网络编程技术

UDP网络编程

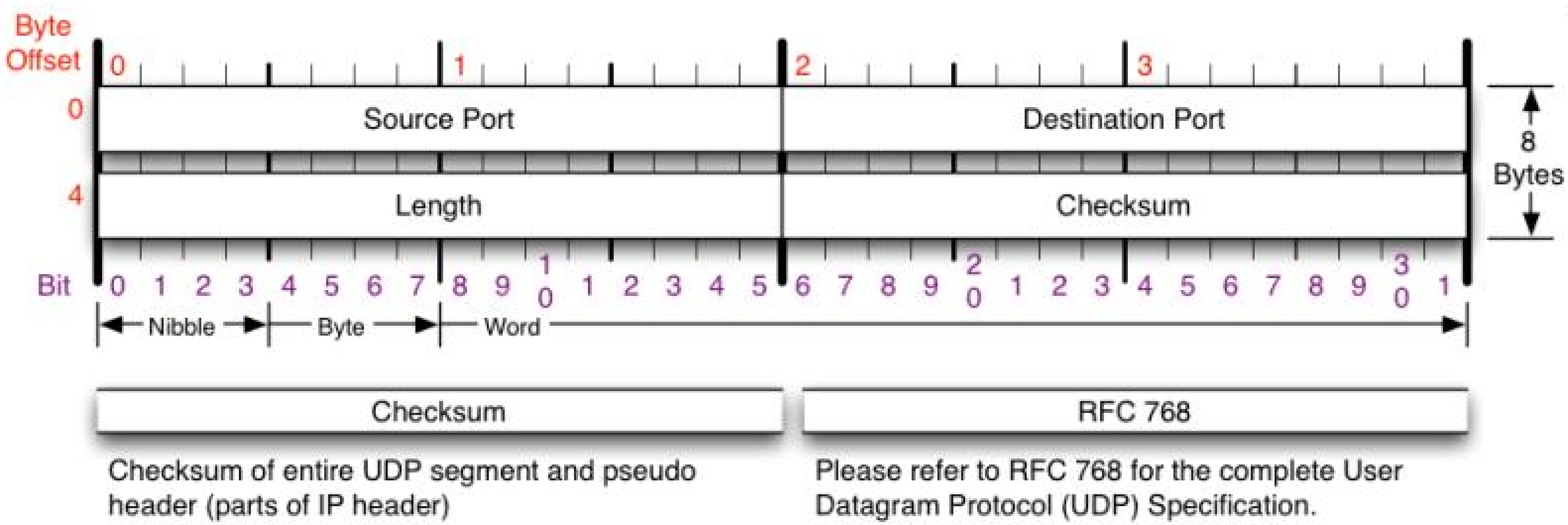
清华大学网研院
张千里

本节课内容

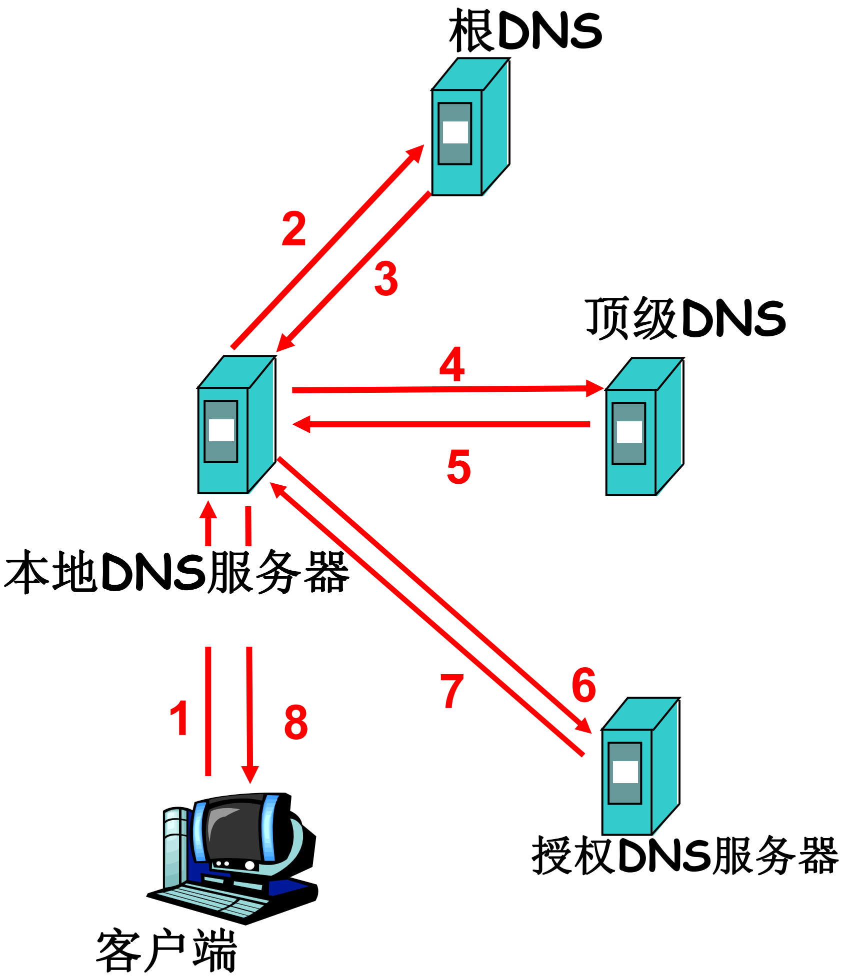
- 主要内容
  - **UDP套接字编程**
- 参考资料
  - **UNIX Network programming：第8、22章**
    - **Elementary UDP Sockets**
    - **Advanced UDP Sockets**

UDP 用途简介（一）
UDP特点

- UDP协议中包头很小，只有8字节
- UDP是无连接的，仅提供了多路复用、可选的差错控制功能
- UDP没有拥塞控制、流量控制等机制
- UDP支持组播和广播

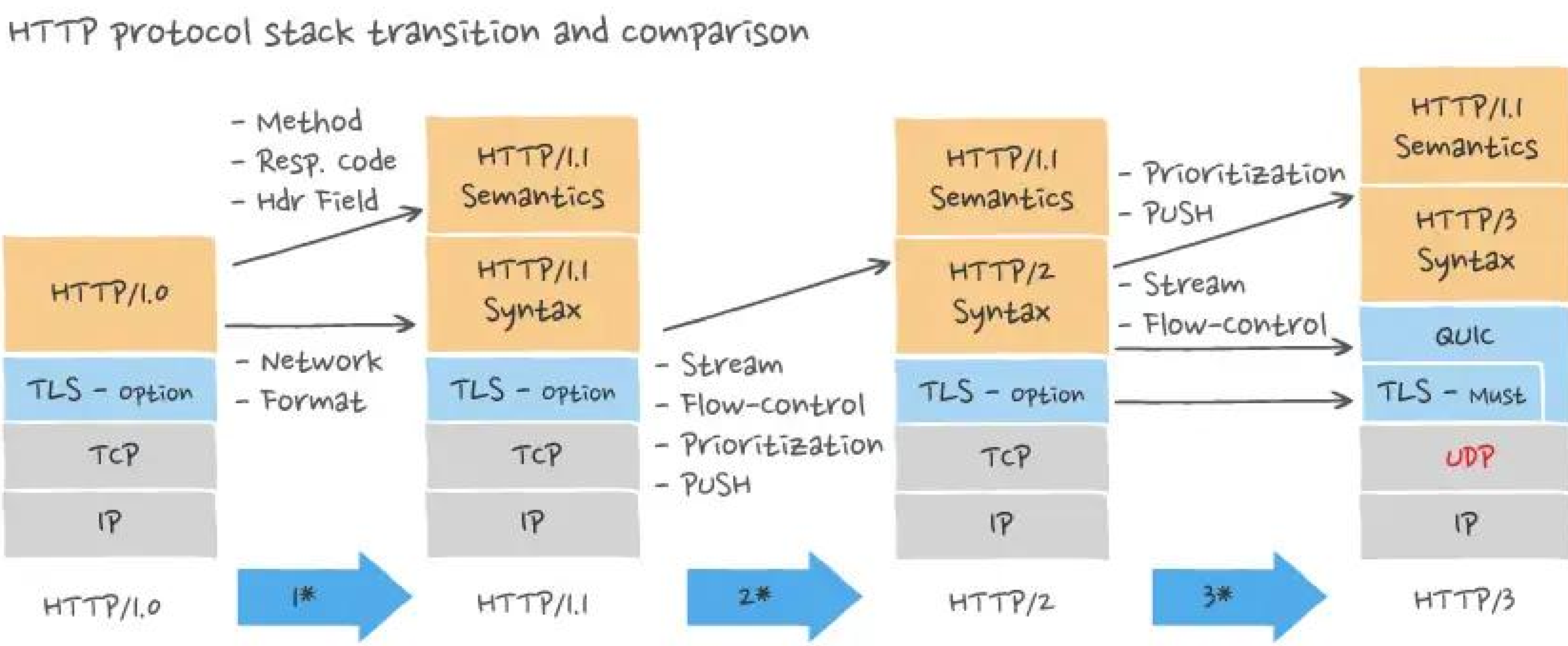


UDP 用途简介（二）
简单的应答协议

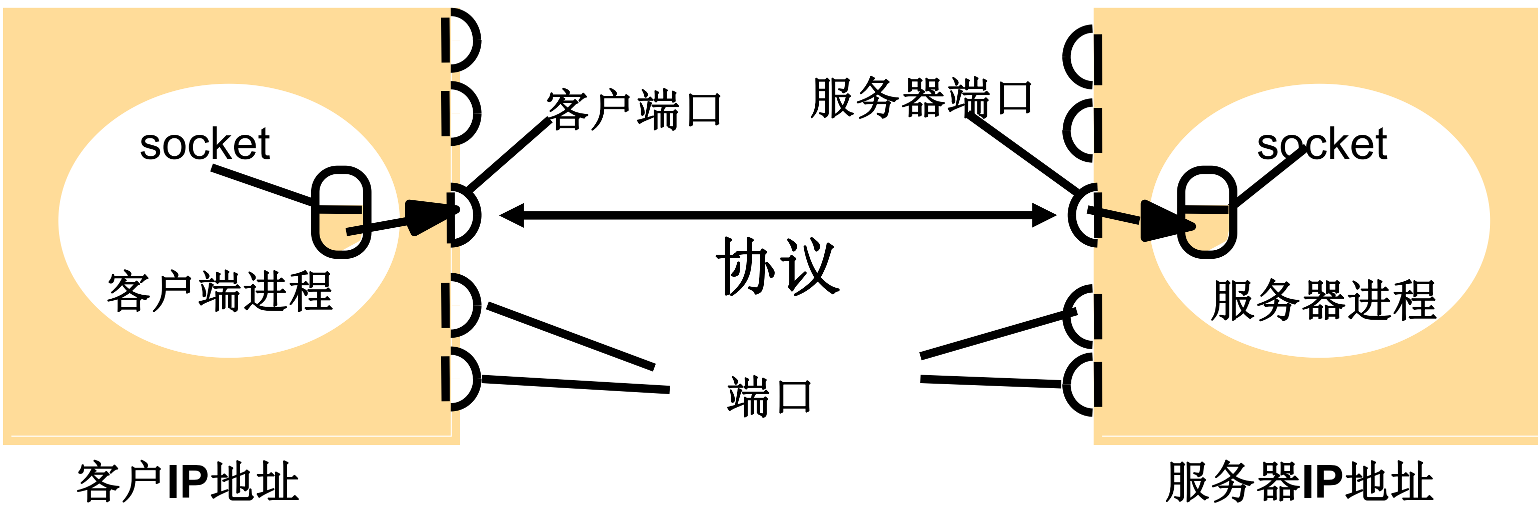


identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

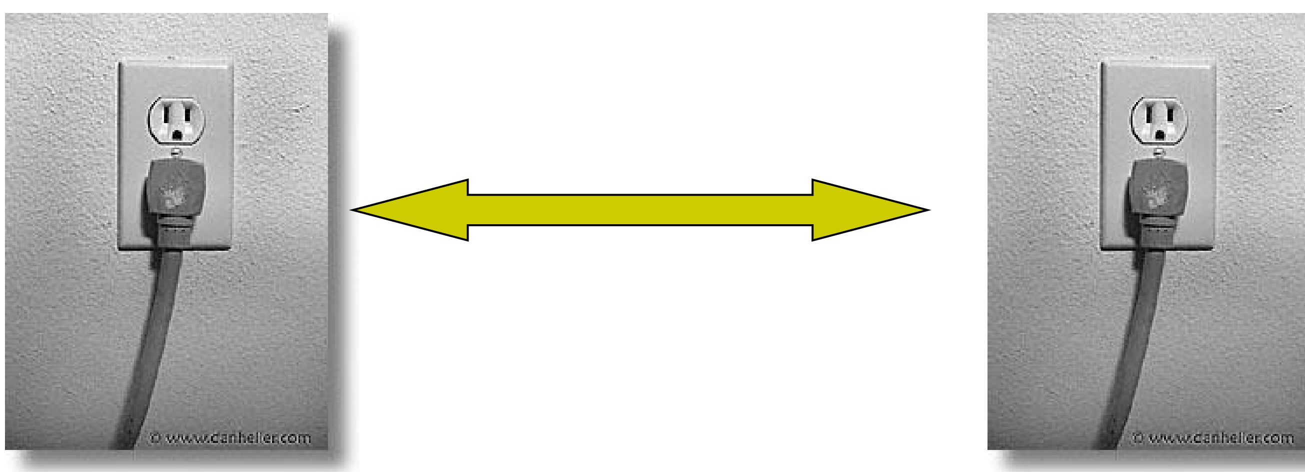
UDP 用途简介（三）最新的发展QUIC



UDP套接字编程（一）
套接字Socket



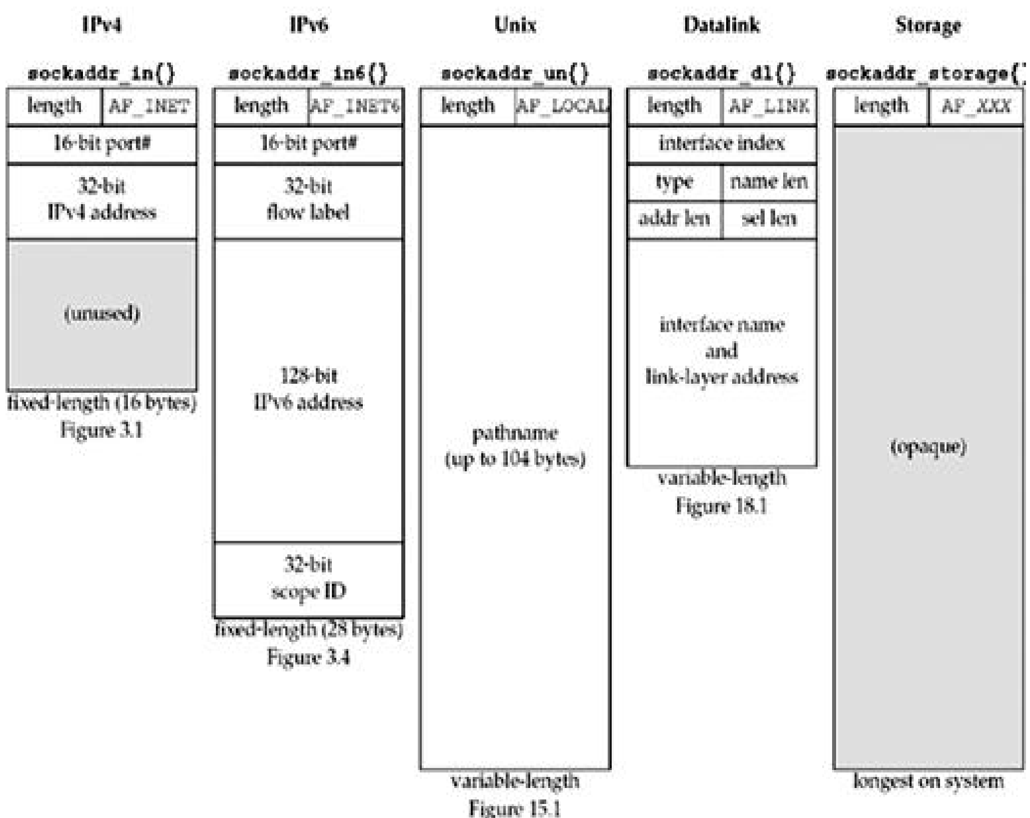
客户IP地址
协议
客户端口



服务器IP地址
协议
服务器端口

UDP套接字编程（二）
套接字的地址：sockaddr_in

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```



续： 字节序（4A3B2C1D）

- Little-endian (“little end first”)

	100	101	102	103	
...	1D	2C	3B	4A	...

  - 最低有效位（least significant byte byte (lsb, 1D)）在先
  - Intel x86, DEC VAX
- Big-endian (“big end first”)

	100	101	102	103	
...	4A	3B	2C	1D	...

  - 最高有效位（most significant byte (msb, 4A)）在先
  - Motorola/SPARC、 网络字节序
- 字节序转换函数
  - htonl、htons、ntohl、ntohs：h代表主机、n代表网络、l代表4字节整型、s代表2字节短整型

UDP套接字编程（三）
创建套接字：socket()

- 基本语法

#include <sys/types.h>

#include <sys/socket.h>

sd = socket(协议族, 服务类型, 协议);

例：int sock = socket(AF_INET,SOCK_DGRAM, 0);

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP|SCTP	TCP|SCTP	Yes		
SOCK_DGRAM	UDP	UDP	Yes		
SOCK_SEQPACKET	SCTP	SCTP	Yes		
SOCK_RAW	IPv4	IPv6		Yes	Yes

UDP套接字编程（四）
套接字和地址绑定：bind()

- 功能：将套接字绑定到协议、地址、端口所构成的套接字地址
  - 通常用于服务器，但也可以用在客户端
  - IPv4使用sockaddr_in结构，包含IP地址和端口号
- 参数说明：
  - mysock:套接字描述符，指明创建连接的套接字
  - myaddr:本地地址，IP地址和端口号

```
int mysock;
struct sockaddr_in myaddr;
mysock = socket(AF_INET,SOCK_DGRAM,0);
memset(&myaddr, 0, sizeof(myaddr));
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons( 53 );
myaddr.sin_addr = htonl( INADDR_ANY );
bind(mysock, (sockaddr *) myaddr,sizeof(myaddr));
```

UDP套接字编程（五）报文发送sendto

```
#include <sys/types.h>
#include <sys/socket.h>
int  sendto(int sd, const void *msg, size_t len, int flags,
            const struct sockaddr *to, socklen_t tolen);
```

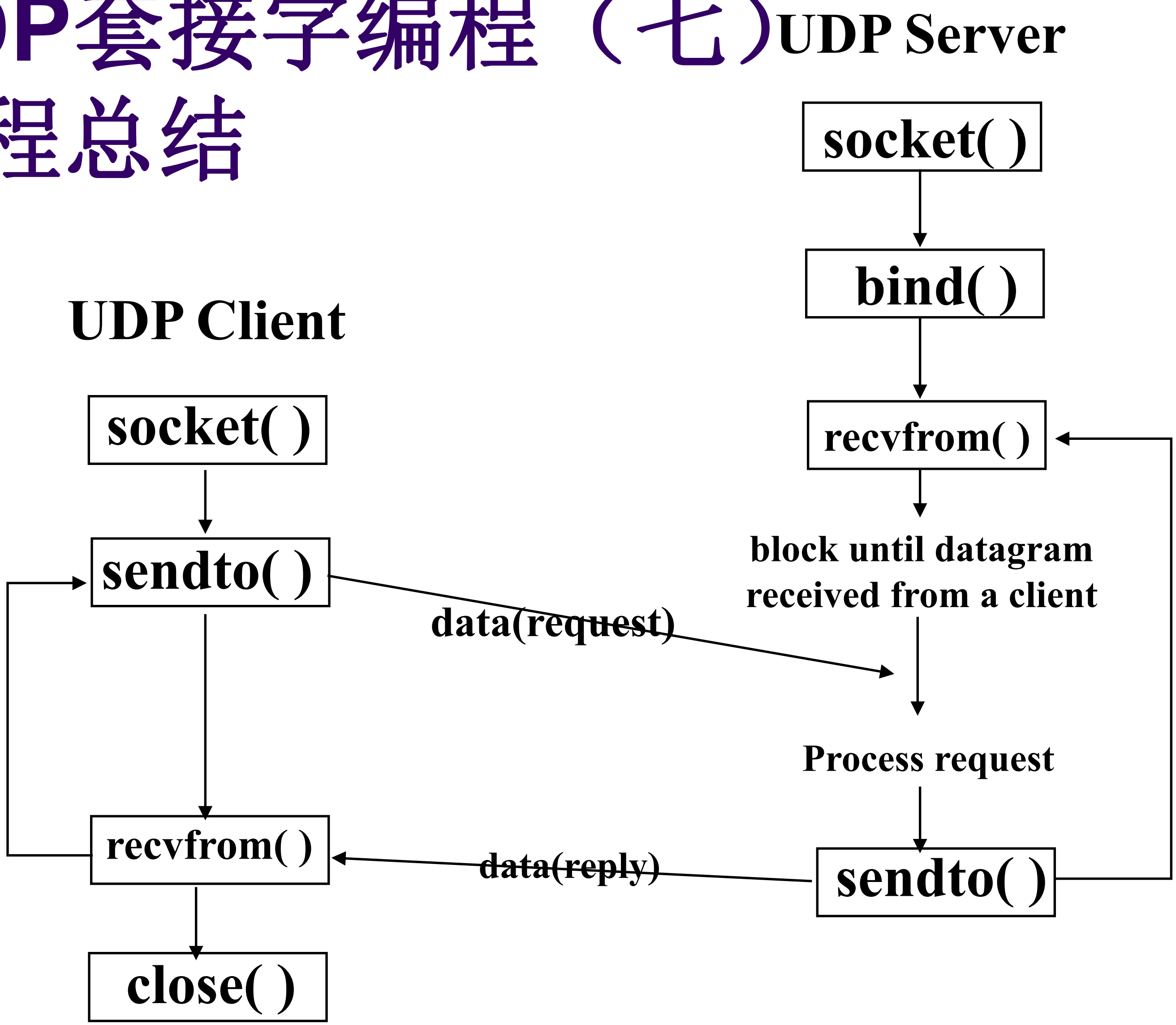
- **sd：套结字描述符**
- **msg：要发送的内容**
- **len：要发送的长度**
- **flags：功能选项，一般设置为0**
- **to：对方地址**
- **tolen：地址的长度（可以用sizeof(struct sockaddr_in)）**

UDP套接字编程（六）报文接收recv等

```
int recv(int sd, void *buf, size_t len, int flags);
int recvfrom(int  sd, void *buf, size_t len, int flags, struct
sockaddr *from, socklen_t *fromlen);
```

- sd：套结字描述符
- buf：接收缓冲区
- len：最大接收长度
- flags：功能选项，一般设置为0
- from：如果不是NULL的话，将在指针所指的结构中填写对方地址
- fromlen：如果不是NULL的话，将在指针所指的结构中填写对方地址的长度
- 返回接收到的字节数，-1表示错误

UDP套接字编程（七）流程总结



Socket functions for UDP client-server

udpcliserv/udpserv01.c

```c
#include    "unp.h"
int main(int argc, char **argv)
{
    int     sockfd;
    struct sockaddr_in servaddr, cliaddr;
    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);
    Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));
    dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
}
```

lib/wrapsock.c

```
int
Socket(int family, int type, int protocol)
{
	int		n;
	if ( (n = socket(family, type, protocol)) < 0)
		err_sys("socket error");
	return(n);
}
```

lib/dg_echo.c

```
#include    "unp.h"
 void  dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
 {
    int     n;
    socklen_t len;
    char    mesg[MAXLINE];
    for ( ; ; ) {
        len = clilen;
        n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);
        Sendto(sockfd, mesg, n, 0, pcliaddr, len);
    }
}
```

udpcliserv/udpcli01.c

```
#include    "unp.h"
int main(int argc, char **argv)
{
   int    sockfd;
   struct sockaddr_in servaddr;
   if(argc != 2)
     err_quit("usage: udpcli <IPaddress>");
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(SERV_PORT);
   Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
   sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
   dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));
   exit(0);
}
```

lib/dg_cli.c

```
#include    "unp.h"
void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
    int     n;
    char    sendline[MAXLINE], recvline[MAXLINE + 1];
    while (Fgets(sendline, MAXLINE, fp) != NULL) {
        Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
        n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
        recvline[n] = 0;        /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

主观题 10分

设置

上面的程序有什么问题?

正常使用主观题需2.0以上版本雨课堂

作答

网络编程需要考虑的问题

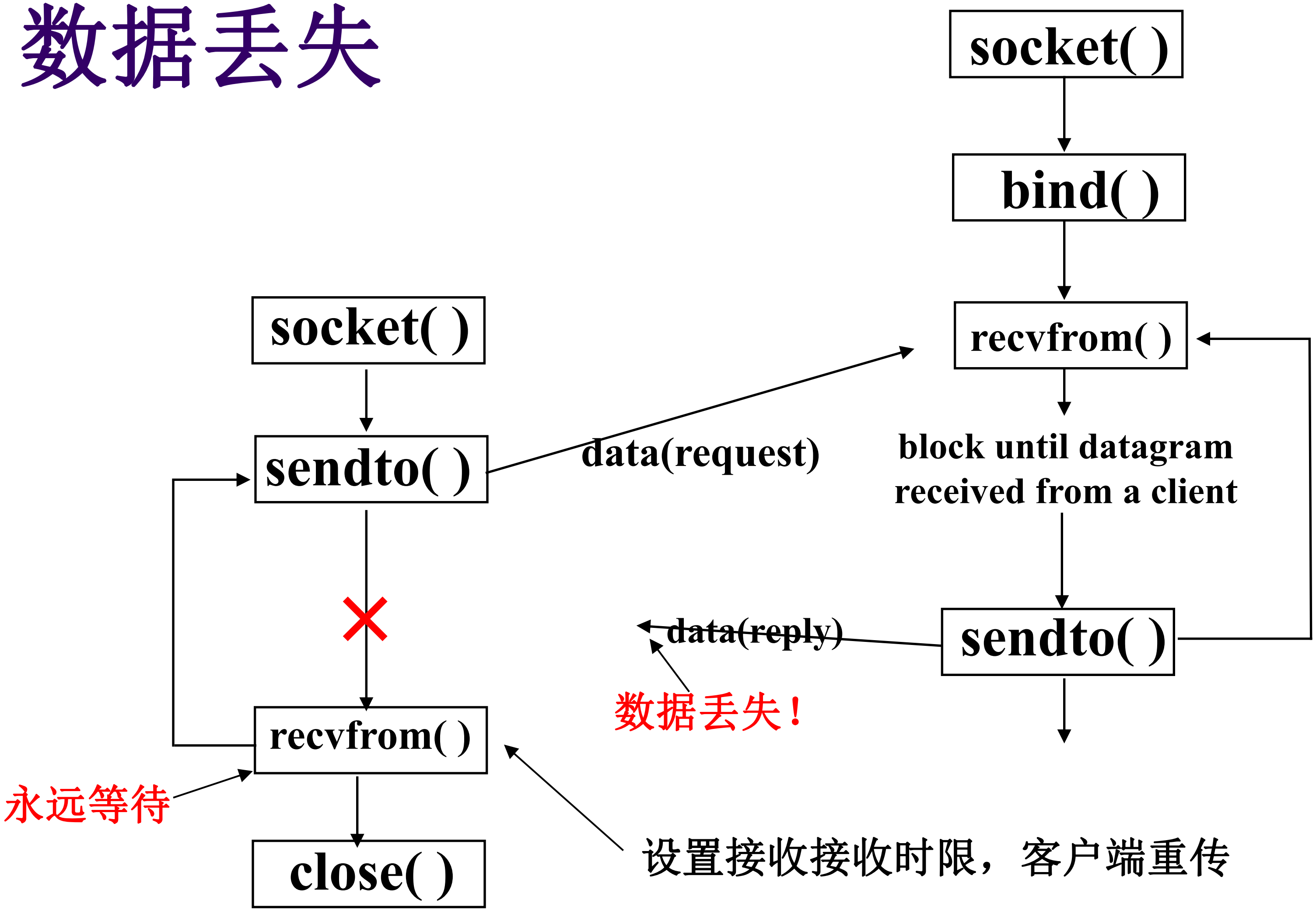
- 网络不可靠：丢包、乱序、抖动、数据错误
- 网络引入了新风险：流量放大、中间人、数据监听
- 网络带来的性能开销：需要考虑性能瓶颈
- 异步引入的复杂性：程序设计复杂性大大提高

- 可靠：
  - **be conservative in what you send, liberal in what you accept**
- 安全：
  - 评估安全风险
  - 建立解决方案
- 性能：吞吐率、响应速度、并发性能
- 一致：同样的输入，同样的结果

可靠性

- 网络传输不可靠
  - 数据包丢失
  - 数据包乱序
  - 传输中的抖动
  - 内容错误（同时校验正确）
- 通信端点的不可靠
  - 客户端、服务器的异常崩溃

可靠性举例（一）
数据丢失

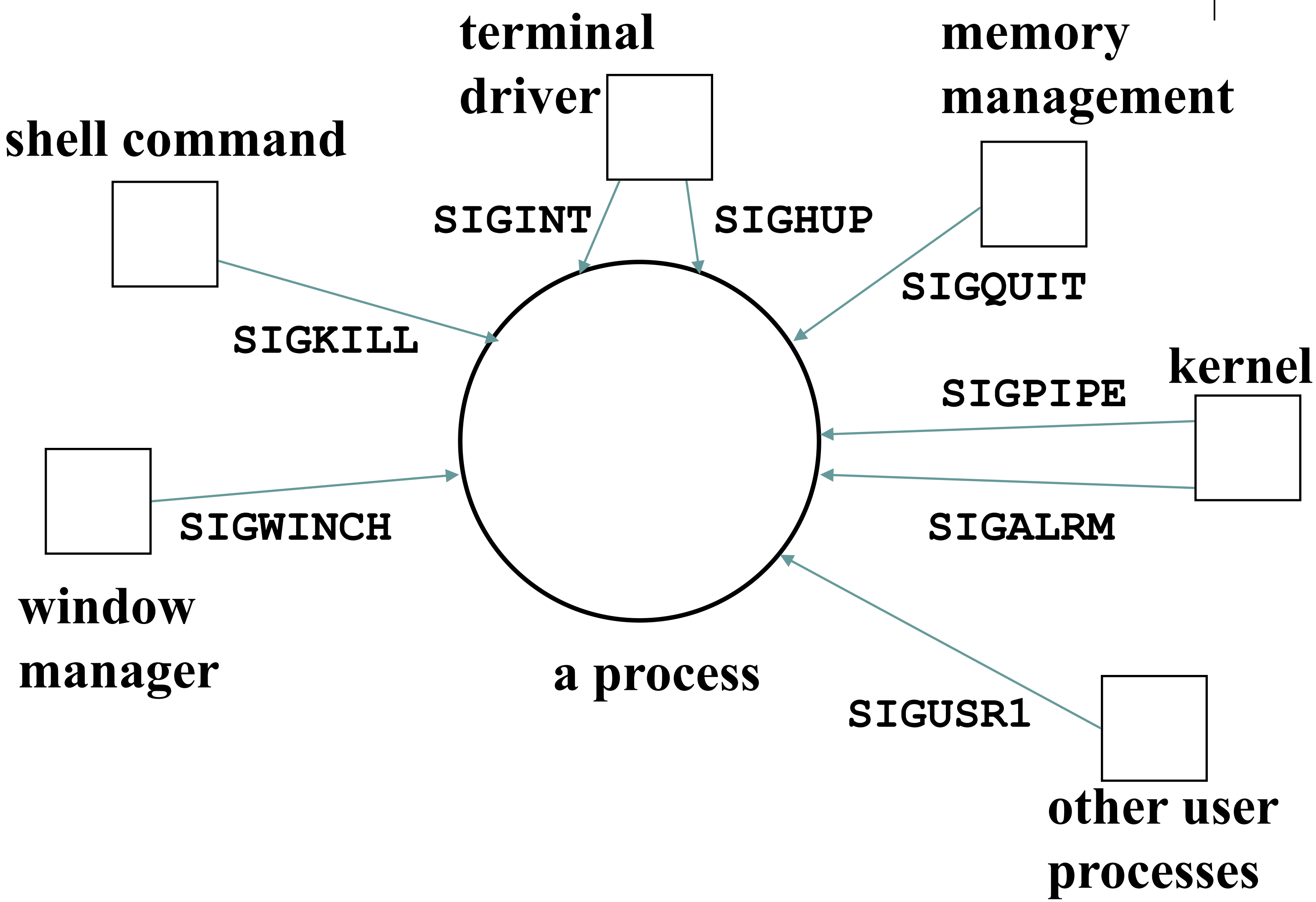


socket( )
bind( )
recvfrom( )
block until datagram received from a client
sendto( )
socket( )
sendto( )
data(request)
~~data(reply)~~
数据丢失！
recvfrom( )
永远等待
close( )
设置接收接收时限，客户端重传

UNIX/LINUX Signal

- 发生原因：产生了异步事件
  - 程序设置产生的事件如定时器
  - 用户操作产生（如**ctrl-c**）
  - 程序运行时出现了问题（内存越界）等
- **Signal**的处理
  - 执行系统默认动作
  - 捕捉信号
    - 设置自己的处理程序
    - 忽略信号（有些信号不能忽略如**SIGKILL, SIGSTOP**和一些硬件异常信号）

信号源



terminal driver
memory management
shell command
SIGINT
SIGHUP
SIGQUIT
SIGKILL
kernel
SIGPIPE
SIGWINCH
SIGALRM
window manager
a process
SIGUSR1
other user processes

sigaction

- 设置自己的信号处理方式

```
#include <signal.h>
int sigaction(int signo,
              const struct sigaction *act,
              struct sigaction *oldact );
```

```
struct sigaction
  {
    void     (*sa_handler)( int );
        /* action to be taken or SIG_IGN, SIG_DFL */
    sigset_t  sa_mask; /* additional signal to be blocked */
    int       sa_flags;   /* modifies action of the signal */
    void     (*sa_sigaction)( int, siginfo_t *, void * );
  }
```

```
struct sigaction{
        void (*) (int) sa_handler
        sigset_t sa_mask
        int sa_flags
}
```

```
int main()
  {
  struct sigaction act;

  act.sa_handler = ouch;

  sigemptyset( &act.sa_mask );

  act.sa_flags = 0;

              IGINT, &act, 0 );

       {
       printf("Hello World!\n
       sleep(1);
       }
  }
```

```
void ouch (int signo) {
…
}
```

设置信号处理函数为ouch

在信号处理函数运行时阻塞的信号

可以设置sa_flags为0，也可以设置为如下标志：
SA_NOCLDSTOP

修改了 **SIGINT**的处理函数 **(ctrl-C) signal**

用户输入**ctrl-c**

利用信号来检测丢包

```
void sigfunc (int sig) {
        return;
}
```

```
newaction.sa_handler = sigfunc;
sigemptyset (&newaction.sa_mask);
newaction.sa_flags = 0;
if (sigaction (SIGALRM, &newaction, &oldaction) < 0) {
        perror ("sigaction");
        exit (1);
}
```

t秒后

```
alarm (t);            过t秒后产生SIGALRM信号
fromlen = sizeof (from);
recvfrom();           等待数据包到达，阻塞
alarm (0);            取消所有SIGALRM事件
….
```

```c
if (alarm(INIT_RECV_WAIT_TIME)) {
      printf("alarm was already set\n");
}
timeExpired = false;
int recvLen;
memset(recvBuffer, 0, sizeof(recvBuffer));
while ((recvLen = recvfrom(sockfd, recvBuffer,
       RECV_BUFFER_SIZE, 0,
      (struct sockaddr*) &destAddr,
      &socketLen)) <= 0) {
      if (timeExpired) {
            printf("find time expired\n");
            goto tryAgain2;
      }
}
alarm(0);
```

超时

```c
void alarmHandler(int sig) {
      timeExpired = true;
      return;
}
```

主观题 10分

设置

上面的程序有什么问题?

正常使用主观题需2.0以上版本雨课堂

作答

不可靠！

```
if (alarm(INIT_RECV_WAIT_TIME)) {
      printf("alarm was already set\n");
}
timeExpired = false;
int recvLen;
memset(recvBuffer, 0, sizeof(recvBuffer));
while ((recvLen = recvfrom(sockfd, recvBuffer,
       RECV_BUFFER_SIZE, 0,
      (struct sockaddr*) &destAddr,
      &socketLen)) <= 0) {
      if (timeExpired) {
            printf("find time expired\n");
            goto tryAgain2;
      }
}
alarm(0);
```

被调度出运行状态并超时

```
void alarmHandler(int sig) {
      timeExpired = true;
      return;
}
```

利用setsockopt建立读写超时

- setsockopt
  - int setsockopt(int sockfd, int level, int optname,const void *opval, socklen_t optlen);
  - level 指明了该选项是一个通用选项还是一个协议相关选项
- SO_RCVTIMEO, SO_SNDTIMEO
  - 指定时间内没有数据可读，堵塞调用结束并返回错误
  - 指定时间内数据未发送成功，堵塞调用结束并返回错误
  - 缺省为0

```c
void
dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
{
    int     n;
    char        sendline[MAXLINE], recvline[MAXLINE + 1];
    struct timeval  tv;
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    Setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    while (Fgets(sendline, MAXLINE, fp) != NULL) {
        Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
        n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
        if (n < 0) {
            if (errno == EWOULDBLOCK) {
                fprintf(stderr, "socket timeout\n");
                continue;
            } else
                err_sys("recvfrom error");
        }
        recvline[n] = 0;    /* null terminate */
        Fputs(recvline, stdout);
    }
}
```

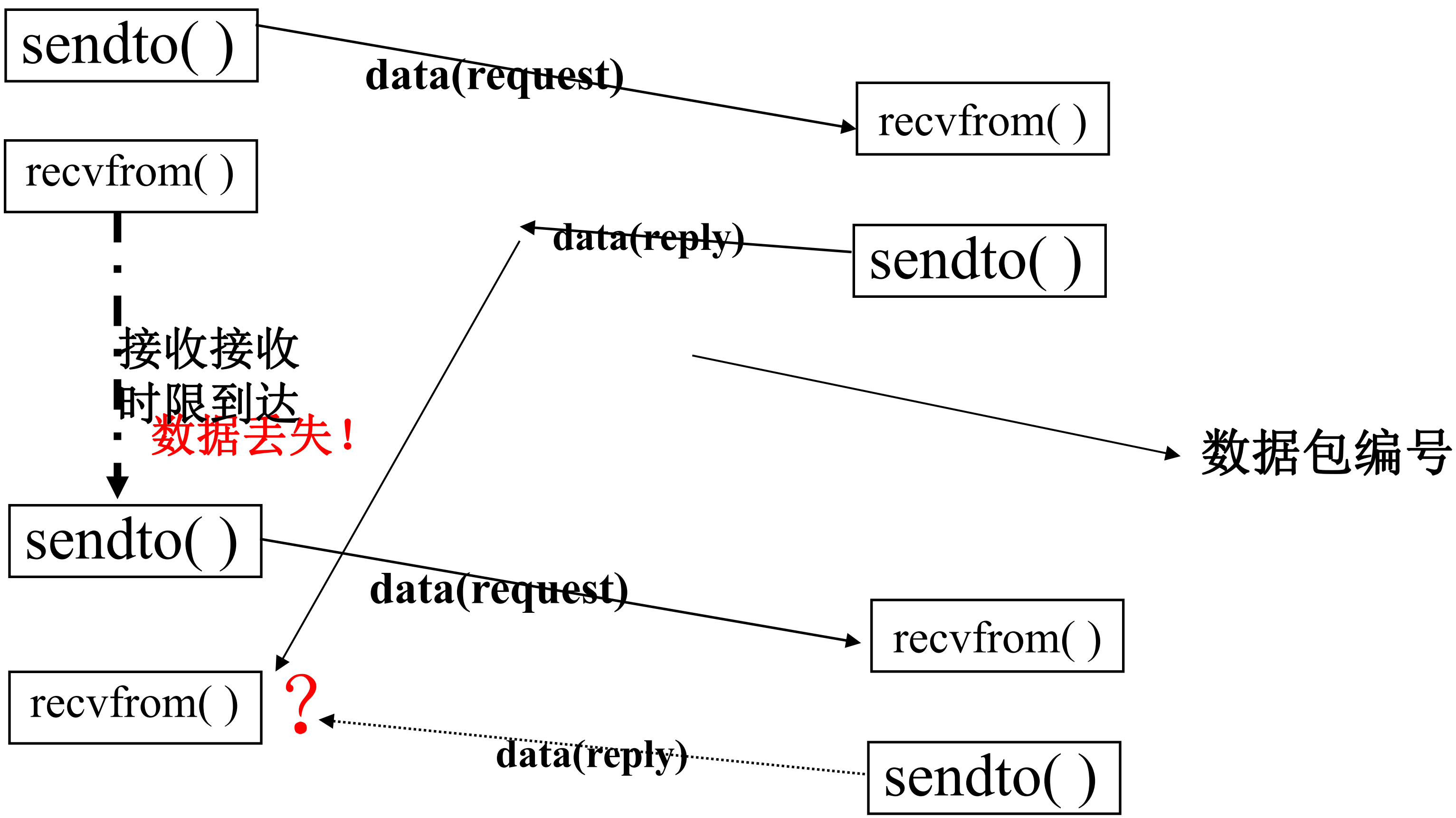
限时读（使用select）

- maxfdp1，最大的待测试fd值加1
- timeval
  - NULL, 直到有fd可读写
  - 等待不超过固定长度时间
  - timeval = 0，立即返回

```
#include <sys/select.h>
#include <sys/time.h>
int select (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset,
const struct timeval *);
struct timeval{
        long  tv_sec;  /* seconds */
        long  tv_usec; /* microseconds */
    }
```

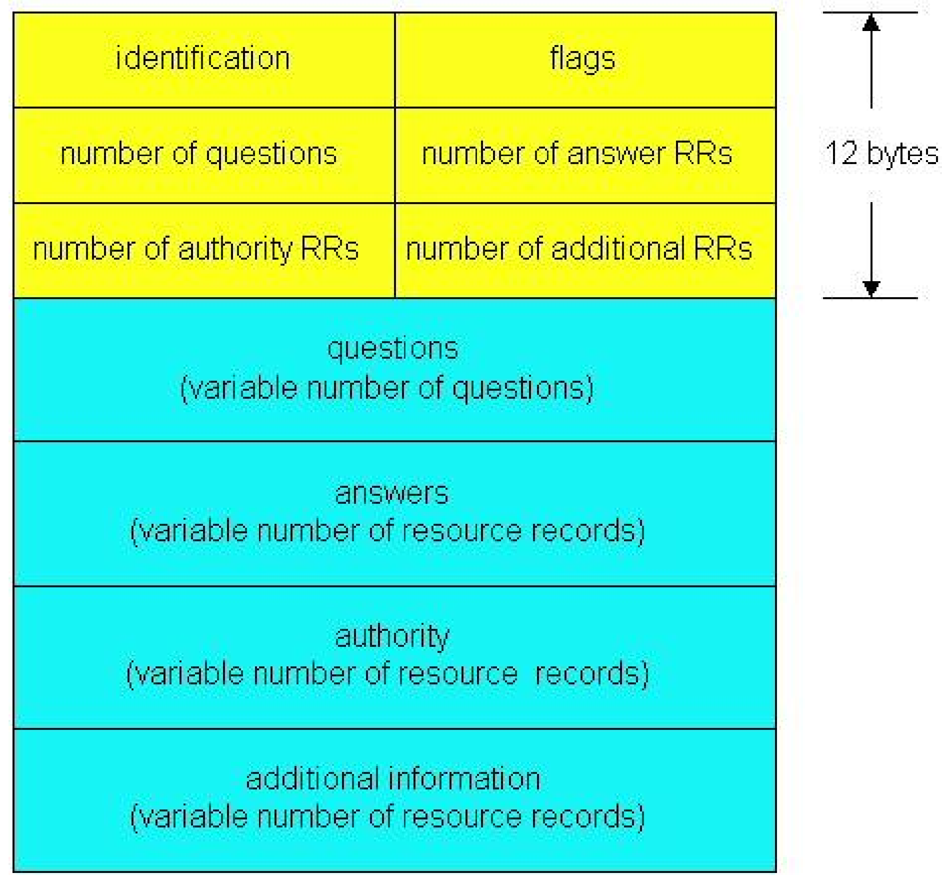
```c
int recv_within_time(int fd, char *buf, size_t buf_n,
                  struct sockaddr* addr,socklen_t *len,
                  unsigned int sec,unsigned usec)
{
 struct timeval tv; fd_set readfds;

 FD_ZERO(&readfds); FD_SET(fd,&readfds);
 tv.tv_sec=sec; tv.tv_usec=usec;
 if (select(fd+1,&readfds,NULL,NULL,&tv) > 0) {
   if(FD_ISSET(fd,&readfds))
    return recvfrom(fd,buf,buf_n,0,addr,len);
 }
 return -1;
}
```

可靠性举例（二）
数据对应


sendto( )
data(request)
recvfrom( )
recvfrom( )
data(reply)
sendto( )
接收接收
时限到达
数据丢失！
数据包编号
sendto( )
data(request)
recvfrom( )
recvfrom( )
?
data(reply)
sendto( )

解决方法：使用标识号

- 使用标识号来进行请求/应答的对应，如**DNS**协议
  - **identification: 16 bit #** 用于查询, 应答报文使用同样的 **#**
  - 而且，**DNS**响应中有问题项
- 当然也可以限定必须经过充分长时间后才能发下一个请求，但是这样会对应用产生较大限制



Q&A