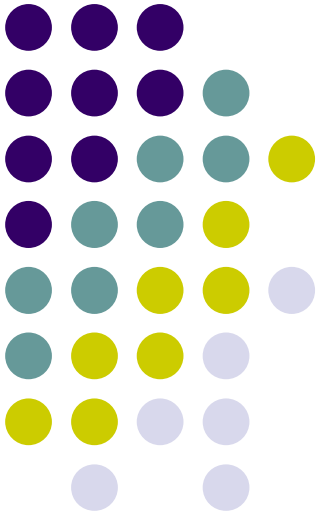
# 网络编程技术

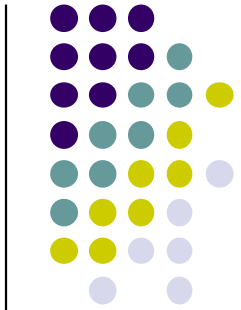
## UDP网络编程（二）

清华大学网研院

张千里

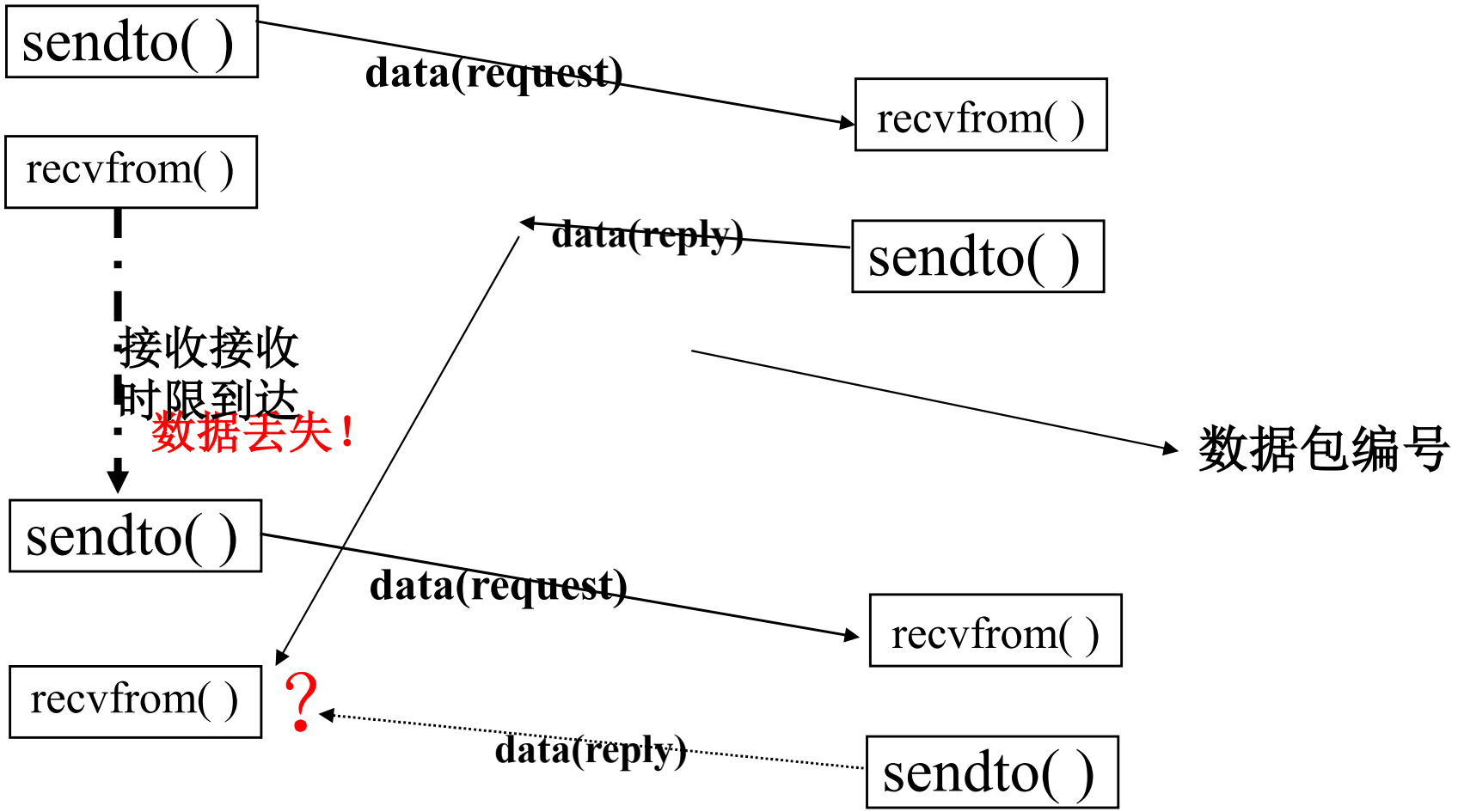


# 本节课内容



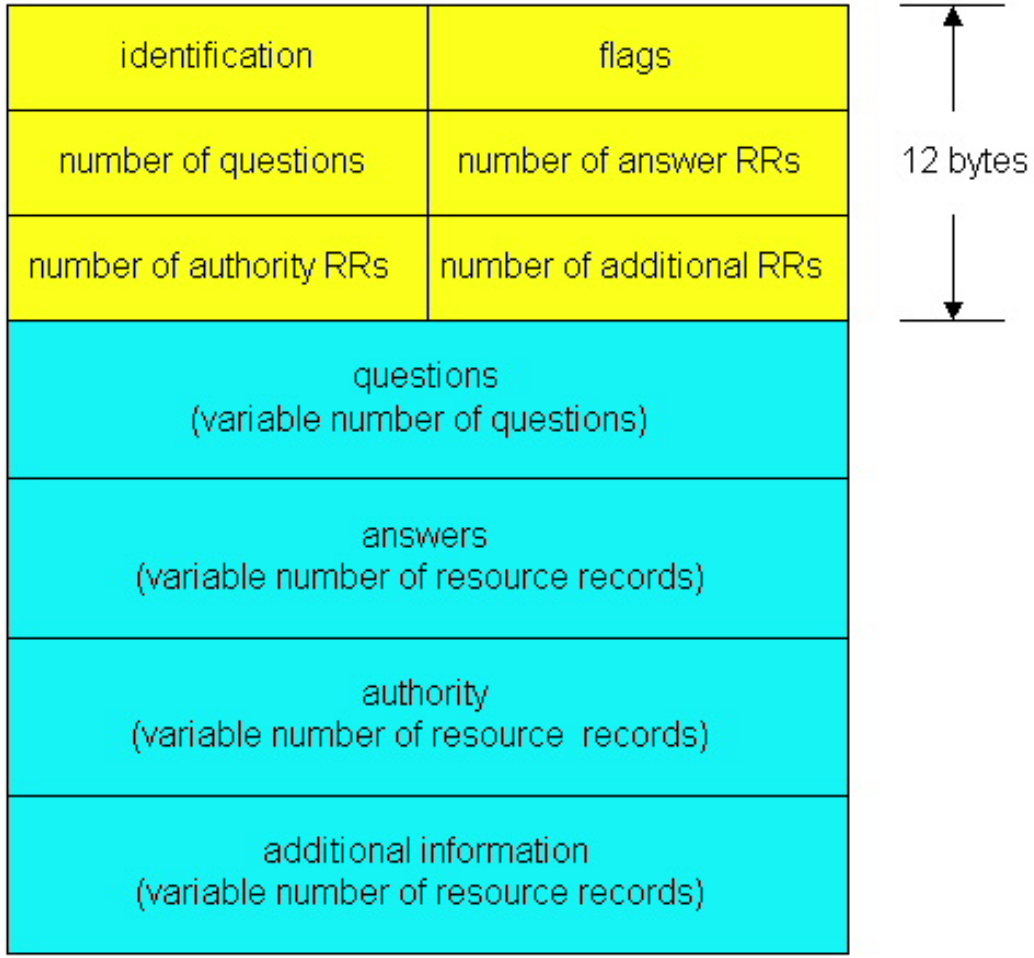
- 主要内容
  - UDP套接字编程
- 参考资料
  - UNIX Network programming：第8、21、22章
    - Elementary UDP Sockets
    - Multicasting
    - Advanced UDP Sockets

# 可靠性举例（二）
# 数据对应

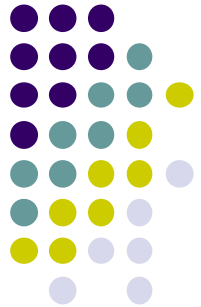

sendto( )
data(request)
recvfrom( )
recvfrom( )
data(reply)
sendto( )
接收接收
时限到达
数据丢失！
数据包编号
sendto( )
data(request)
recvfrom( )
recvfrom( )
?
data(reply)
sendto( )

# 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

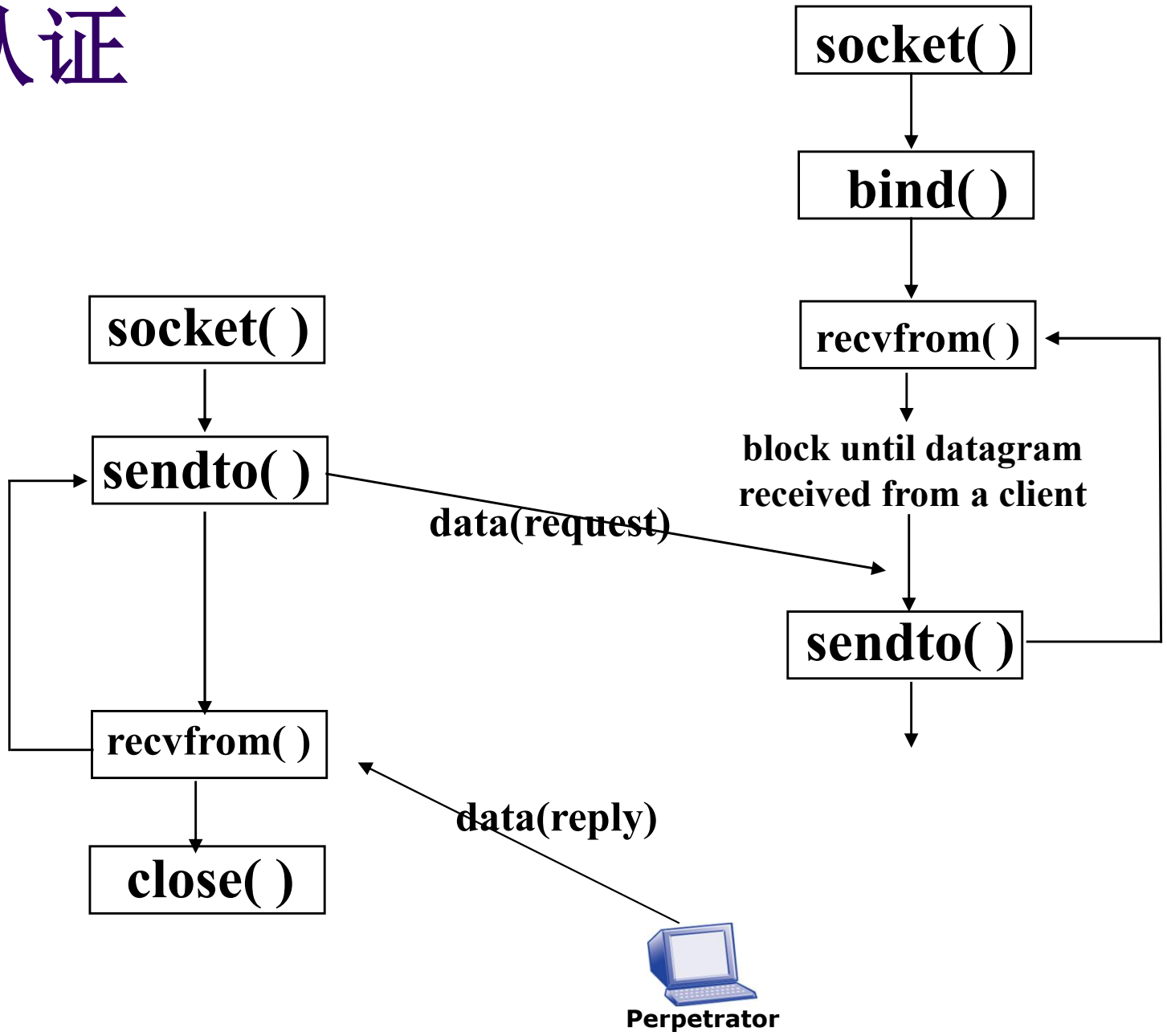


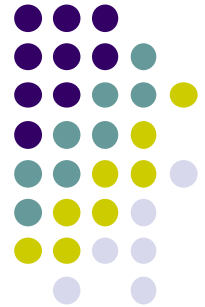
# 安全性



- 客户端所需要考虑的
  - 避免服务器假冒（源认证）
- 服务器所需要考虑的
  - 避免未认证的客户端的服务请求
  - 避免被拒绝服务攻击
  - 避免被利用进行拒绝服务攻击
  - 避免泄露客户的会话内容

# 安全性举例（一）
# 源认证


socket( )
bind( )
recvfrom( )
block until datagram
received from a client
sendto( )
socket( )
sendto( )
data(request)
recvfrom( )
data(reply)
close( )
Perpetrator



## 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);
    }
}
```

接到的数据不源于服务器！

```
        n = Recvfrom(sockfd, recvline, MAXLINE, 0, preply_addr, &len);
        if (len != servlen || memcmp(pservaddr, preply_addr, len) != 0) {
            printf("reply from %s (ignored)\n", Sock_ntop(preply_addr, len));
            continue;
        }
```

# 主观题 10分

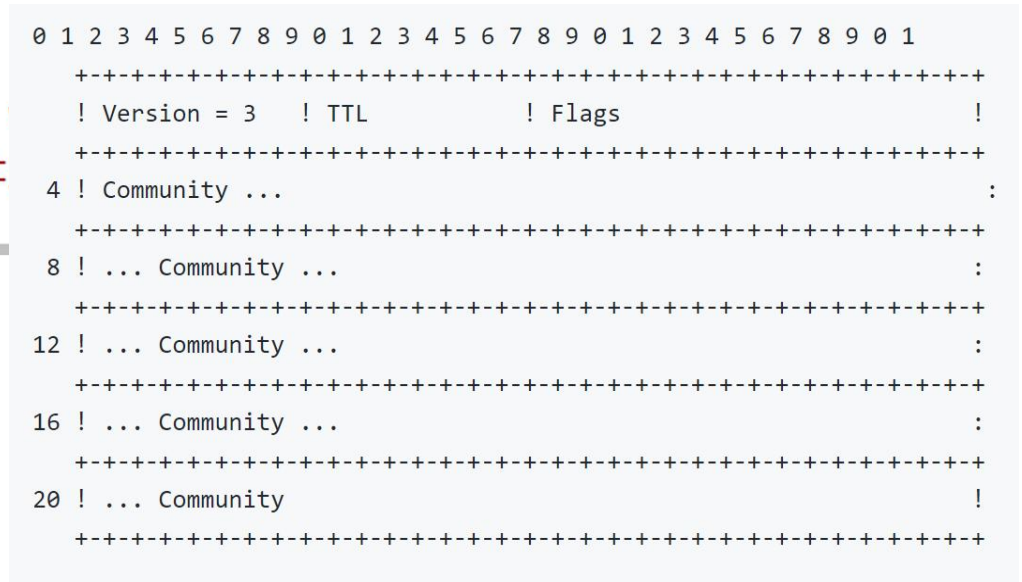
设置

上面的程序是否解决了源认证的问题？你有什么更好的建议？

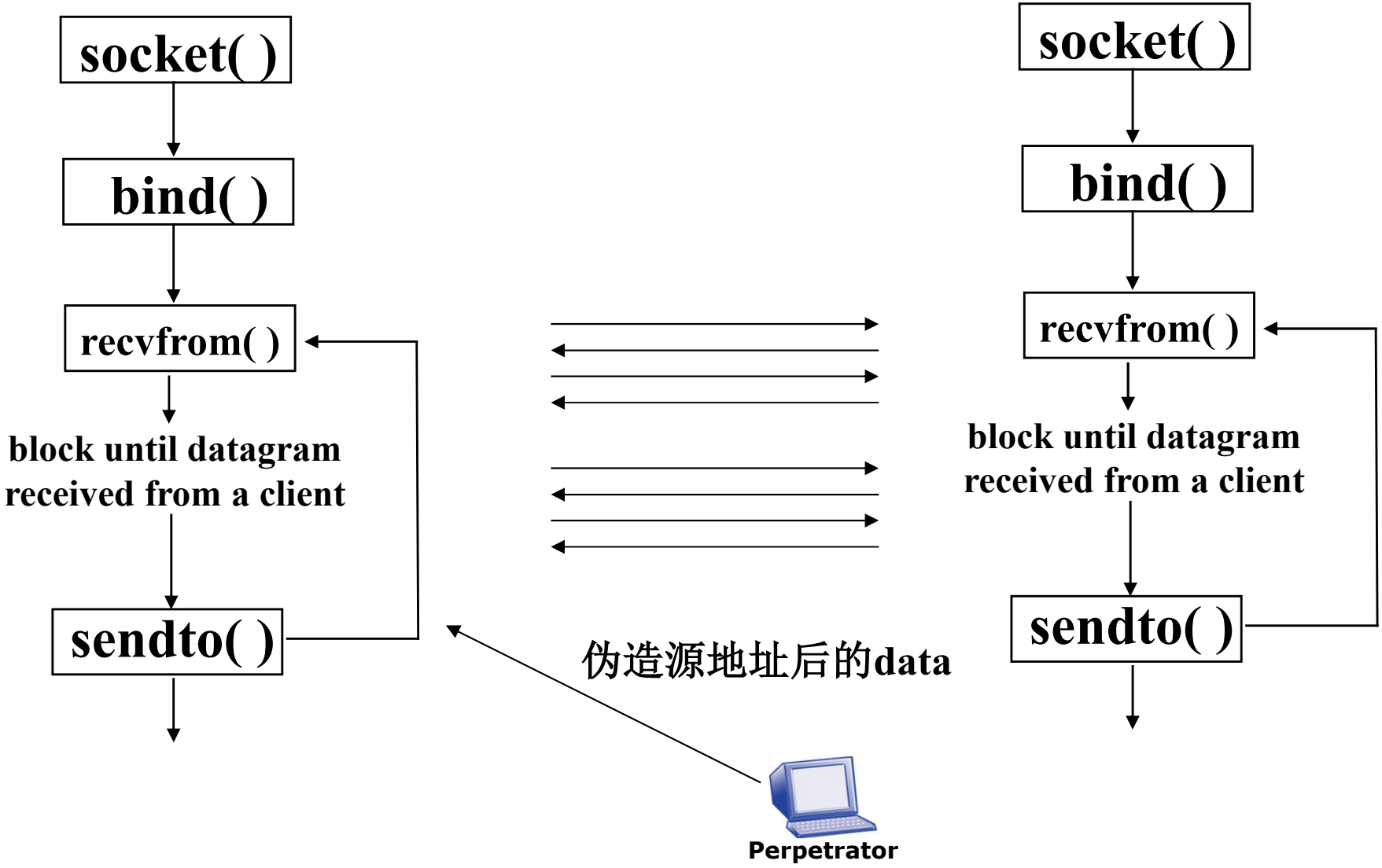
正常使用主观题需2.0以上版本雨课堂

作答

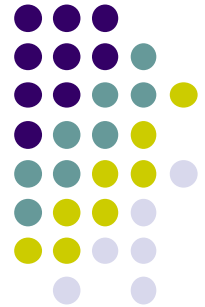
```
if(udp_size < 24) {
    traceEvent(TRACE_DEBUG, "dropped a packet too short to be valid");
    return -1;
}
if((udp_buf[23] == (uint8_t)0x00) // null terminated community name
   && (udp_buf[00] == N2N_PKT_VERSION) // correct packet version
   && ((be16toh(*(uint16_t*)&(udp_buf[02])) & N2N_FLAGS_TYPE_MASK) <= MSG_TYPE_MAX_TYPE) // me
   && ( be16toh(*(uint16_t*)&(udp_buf[02])) < N2N_FLAGS_OPTIONS) // flags
   ) {
    /* most probably unencrypted */
    /* make sure, no downgrading happens here and no unencrypted packets can be
     * injected in a community which definitely deals with encrypted headers */
    HASH_FIND_COMMUNITY(sss->communities, (char *)&udp_buf[04], comm);
    if(comm) {…
} else {
    /* most probably encrypted */
    /* cycle through the known communities (as keys) to eventually decrypt */
    HASH_ITER(hh, sss->communities, comm, tmp) {…
    if(!header_enc) {
        // no matching key/community
        traceEvent(TRACE_DEBUG, "dropped a packet
                   "for which no matching communit
        return -1;
```



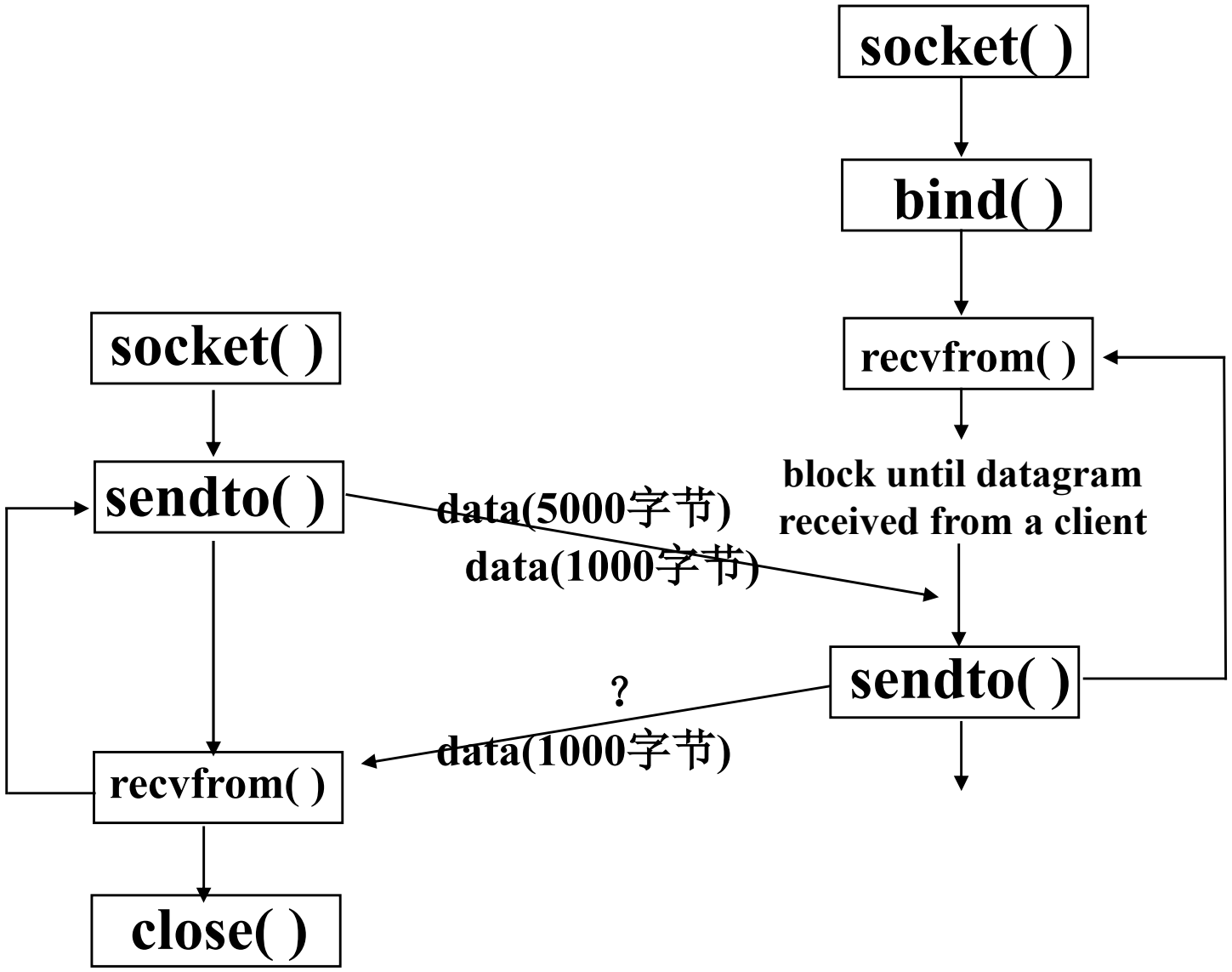
# 安全性举例（二）
# 拒绝服务攻击



# 一致性



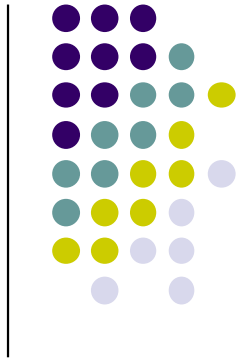
- 协议设计中的一致性考虑，举例：ECHO就是将用户输入的内容返回给用户
  - 问题1：协议如何进行切割？
    - 以回车符进行切割
    - 以一个报文进行切割
  - 问题2：长度是否有限制？
    - MAXLINE的影响
    - UDP最大报文长度的影响
    - MTU的影响
  - 问题3：何时返回？
    - 长请求需要接收到所有数据后返回
    - 每收到一部分后就立即返回
- 服务器实现应当严格遵循协议规范
  - 举例：错误处理，返回错误、直接忽略还是服务器退出？
  - 举例：超长报文，是否会返回部分？


socket( )
bind( )
recvfrom( )
block until datagram
received from a client
sendto( )
data(5000字节)
data(1000字节)
?
data(1000字节)
socket( )
sendto( )
recvfrom( )
close( )

# 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);
```

- flags：功能选项，一般设置为0
  - MSG_TRUNC：指明数据报尾部数据已被丢弃，因为它比所提供的缓冲区需要更多的空间。
    - 一般而言，当收到比buf大的数据包时，剩余部分会被丢弃，返回的长度即buf的大小
    - 使用MSG_TRUNC之后，返回的长度为整个数据包的长度



```
char buffer[MAX_SIZE];
struct sockaddr_in from;
struct sockaddr_in servaddr;
socklen_t fromlen;
int available_data;
int sock;
sock = socket(AF_INET, SOCK_DGRAM, 0);
fromlen = sizeof (from);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(2000);
bind(sock, (struct sockaddr *) &servaddr, sizeof(servaddr));
available_data = recvfrom(sock, (char*)buffer, MAX_SIZE,
        MSG_TRUNC, (struct sockaddr *)&from, &fromlen);
if (available_data > MAX_SIZE) {
    fprintf(stderr, "UDP Packet is bigger than expected\n");
    exit(EXIT_FAILURE);
}
```

# 其他可选的标志

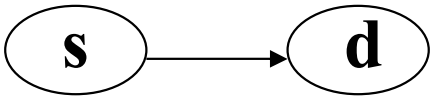
| *flags* | Description | recv | send |
|---|---|---|---|
| MSG_DONTROUTE | 不进行路由表查询 | | X |
| MSG_DONTWAIT | 非阻塞 | X | X |
| MSG_OOB | 带外数据 | X | X |
| MSG_PEEK | 不清除接收队列 | X | |
| MSG_WAITALL | 阻塞到读满数据为止（除非信号或者其他数据到来） | X | |

# 组播和广播

- 广播和组播不能在TCP下工作
  - 使用广播的协议
    - DHCP
    - NTP
  - IPv6大量使用组播

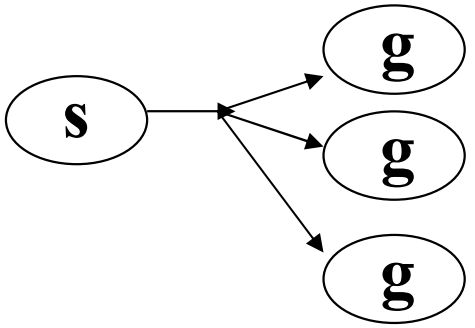
单播

unicast




组播

multicast




| Type | IPv4 | IPv6 | TCP | UDP | # IP interfaces identified | # IP interfaces delivered to |
|---|---|---|---|---|---|---|
| Unicast | • | • | • | • | One | One |
| Anycast | * | • | Not yet | • | A set | One in set |
| Multicast | opt. | • | | • | A set | All in set |
| Broadcast | • | | | • | All | All |

# IPv4组播地址

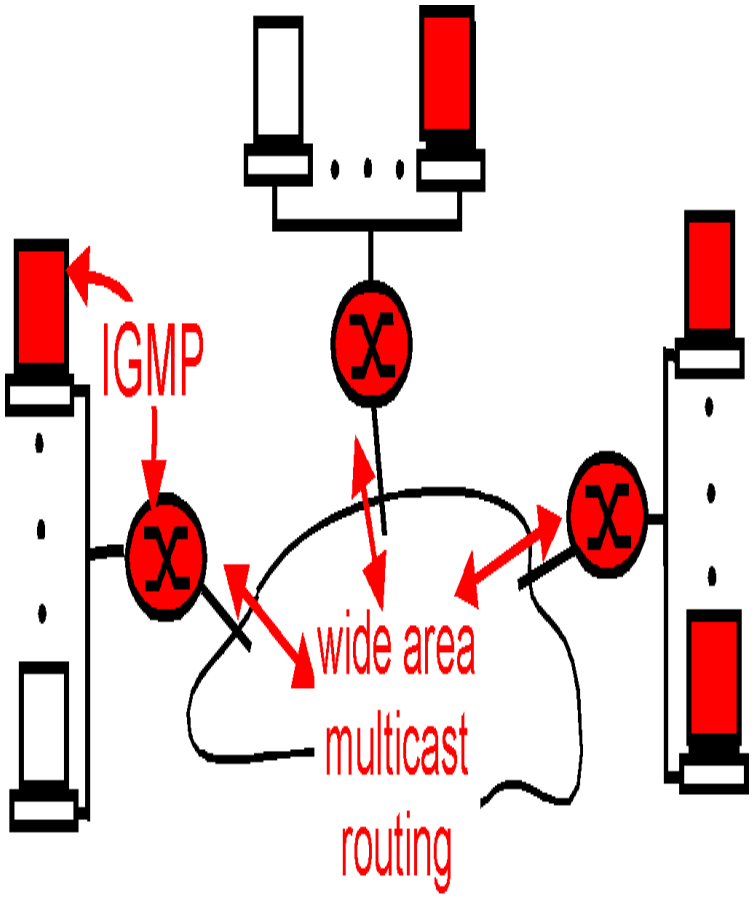
- 预定义的组播地址
  - **224.0.0.1**：子网里的所有系统
  - **224.0.0.2**：子网里的所有路由器
  - …

**Figure 21.3. Scope of IPv4 and IPv6 multicast addresses.**

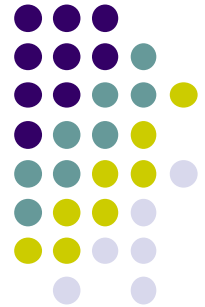
| Scope | IPv6 scope | IPv4 | |
|---|---|---|---|
| | | TTL scope | Administrative scope |
| Interface-local | 1 | 0 | |
| Link-local | 2 | 1 | 224.0.0.0 to 224.0.0.255 |
| Site-local | 5 | <32 | 239.255.0.0 to 239.255.255.255 |
| Organization-local | 8 | | 239.192.0.0 to 239.195.255.255 |
| Global | 14 | ≤255 | 224.0.1.0 to 238.255.255.255 |

# IGMP

- 主机通过IGMP协议，向和自己相连的路由器报告组播组的加入、退出
- 当主机中的一个进程首次加入某个组时，主机向路由器发送报告，申请加入、退出组
- 路由器定期进行询问，主机响应
- 主机退出一个组时，不告知路由器，而是通过定期询问发现



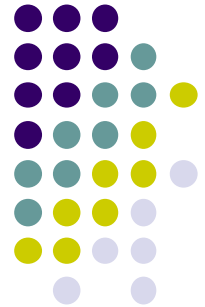
# 组播套接字选项



| Command | Datatype | Description |
|---|---|---|
| IP_MULTICAST_IF | struct in_addr | Specify default interface for outgoing multicasts |
| IP_MULTICAST_TTL | u_char | Specify TTL for outgoing multicasts |
| IP_MULTICAST_LOOP | u_char | Enable or disable loopback of outgoing multicasts |
| IPV6_MULTICAST_IF | u_int | Specify default interface for outgoing multicasts |
| IPV6_MULTICAST_HOPS | int | Specify hop limit for outgoing multicasts |
| IPV6_MULTICAST_LOOP | u_int | Enable or disable loopback of outgoing multicasts |

| Command | Datatype | Description |
|---|---|---|
| IP_ADD_MEMBERSHIP | struct ip_mreq | Join a multicast group |
| IP_DROP_MEMBERSHIP | struct ip_mreq | Leave a multicast group |
| IP_BLOCK_SOURCE | struct ip_mreq_source | Block a source from a joined group |
| IP_UNBLOCK_SOURCE | struct ip_mreq_source | Unblock a previously blocked source |
| IP_ADD_SOURCE_MEMBERSHIP | struct ip_mreq_source | Join a source-specific group |
| IP_DROP_SOURCE_MEMBERSHIP | struct ip_mreq_source | Leave a source-specific group |
| IPV6_JOIN_GROUP | struct ipv6_mreq | Join a multicast group |
| IPV6_LEAVE_GROUP | struct ipv6_mreq | Leave a multicast group |
| MCAST_JOIN_GROUP | struct group_req | Join a multicast group |
| MCAST_LEAVE_GROUP | struct group_req | Leave a multicast group |
| MCAST_BLOCK_SOURCE | struct group_source_req | Block a source from a joined group |
| MCAST_UNBLOCK_SOURCE | struct group_source_req | Unblock a previously blocked source |
| MCAST_JOIN_SOURCE_GROUP | struct group_source_req | Join a source-specific group |
| MCAST_LEAVE_SOURCE_GROUP | struct group_source_req | Leave a source-specific group |

# 组播套接字选项（2）

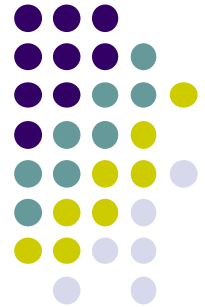


- IP_ADD_MEMBERSHIP, IPV6_ADD_MEMBERSHIP
  - 通过指定的接口加入组
- IP_DROP_MEMBERSHIP, IPV6_DROP_MEMBERSHIP
  - 通过指定的接口退出组

```
Struct ip_mreq{
struct in_addr imr_multiaddr;
struct in_addr imr_interface;
};
```

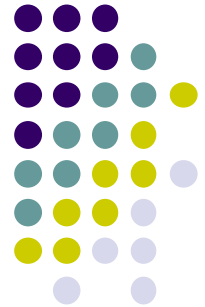
```
Struct ipv6_mreq{
struct in6_addr ipv6mr_multiaddr;
struct int ipv6mr_interface;
};
```

# 组播套接字选项（3）

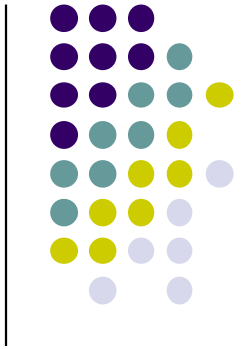


- IP_MULTICAST_IF, IPV6_MULTICAST_IF
  - 指定套接字发送组播数据包时使用的接口
  - 这个接口在IPv4中是in_addr结构，而对于IPv6，则可以是接口序号（无符号整数）
- IP_MULTICAST_TTL, IPV6_MULTICAST_TTL
  - 设置组播数据包的TTL（IPv4）或者跳数上限（IPv6）
  - 未设置的话缺省为1
- IP_MULTICAST_LOOP, IPV6_MULTICAST_LOOP
  - 允许/禁止接受本地发送的组播报文（缺省允许）

# UNP中的例子（lib/mcast_join.c）
# mcast_join及其他

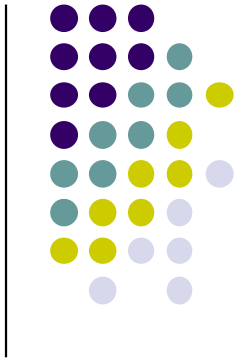


```
#include “unp.h”
int mcast_join(int sockfd, const struct sockaddr *addr, socklen_t salen, const
    char *ifname, u_int ifindex);
int mcast_leave(int sockfd, const struct sockaddr *addr, socklen_t salen);
int mcast_set_if(int sockfd, const char *ifname, u_int ifindex);
int mcast_set_loop(int sockfd, int flag);
int mcast_set_ttl(int sockfd, int ttl);
                                    All above return :0 if ok, -1 on error
int mcast_get_if(int sockfd);
                                    return : nonnegative interface index if OK, -1 error
int mcast_get_loop(int sockfd);
                                    return : current loopback flag if OK, -1 error
int mcast_get_ttl(int sockfd);
                                    return : current TTL or hop limit if OK, -1 error
```



```
int
mcast_join(int sockfd, const SA *grp, socklen_t grplen,
                   const char *ifname, u_int ifindex)
{
#ifdef MCAST_JOIN_GROUP
        struct group_req req;
        if (ifindex > 0) {
                req.gr_interface = ifindex;
        } else if (ifname != NULL) {
                if ( (req.gr_interface = if_nametoindex(ifname)) == 0) {
                        errno = ENXIO;  /* i/f name not found */
                        return(-1);
                }
        } else
                req.gr_interface = 0;
        if (grplen > sizeof(req.gr_group)) {
                errno = EINVAL;
                return -1;
        }
        memcpy(&req.gr_group, grp, grplen);
        return (setsockopt(sockfd, family_to_level(grp->sa_family),
                        MCAST_JOIN_GROUP, &req, sizeof(req)));
```

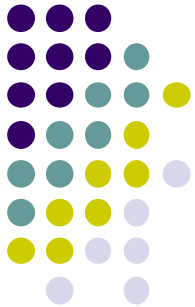
```
        switch (grp->sa_family) {
        case AF_INET: {
                struct ip_mreq          mreq;
                struct ifreq            ifreq;

                memcpy(&mreq.imr_multiaddr,
                        &((const struct sockaddr_in *) grp)->sin_addr,
                        sizeof(struct in_addr));

                if (ifindex > 0) {
                        if (if_indextoname(ifindex, ifreq.ifr_name) == NULL) {
                                errno = ENXIO;  /* i/f index not found */
                                return(-1);
                        }
                        goto doioctl;
                } else if (ifname != NULL) {
                        strncpy(ifreq.ifr_name, ifname, IFNAMSIZ);
doioctl:
                        if (ioctl(sockfd, SIOCGIFADDR, &ifreq) < 0)
                                return(-1);
                        memcpy(&mreq.imr_interface,
                                &((struct sockaddr_in *) &ifreq.ifr_addr)->sin_addr
                                sizeof(struct in_addr));
                } else
                        mreq.imr_interface.s_addr = htonl(INADDR_ANY);

                return(setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
                                        &mreq, sizeof(mreq)));
        }
```



```
case AF_INET6: {
        struct ipv6_mreq        mreq6;

        memcpy(&mreq6.ipv6mr_multiaddr,
                    &((const struct sockaddr_in6 *) grp)->sin6_addr,
                    sizeof(struct in6_addr));

        if (ifindex > 0) {
                mreq6.ipv6mr_interface = ifindex;
        } else if (ifname != NULL) {
                if ( (mreq6.ipv6mr_interface = if_nametoindex(ifname)) == 0) {
                        errno = ENXIO;  /* i/f name not found */
                        return(-1);
                }
        } else
                mreq6.ipv6mr_interface = 0;

        return(setsockopt(sockfd, IPPROTO_IPV6, IPV6_JOIN_GROUP,
                                        &mreq6, sizeof(mreq6)));
}
```

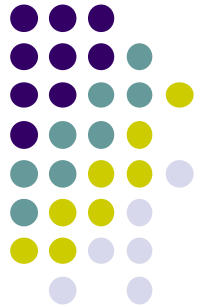
```
int
mcast_set_loop(int sockfd, int onoff)
{
        switch (sockfd_to_family(sockfd)) {
        case AF_INET: {
                u_char          flag;

                flag = onoff;
                return(setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_LOOP,
                                  &flag, sizeof(flag)));
        }

#ifdef  IPV6
        case AF_INET6: {
                u_int           flag;

                flag = onoff;
                return(setsockopt(sockfd, IPPROTO_IPV6, IPV6_MULTICAST_LOOP,
                                  &flag, sizeof(flag)));
        }
#endif

        default:
                errno = EAFNOSUPPORT;
                return(-1);
        }
```

# mcast/main.c

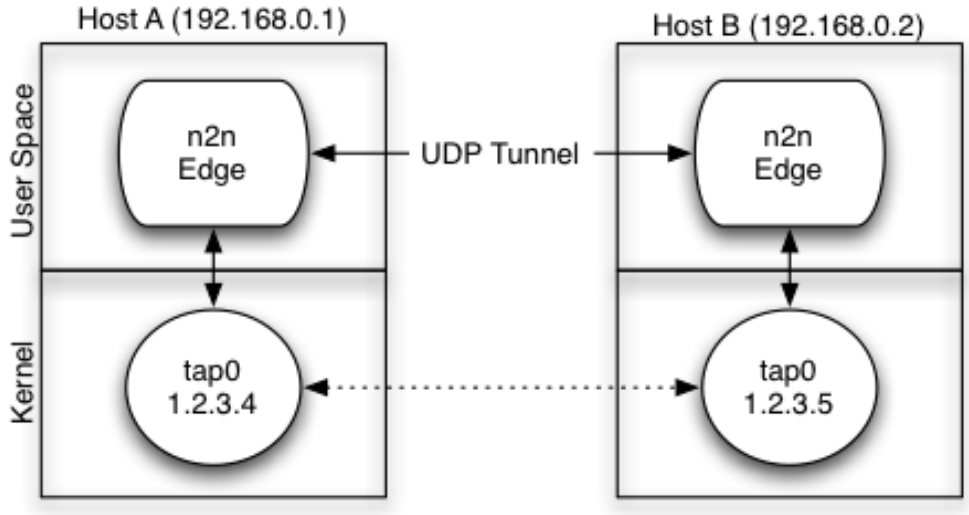


```
if (argc != 3)
        err_quit("usage: sendrecv <IP-multicast-address> <port#>");

sendfd = Udp_client(argv[1], argv[2], (void **) &sasend, &salen);

recvfd = Socket(sasend->sa_family, SOCK_DGRAM, 0);

Setsockopt(recvfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

sarecv = Malloc(salen);
memcpy(sarecv, sasend, salen);
Bind(recvfd, sarecv, salen);

Mcast_join(recvfd, sasend, salen, NULL, 0);
Mcast_set_loop(sendfd, 0);

if (Fork() == 0)
        recv_all(recvfd, salen);                /* child -> receives */

send_all(sendfd, sasend, salen);        /* parent -> sends */
```

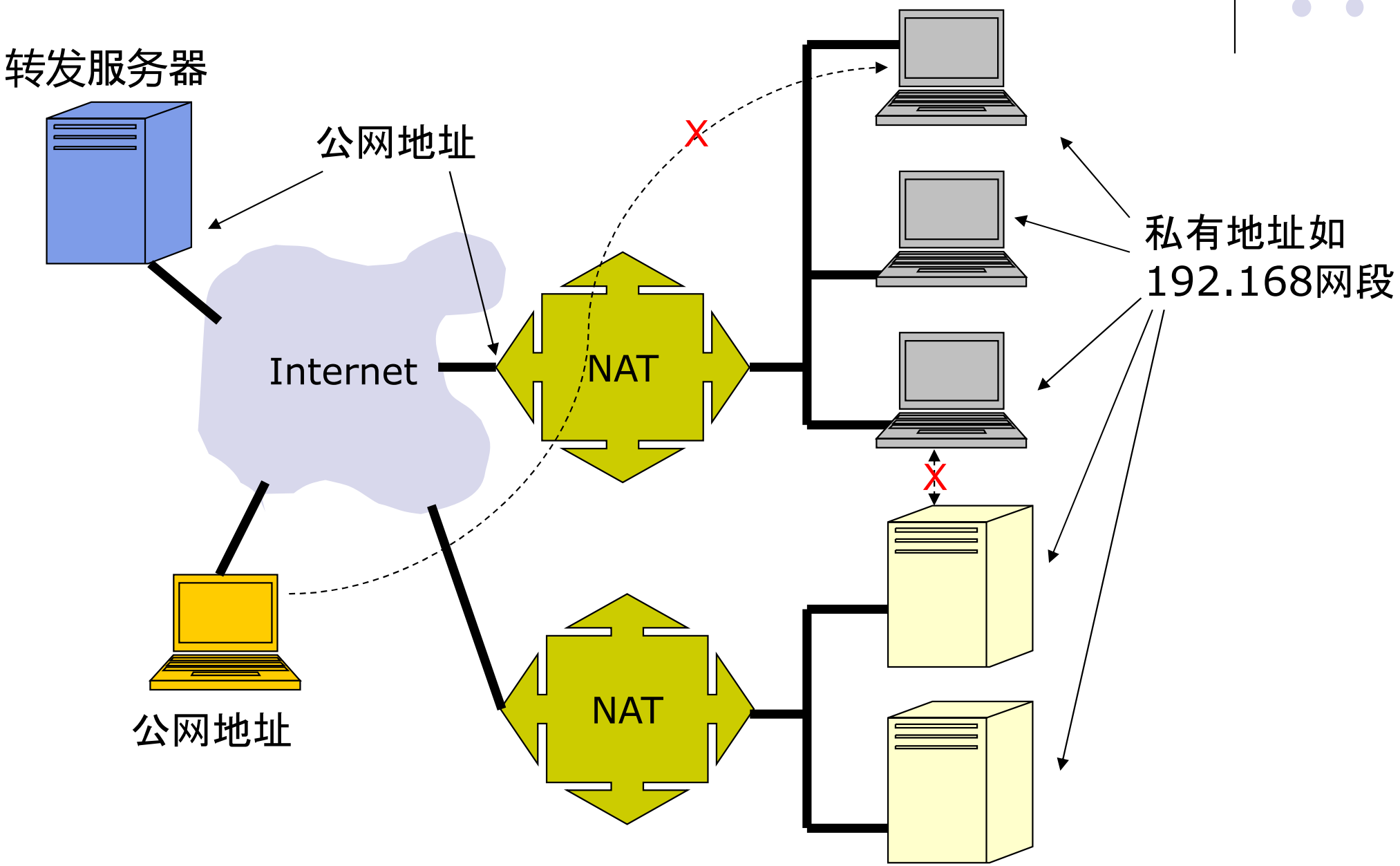
# UDP实际项目介绍
# n2n



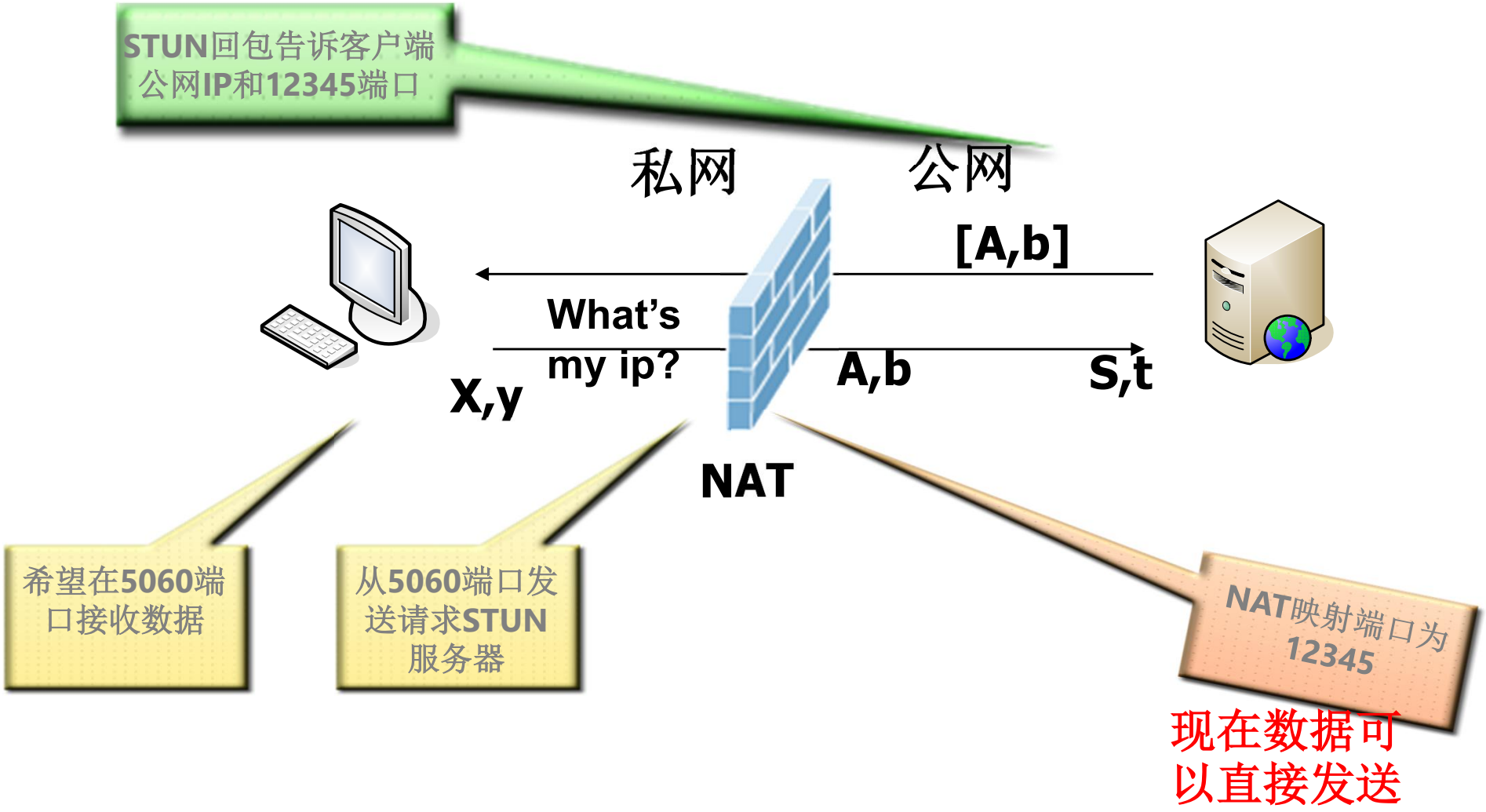
- https://www.ntop.org/products/n2n/
- 设计需求
  - 边缘节点之间的流量加密
  - 边缘节点之间尽可能直接通讯，而不需要转发



# NAT

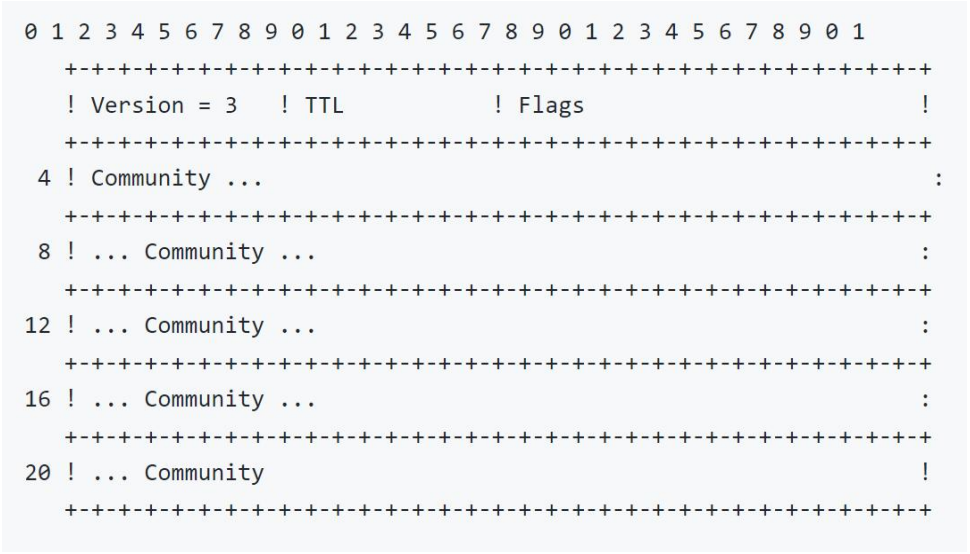


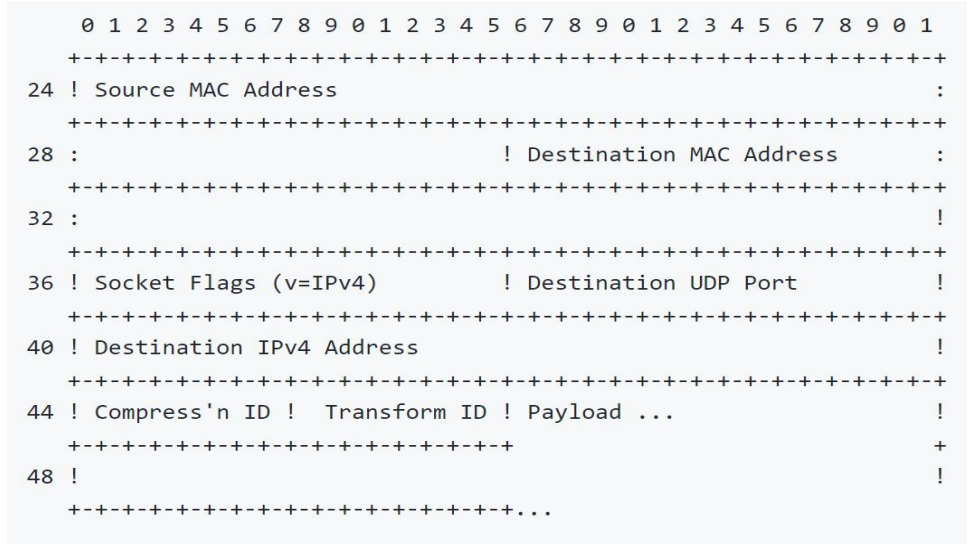
# NAT穿越STUN


STUN回包告诉客户端
公网IP和12345端口
私网
公网
[A,b]
What's
my ip?
X,y
A,b
S,t
NAT
希望在5060端
口接收数据
从5060端口发
送请求STUN
服务器
NAT映射端口为
12345
现在数据可
以直接发送

# n2n（二）协议设计

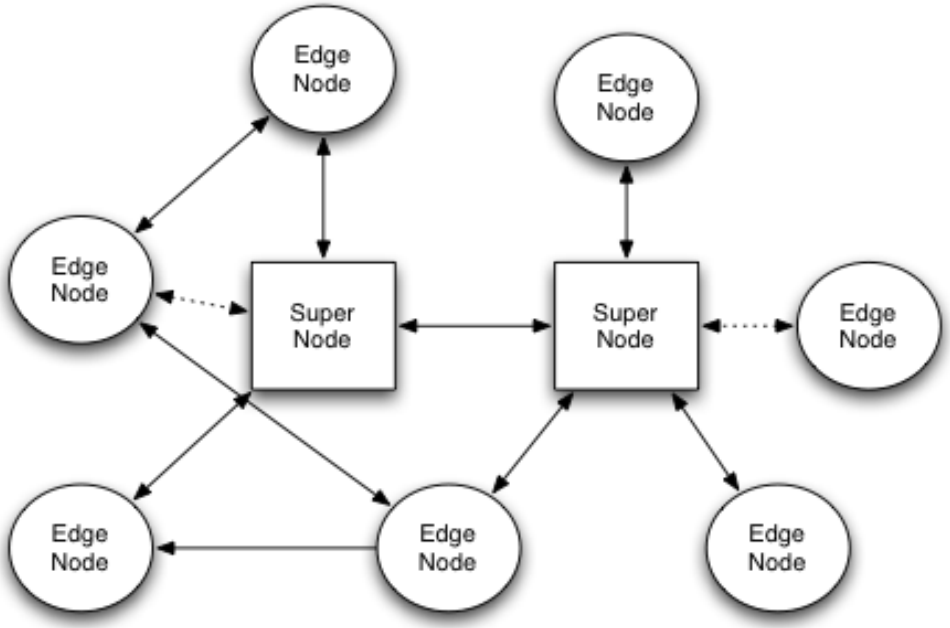
- 每种原语通用头部
- EDGE间通讯（SN转发）
  - PACKET
  - REGISTER
  - REGISTER_ACK
  - PEER_INFO
- 与SN的通讯
  - QUERY_PEER
  - REGISTER_SUPER
  - UNREGISTER_SUPER
  - REGISTER_SUPER_OK
  - REGISTER_SUPER_NAK



# n2n（三）协议状态

- （可选）EDGE向SN发送QUERY_PEER
- EDGE向SN发送REGISTER_SUPER
- 当发送数据时，向目标发送REGISTR，在成功后发送PACKET
- 如果打洞不成功，则PACKET还需要SN转发



# 主观题 10分

设置

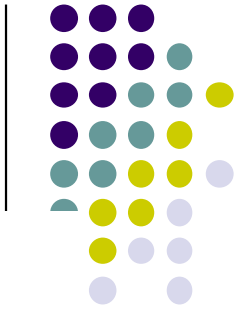
请问以下的代码存在哪些问题？

```
receive = Recvfrom(server_socket, receive_buffer, SERVER_BUFFER, 0,
                                    (sockaddr *)&fromaddr, &addrlen);
version = ntohl(*((unsigned int *)receive_buffer));
switch(version){
        case 1: {….};
        case 2: {….};
        default:
                version = 1; // default to version 1
                break;
}
…
```

正常使用主观题需2.0以上版本雨课堂

作答

# n2n（三）协议实现



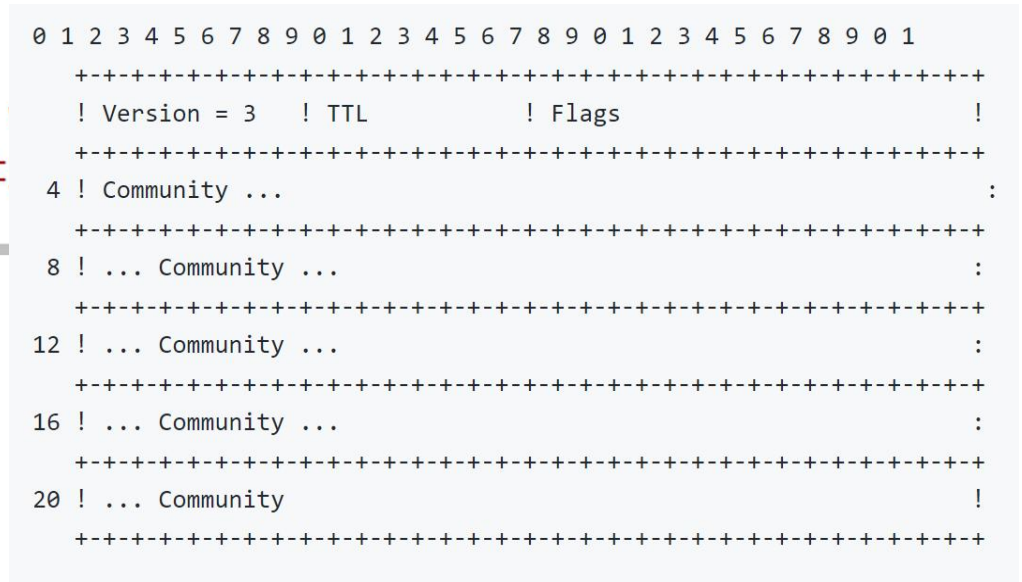
- 原语定义
  - n2n_typedefs.h
- 协议处理
  - edge_utils.c：process_udp
  - supernode.c：process_udp

```
typedef struct n2n_PACKET {
    n2n_mac_t      srcMac;
    n2n_mac_t      dstMac;
    n2n_sock_t     sock;
    uint8_t        transform;
    uint8_t        compression;
} n2n_PACKET_t;
```

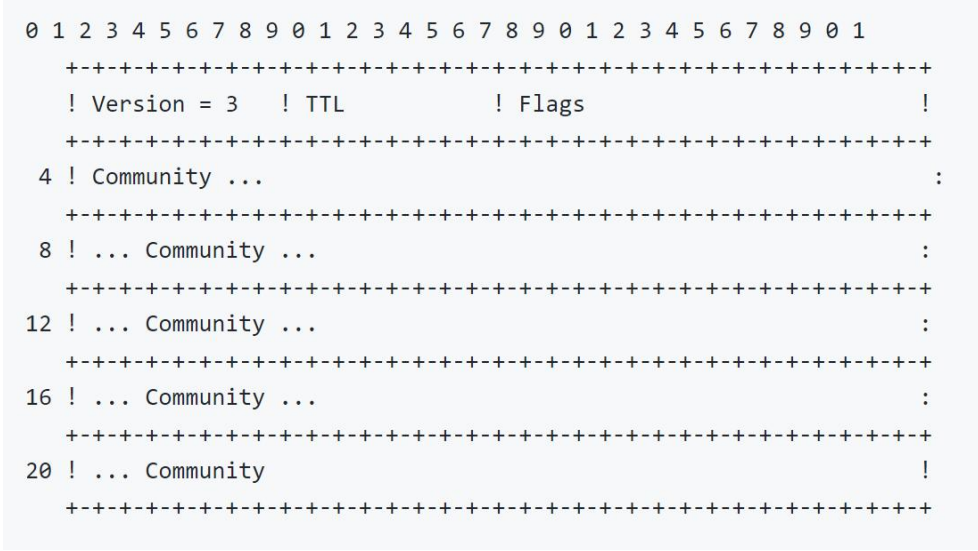
```
switch(msg_type) {
    case MSG_TYPE_PACKET: {
```

```
rc = select(max_sock + 1, &socket_mask, NULL, NULL, &wait_time);

now = time(NULL);

if(rc > 0) {

    // external udp
    if(FD_ISSET(sss->sock, &socket_mask)) {
        struct sockaddr_in sender_sock;
        socklen_t i;

        i = sizeof(sender_sock);
        bread = recvfrom(sss->sock, (void *)pktbuf, N2N_SN_PKTBUF_SIZE, 0 /*flags*/,
                         (struct sockaddr *)&sender_sock, (socklen_t *)&i);
```

```
if(udp_size < 24) {
    traceEvent(TRACE_DEBUG, "dropped a packet too short to be valid");
    return -1;
}
if((udp_buf[23] == (uint8_t)0x00) // null terminated community name
   && (udp_buf[00] == N2N_PKT_VERSION) // correct packet version
   && ((be16toh(*(uint16_t*)&(udp_buf[02])) & N2N_FLAGS_TYPE_MASK) <= MSG_TYPE_MAX_TYPE) // me
   && ( be16toh(*(uint16_t*)&(udp_buf[02])) < N2N_FLAGS_OPTIONS) // flags
   ) {
    /* most probably unencrypted */
    /* make sure, no downgrading happens here and no unencrypted packets can be
     * injected in a community which definitely deals with encrypted headers */
    HASH_FIND_COMMUNITY(sss->communities, (char *)&udp_buf[04], comm);
    if(comm) {…
} else {
    /* most probably encrypted */
    /* cycle through the known communities (as keys) to eventually decrypt */
    HASH_ITER(hh, sss->communities, comm, tmp) {…
    if(!header_enc) {
        // no matching key/community
        traceEvent(TRACE_DEBUG, "dropped a packet
                   "for which no matching communit
        return -1;
```



# n2n（四）协议解析

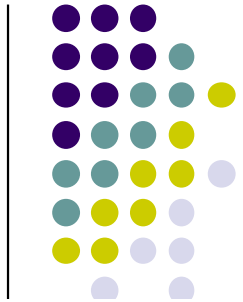



- wire.c

```
int decode_common (n2n_common_t * out,
                   const uint8_t * base,
                   size_t * rem,
                   size_t * idx) {

    size_t idx0 = *idx;
    uint8_t dummy = 0;

    decode_uint8(&dummy, base, rem, idx);

    if(N2N_PKT_VERSION != dummy) {
        return -1;
    }

    decode_uint8(&(out->ttl), base, rem, idx);
    decode_uint16(&(out->flags), base, rem, idx);
    out->pc = (out->flags & N2N_FLAGS_TYPE_MASK);
    out->flags &= N2N_FLAGS_BITS_MASK;

    decode_buf(out->community, N2N_COMMUNITY_SIZE, base, rem, idx);

    return (*idx - idx0);
```

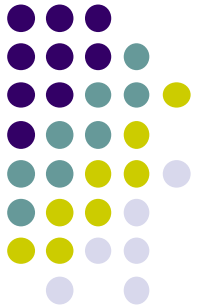
```
    if(*rem < 2) {
        return 0;
    }

    *out  = ( base[*idx] & 0xff ) << 8;
    *out |= ( base[1 + *idx] & 0xff );
    *idx += 2;
    *rem -= 2;

    return 2;
```

# 协议打包



```
int encode_common (uint8_t * base,
                   size_t * idx,
                   const n2n_common_t * common) {

    uint16_t flags = 0;

    encode_uint8(base, idx, N2N_PKT_VERSION);
    encode_uint8(base, idx, common->ttl);

    flags  = common->pc & N2N_FLAGS_TYPE_MASK;
    flags |= common->flags & N2N_FLAGS_BITS_MASK;

    encode_uint16(base, idx, flags);
    encode_buf(base, idx, common->community, N2N_COMMUNITY_SIZE);

    return -1;
}
```

```
*(base + (*idx))     = ( v >> 8) & 0xff
*(base + (1 + *idx)) = ( v & 0xff );
*idx += 2;

return 2;
```



# Q&A