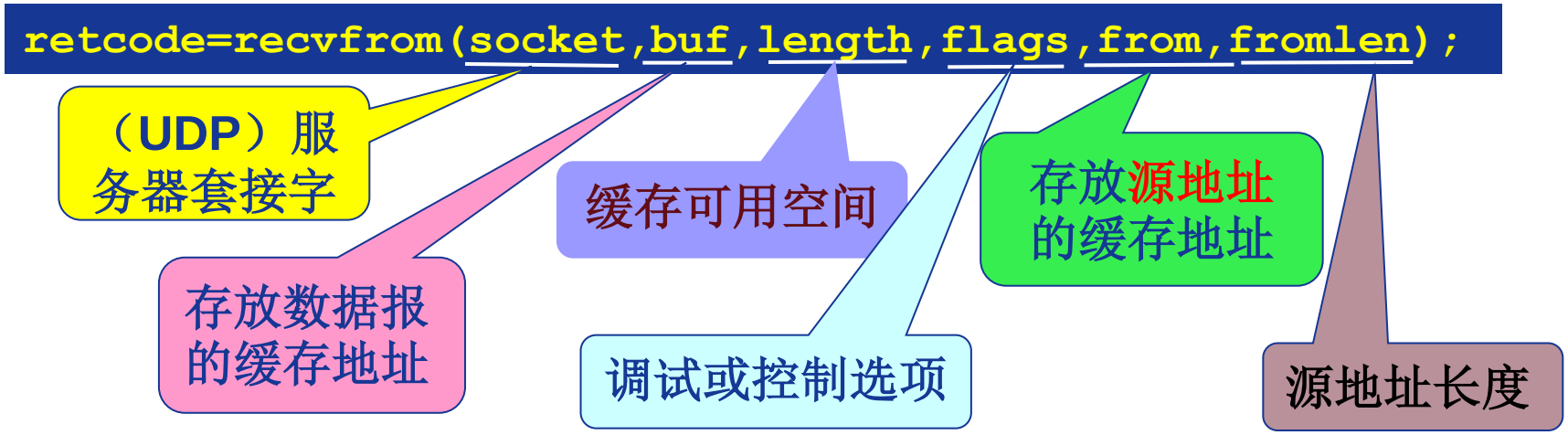
数据发送

- 服务器端不能使用*connect()*函数
- 无连接服务器使用*sendto()*函数发送数据报

```
retcode=sendto(socket,data,length,flags,destaddr,addrlen);
```

获取客户端点地址

- 调用*recvfrom()*函数接收数据时，自动提取



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.

并发无连接服务器基本流程

主线程1: 创建套接字，并绑定熟知端口号；

主线程2: 反复调用***recvfrom()***函数，接收下一个客户请求，并创建新线程处理该客户响应；

子线程1: 接收一个特定请求；

子线程2: 依据应用层协议构造响应报文，并调用***sendto()***发送；

子线程3: 退出(一个子线程处理一个请求后即终止)。

并发面向连接服务器基本流程

主线程1: 创建（主）套接字，并绑定熟知端口号；

主线程2: 设置（主）套接字为被动监听模式，准备用于服务器；

主线程3: 反复调用***accept()***函数接收下一个连接请求（通过主套接字），并创建一个新的子线程处理该客户响应；

子线程1: 接收一个客户的服务请求（通过新创建的套接字）；

子线程2: 遵循应用层协议与特定客户进行交互；

子线程3: 关闭/释放连接并退出（线程终止）.

服务器的实现

- 设计一个底层过程隐藏底层代码：
  - *passivesock()*
- 两个高层过程分别用于创建服务器端UDP套接字和TCP套接字（调用passivesock()函数）：
  - *passiveUDP()*
  - *passiveTCP()*

服务器的实现-*passivesock()*

```
/* passsock.cpp - passivesock */
#include <stdlib.h>
#include <string.h>
#include <winsock.h>
void	errexit(const char *, ...);

/*------------------------------------------------------------------------
 * passivesock - allocate & bind a server socket using TCP or UDP
 *------------------------------------------------------------------------
 */
SOCKET passivesock(const char *service, const char *transport, int qlen)
```

服务器的实现-passivesock()

```
{
    struct servent *pse;    /* pointer to service information entry      */
    struct protoent *ppe;  /* pointer to protocol information entry     */
    struct sockaddr_in sin;/* an Internet endpoint address              */
    SOCKET      s;          /* socket descriptor                         */
    int     type;     /* socket type (SOCK_STREAM, SOCK_DGRAM)*/

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;

   /* Map service name to port number */
    if ( pse = getservbyname(service, transport) )
            sin.sin_port = (u_short)pse->s_port;
    else if ( (sin.sin_port = htons((u_short)atoi(service))) == 0 )
            errexit("can't get \"%s\" service entry\n", service);
```

服务器的实现-passivesock()

```
 /* Map protocol name to protocol number */
 if ( (ppe = getprotobyname(transport)) == 0)
        errexit("can't get \"%s\" protocol entry\n", transport);
/* Use protocol to choose a socket type */
 if (strcmp(transport, "udp") == 0)
        type = SOCK_DGRAM;
 else
        type = SOCK_STREAM;
/* Allocate a socket */
 s = socket(PF_INET, type, ppe->p_proto);
 if (s == INVALID_SOCKET)
        errexit("can't create socket: %d\n", GetLastError());
/* Bind the socket */
 if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) == SOCKET_ERROR)
        errexit("can't bind to %s port: %d\n", service,
                GetLastError());
 if (type == SOCK_STREAM && listen(s, qlen) == SOCKET_ERROR)
        errexit("can't listen on %s port: %d\n", service,
                GetLastError());
 return s;}
```

服务器的实现-*passiveUDP()*

```
/* passUDP.cpp - passiveUDP */

#include <winsock.h>

SOCKET passivesock(const char *, const char *, int);

/*------------------------------------------------------------------------
 * passiveUDP - create a passive socket for use in a UDP server
 *------------------------------------------------------------------------
 */
SOCKET passiveUDP(const char *service)
{
    return passivesock(service, "udp", 0);
}
```

服务器的实现-*passiveTCP()*

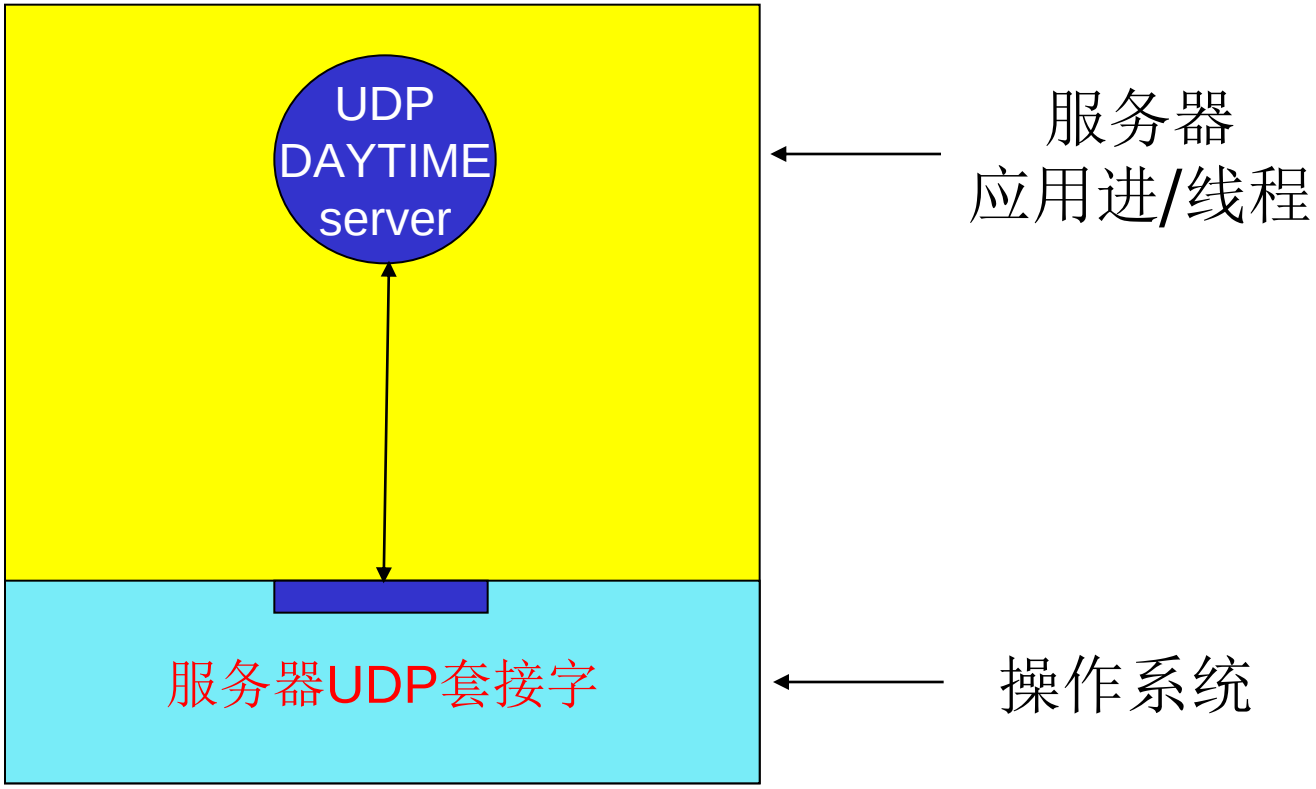
```
/* passTCP.cpp - passiveTCP */

#include <winsock.h>

SOCKET passivesock(const char *, const char *, int);

/*------------------------------------------------------------------------
 * passiveTCP - create a passive socket for use in a TCP server
 *------------------------------------------------------------------------
 */
SOCKET passiveTCP(const char *service, int qlen)
{
    return passivesock(service, "tcp", qlen);
}
```

例1：无连接循环DAYTIME服务器



例1：无连接循环DAYTIME服务器

```
/* UDPdtd.cpp - main, UDPdaytimed */
#include <stdlib.h>
#include <winsock.h>
#include <time.h>

void	errexit(const char *, ...);
SOCKET passiveUDP(const char *);

#define WSVERS		MAKEWORD(2, 0)

/*------------------------------------------------------------------------
 * main - Iterative UDP server for DAYTIME service
 *------------------------------------------------------------------------
 */
void main(int argc, char *argv[])
```

例1：无连接循环DAYTIME服务器

```
{
    struct sockaddr_in fsin;          /* the from address of a client     */
    char  *service = "daytime";       /* service name or port number      */
    SOCKET sock;                      /* socket                           */
    int     alen;                     /* from-address length              */
    char  *  pts;                     /* pointer to time string           */
    time_t  now;                      /* current time                     */
    WSADATA wsadata;

    switch (argc)
    {
      case             1:
            break;
      case             2:
            service = argv[1];
            break;
      default:
            errexit("usage: UDPdaytimed [port]\n");
    }
```

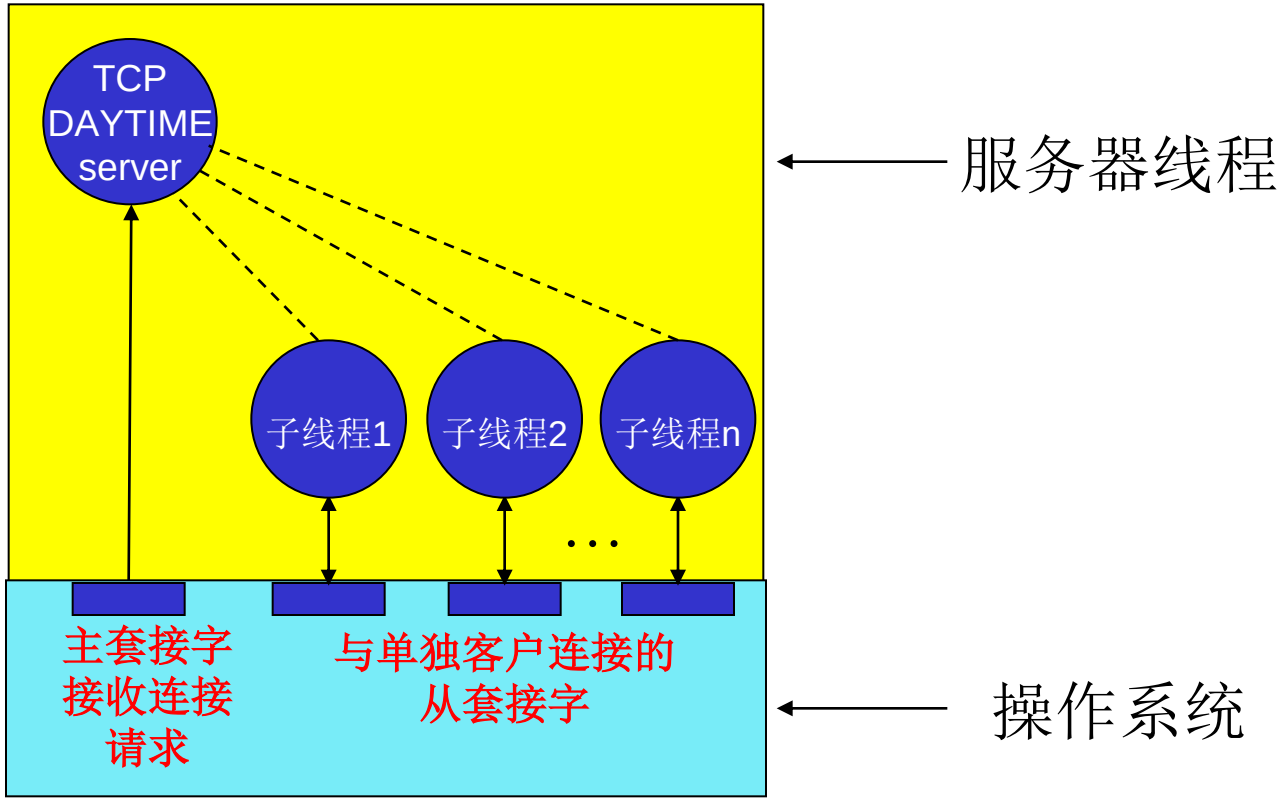
例1：无连接循环DAYTIME服务器

```
if (WSAStartup(WSVERS, &wsadata) != 0)
        errexit("WSAStartup failed\n");

    sock = passiveUDP(service);

    while (1)
     {
        alen = sizeof(struct sockaddr);
        if (recvfrom(sock, buf, sizeof(buf), 0,
            (struct sockaddr *)&fsin, &alen) == SOCKET_ERROR)
                errexit("recvfrom: error %d\n", GetLastError());
        (void) time(&now);
        pts = ctime(&now);
        (void) sendto(sock, pts, strlen(pts), 0,
                (struct sockaddr *)&fsin, sizeof(fsin));
    }
    return 1;       /* not reached */
}
```

例2：面向连接并发DAYTIME服务器



例2：面向连接并发DAYTIME服务器

```
/* TCPdtd.cpp - main, TCPdaytimed */
#include <stdlib.h>
#include <winsock.h>
#include <process.h>
#include <time.h>

void	errexit(const char *, ...);
void	TCPdaytimed(SOCKET);
SOCKET passiveTCP(const char *, int);

#define QLEN  5
#define WSVERS  MAKEWORD(2, 0)

/*------------------------------------------------------------------------
 * main - Concurrent TCP server for DAYTIME service
 *------------------------------------------------------------------------
 */
void main(int argc, char *argv[])
```

例2：面向连接并发DAYTIME服务器

```
{
    struct sockaddr_in fsin;        /* the from address of a client     */
    char *service = "daytime";      /* service name or port number*/
    SOCKET msock, ssock;            /* master & slave sockets           */
    int    alen;                    /* from-address length              */
    WSADATA wsadata;

    switch (argc) {
    case 1:
            break;
    case 2:
            service = argv[1];
            break;
    default:
            errexit("usage: TCPdaytimed [port]\n");
    }
```

例2：面向连接并发DAYTIME服务器

```
if (WSAStartup(WSVERS, &wsadata) != 0)
        errexit("WSAStartup failed\n");

    msock = passiveTCP(service, QLEN);

    while (1) {
        alen = sizeof(struct sockaddr);
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if (ssock == INVALID_SOCKET)
                errexit("accept failed: error number %d\n",
                        GetLastError());
        if (_beginthread((void (*)(void *)) TCPdaytimed, 0,
            (void *)ssock) < 0) {
                errexit("_beginthread: %s\n", strerror(errno));
        }
    }
    return 1;       /* not reached */
}
```

例2：面向连接并发DAYTIME服务器

```
/*------------------------------------------------------------------
 * TCPdaytimed - do TCP DAYTIME protocol
 *------------------------------------------------------------------
 */
void TCPdaytimed(SOCKET fd)
{
    char *          pts;                /* pointer to time string   */
    time_t           now;               /* current time             */

    (void) time(&now);
    pts = ctime(&now);
    (void) send(fd, pts, strlen(pts), 0);
    (void) closesocket(fd);
}
```

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

立足航天，服务国防，面向国民经济主战场

谢 谢！