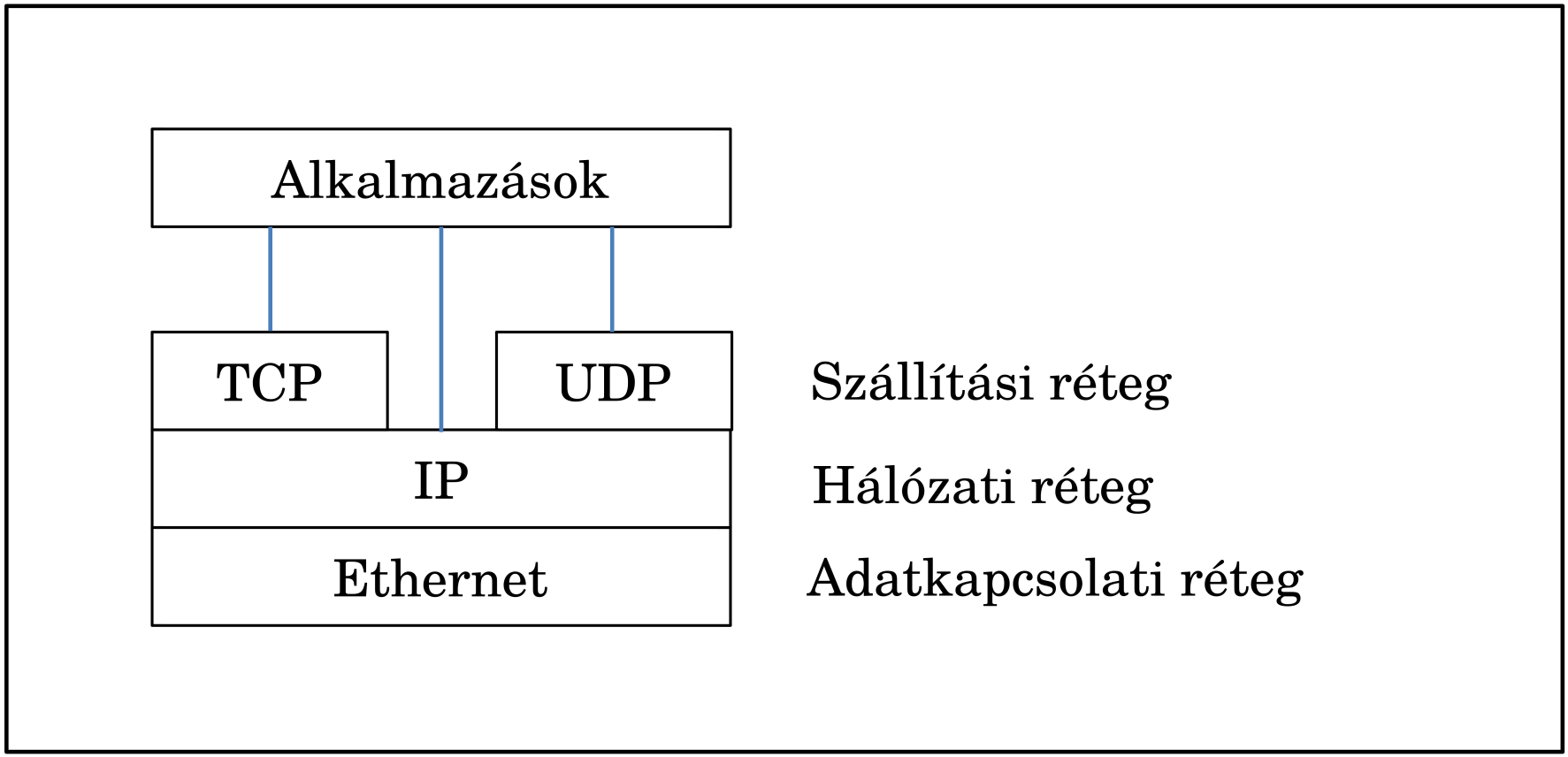
Linux Programozás

IP

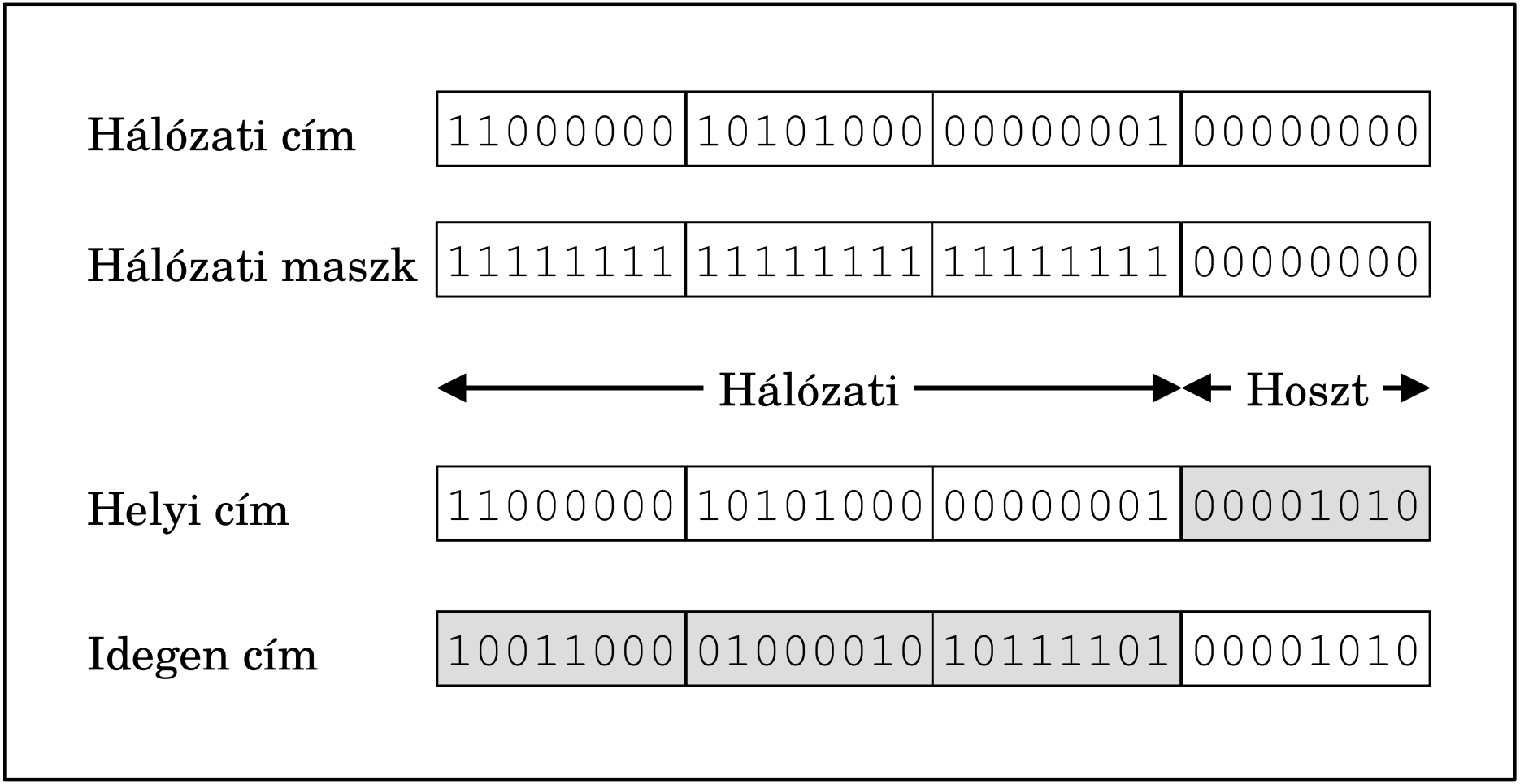
Az IP protokoll rétegződése



IPv4-es címzés

- 32 bit (4 bájt)
- Pontozott IP cím formátum: 152.66.188.11
- A 32 bites IP cím és a hálózati maszk meghatároz egy címtartományt:
  - Ha csak a maszkon kívül van eltérés: lokális cím
  - Ha a maszkolt részen is van eltérés: távoli cím
- A tartomány első és utolsó címe speciális:
  - Első: hálózati cím
  - Utolsó: broadcast cím

IPv4 hálózati maszk



IPv4 címosztályok

- Régen ez alapján ment a routing, de ma már nem.
- A 32 bites cím felosztása:
  - M bit: címosztály
  - N bit: hálózat címe
  - 32-M-N bit: hoszt címe

IPv4 címosztályok

Cím	Első cím	Utolsó cím	Azonosító	N
A osztály	1.0.0.0	127.255.255.255	0	7
B osztály	128.0.0.0	191.255.255.255	10	14
C osztály	192.0.0.0	223.255.255.255	110	21
D osztály	224.0.0.0	239.255.255.255	1110	Többes küldés
E osztály	240.0.0.0	247.255.255.255	11110	Fenntartva

IPv4 speciális címek

- Loopback címtartomány:
  - 127.0.0.0/8
- Privát címtartományok:
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16
- Multicast tartomány (többes küldés):
  - 224.0.0.0/4 (D osztály)

IPv6 címzés

- IPv4 címtartomány kifogyott
- 128 bites címek
- Továbbá:
  - automatikus cím konfiguráció
  - fejlettebb többesküldés
  - a routerek feladata egyszerűsödött
  - több útvonalválasztási opció
  - mobilitás
  - hitelesítés
  - adatbiztonság

IPv6 címzés

- Szöveges leírás:

```
8000:0000:0000:0000:0325:6A41:FEED:DEAD
```

- Rövid alak:

```
8000::325:6A41:FEED:DEAD
```

- Régi IP címek:

```
::152.66.188.11
::FFFF:9842:BC0B
```

- Az IPv4 címek az IPv6-ban:

```
0000:0000:0000:0000:0000:FFFF:XXXX:XXXX
::FFFF:XXXX:XXXX
```

Portok

- A virtuális csatornákat címzi az alkalmazások között.
- 2 bájt
- Külön TCP és UDP portok
- <1024: jól ismert portok – csak root
- 1024-49151: regisztrált portok
- 49152-65535: dinamikus és magánportok

Szolgáltatások portjai

Szolgáltatás neve	Port
ftp-data	20
ftp	21
ssh	22
telnet	23
smtp	25
http	80
portmap	111
https	443

HW függő különbségek

- Little endian és big endian gépek is beszélgetnek.
  - Bytestream esetén az alkalmazásra van bízva.
  - Címzésnél egységesnek kell lennie → big endian
- Hordozható kódhoz konverziós függvények:

Függvény	Leírás
ntohs	Egy 16 bites számot a hálózati byte-sorrendből a hoszt byte-sorrendbe vált át.
ntohl	Egy 32-bites számot a hálózati byte-sorrendből a hoszt byte-sorrend jébe vált át.
htons	Egy 16-bites számot a hoszt byte-sorrendjéből hálózati byte-sorrendbe vált át.
htonl	Egy 32-bites számot a gép byte-sorrendjéből hálózati byte-sorrendbe vált át.

IPv4 cím struktúra

- Header: `netinet/in.h`

```
struct sockaddr_in
{
  sa_family_t    sin_family;  /* Címcsalád = AF_INET */
  in_port_t      sin_port;    /* A port száma */
  struct in_addr sin_addr;    /* IPv4 cím */
  unsigned char  sin_zero[8]; /* struct sockaddr vége */
};

struct in_addr
{
  in_addr_t s_addr; /* előjel nélküli 32 bites szám */
}
```

IPv6 cím struktúra

- Header: `netinet/in.h`

```
struct sockaddr_in6
{
  sa_family_t     sin6_family;    /* Címcsalád = AF_INET6 */
  in_port_t       sin6_port;      /* A port száma */
  uint32_t        sin6_flowinfo;
  struct in6_addr sin6_addr;      /* IPv6 cím */
  uint32_t        sin6_scope_id;
};

struct in6_addr
{
  uint8_t s6_addr[16];
};
```

Szöveges ↔ bináris cím konverzió

- Csak IPv4 esetén (elavultak):
  - `inet_aton()`, `inet_ntoa()`
- Univerzális függvények (arpa/inet.h):
  - Szöveg → bináris:

```
int inet_pton(int af, const char *src, void *dst);
```

  - Bináris → szöveg:

```
const char *inet_ntop(int af, const void *src, char *dst,
socklen_t size);
```

  - af: címcsalád
  - src: forrás (szöveg illetve in_addr/in_addr6)
  - dst: cél (in_addr/in_addr6 illetve karakter tömb)
  - Megfelelő buffer méret:

```
#define INET_ADDRSTRLEN 16
#define INET6_ADDRSTRLEN 46
```

Lokális címek (szerverekhez)

- Szerver socket cím összeállításnál:
  - Valamelyik interfész IP címét adjuk meg.
  - Mindegyik interfészhez hozzárendeljük:

```
#define INADDR_ANY ((in_addr_t)
0x00000000)

#define IN6ADDR_ANY_INIT
{{{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }}}
```

- C-ben tömb értéket csak létrehozásnál adhatunk át egyszerűen:

```
const struct in6_addr in6addr_any =
IN6ADDR_ANY_INIT;
```

Loopback címek

- A loopback címek konstansai:

```
#define INADDR_LOOPBACK ((in_addr_t)
0x7f000001) /* 127.0.0.1 */

#define IN6ADDR_LOOPBACK_INIT
{{{ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 }}
}

const struct in6_addr in6addr_loopback
= IN6ADDR_LOOPBACK_INIT;
```

Név- és címfeloldás

- Az összerendelések helye:
  - Lokálisan: /etc/hosts
  - Szerveren: DNS, mDNS, LDAP, Yellow Pages, stb.
- Hoszt név, szolgáltatás név → IP cím és port
  - `getaddrinfo()`
- Socket cím → Hoszt név és szolgáltatás név
  - `getnameinfo()`

getaddrinfo()

```
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(const char *node, const char *service,
const struct addrinfo *hints, struct addrinfo **res);
void freeaddrinfo(struct addrinfo *res);
const char *gai_strerror(int errcode);
```

- node: Hoszt név
- service: Szolgáltatás
- hints: Kritériumok (NULL is lehet)
- res: Válasz (láncolt lista)
- Visszatérés: 0, vagy hibakód
- Hiba feldolgozás: `gai_strerror()`

getaddrinfo() kritériumok és válasz

```
struct addrinfo {
  int              ai_flags;
  int              ai_family;
  int              ai_socktype;
  int              ai_protocol;
  size_t           ai_addrlen;
  struct sockaddr *ai_addr;
  char            *ai_canonname;
  struct addrinfo *ai_next;
};
```

- ai_family: Címcsalád (AF_UNSPEC, AF_INET, AF_INET6)
- ai_socktype: Socket típusa (0, SOCK_STREAM, SOCK_DGRAM)
- ai_protocol: Protokoll index (0: nincs megkötés)
- ai_addr, ai_addrlen: cím és hoszza
- ai_canonname: Ha az ai_flags tartalmazza a AI_CANONNAME értéket. A lista első eleménél.
- ai_next: Következő elem.

ai_flags mező (kritériumok esetén)

Jelzőbit	Leírás
AI_ADDRCONFIG	A válaszlista csak akkor tartalmaz IPv4-es címet, ha a lokális gépnek legalább egy IPv4-es címe van, amely nem a loopbackcím. Illetve IPv6-os címek esetén hasonlóképpen.
AI_ALL	Csak az *AI_V4MAPPED* értékkel van jelentése. Értelmezését lásd ott.
AI_CANONNAME	Ha a *node* paraméter értéke nem *NULL*, akkor a válaszlista első elemének *ai_canonname* mezője tartalmazza a hoszt nevét.
AI_NUMERICHOST	Kikapcsolja a névfeloldást, és a *node* paraméter értéke csak numerikus címreprezentáció lehet. Ezzel megtakaríthatjuk a névfeloldás idejét.
AI_NUMERICSERV	Kikapcsolja a szolgáltatás névfeloldását. Szolgáltatásnak csak számot adhatunk meg (szövegesen).
AI_PASSIVE	A címstruktúra egy szerversocket címhez kötéséhez használható lokális címet tartalmaz, ha ez az opció be van állítva, és a *node* paraméter *NULL*. Vagyis a cím *INADDR_ANY* vagy *IN6ADDR_ANY_INIT* lesz.
AI_V4MAPPED	Ha ez az opció be van kapcsolva, és mellette a kritérium *ai_family* mező értéke *AF_INET6*, és a függvény ennek ellenére csak IPv4-es címet talál, akkor az IPv4-es címet IPv6-os formátumban adja vissza. Ha az *AI_ALL* opcióval együtt alkalmazzuk, akkor IPv6-os és IPv4-es címeket is visszaad, de utóbbiakat IPv6-os formátumba alakítva.

getnameinfo()

```
#include <sys/socket.h>
#include <netdb.h>
int getnameinfo(const struct sockaddr *sa,
socklen_t salen, char *host, size_t hostlen, char
*serv, size_t servlen, int flags);
```

- sa, salen: A socket cím (sockaddr_in, sockaddr_in6)
- host, hostlen: Lefoglalt tömb a névnek.

```
#define NI_MAXHOST      1025
```

- serv, servlen: Lefoglalt tömb a szolgáltatásnak.

```
#define NI_MAXSERV      32
```

- flags: beállítások
- Visszatérés: 0, vagy hibakód (lásd: `gai_strerror()`)

getnameinfo() flags

Jelzőbit	Leírás
NI_DGRAM	Egyes portoknál TCP és UDP protokollok esetében más-más szolgáltatás fut. Alapértelmezetten a TCP-szolgáltatás nevét kapjuk vissza. Ezzel a paraméterrel az UDP-szolgáltatást kapjuk meg.
NI_NAMEREQD	Alapértelmezetten, ha a hosztnevet nem találja a függvény, akkor szöveges IP-címet ad vissza. Ha megadjuk ezt az opciót, akkor ebben az esetben hibával tér vissza a függvény.
NI_NOFQDN	Alapértelmezetten teljes hosztnevet (Fully Qualified Domain Name) kapunk vissza. Ezzel az opcióval csak a rövid hosztnevet.
NI_NUMERICHOST	Ha megadjuk, akkor nem történik névfeloldás, hanem csak szöveges IP-címet kapunk vissza. Ez történik akkor is, ha nem sikerül a hosztnevet kideríteni.
NI_NUMERICSERV	Numerikusan, egy decimális számot tartalmazó szövegként kapjuk vissza a portot.

Összeköttetés alapú kommunikáció

- Korábban láttuk.
- Csak IP cím struktúrákat kell használnunk.
- Kommunikáció:
  - read(), write()
  - send(), recv()

send()

```
#include <sys/socket.h>

ssize_t send(int sockfd, const void
*buf, size_t len, int flags);
```

- Mint a write() + flags

send() flags

Jelzőbit	Leírás
MSG_DONTROUTE	A csomag nem mehet keresztül az útválasztókon, csak közvetlenül ugyanazon a hálózaton lévő gép kaphatja meg.
MSG_DONTWAIT	Engedélyezi a nem blokkoló I/O-t. *EAGAIN* hibával tér vissza, ha várakozni kellett volna a kiírásra, mert tele van a buffer.
MSG_MORE	További adatot szeretnénk még kiküldeni. Ezért nem küldi el a csomagot addig, amíg nem kap egy *send()* hívást *MSG_MORE* opció nélkül.
MSG_NOSIGNAL	Adatfolyam-alapú kapcsolat esetén a program nem kap *SIGPIPE* jelzést, amikor a kapcsolat megszakad. Ez azonban az *EPIPE* hibajelzést nem érinti.
MSG_OOB	Soron kívüli sürgős adatcsomagot (out-of-band data) küld. Általában jelzések hatására használják.

recv()

```
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf,
size_t len, int flags);
```

- Mint a read() + flags

recv() flags

Jelzőbit	Leírás
MSG_DONTWAIT	Engedélyezi a nem blokkoló I/O-t. *EAGAIN* hibával tér vissza, ha várakozni kellett volna az olvasásra, mert nem érkezett adat.
MSG_OOB	Soron kívüli adat fogadása.
MSG_PEEK	Az adat beolvasása történik meg anélkül, hogy a beolvasott adatot eltávolítaná a bufferből. A következő *recv()* hívás ugyanazt az adatot még egyszer kiolvassa.
MSG_WAITALL	Addig nem tér vissza, amíg a megadott buffer meg nem telik, vagy egyéb rendhagyó dolog nem történik, például jelzés érkezik.

Fájl küldés

```
#include <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd,
off_t *offset, size_t count);
```

- in_fd: Állomány
- out_fd: Socket
- offset: Eltolás mutatója. Innen kezdi olvasni. Visszatérésnél módosítja, hogy folytathassuk.
- count: Bájtok száma
- Visszatérés: Elküldött bájtok száma

TCP szerver struktúrák

- Párhuzamosan kell kapcsolatokat kezelnünk és újakat fogadnunk
- Módszerek:
  - Kapcsolatonként egy szál/folyamat
  - Thread/process pool (előre elindított szálak/folyamatok)
  - Egy szálon: poll(), select()

TCP kliens struktúrák

- Egyszerre kell a kapcsolatot és a felhasználót kezelni.
- Szöveges módban:
  - select(), poll(), epoll()
  - szálak
- Grafikus módban:
  - Aszinkron socket kezelés (grafikus fejlesztői könyvtárak tartalmazzák)

Összeköttetés nélküli kommunikáció

- Mint korában láttuk, csak IP címstruktúrákkal.
- UDP:
  - Nem garantált, hogy a csomag célbaér
  - Nem garantált a csomagok sorrendje
  - Max. 64 kB
  - Gyorsabb, mint a TCP
  - Többes küldést tesz lehetővé.

Többes küldés címei (IPv4)

- Valamelyik D osztályú (224.0.0.0–239.255.255.255) címet kell használnunk csoport címként.
- Előre definiált csoportok:
  - http://www.iana.org/assignments/multicast-addresses/multicast-addresses.txt

IP cím	Hosztok
224.0.0.0	báziscím, foglalt
224.0.0.1	az egy LAN-on lévő hosztok
224.0.0.2	az egy LAN-on lévő útválasztók

Többes küldés címei (IPv6)

- Az ff00::/8 tartomány van fenntartva.
- Számos előre definiált csoport van:
  - http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml

IP cím	Hosztok
ff02::1	Az egy LAN-on lévő hosztok
ff02::2	Az egy LAN-on lévő útválasztók

Többes küldés

- Előre definiált csoportok esetén:
  - Az adott címre küldjük a csomagokat.
- Saját csoport esetén:
  - Regisztrálni kell az alkalmazásokban a gépeket az adott csoportba, hogy megkapjuk a csomagokat.
  - Az adott címre kell küldenünk csomagokat.

Socket beállítások

```
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int
optname, void *optval, socklen_t *optlen);

int setsockopt(int sockfd, int level, int
optname, const void *optval, socklen_t
optlen);
```

- sockfd: Socket
- level: Protokoll szint
- optname: opció azonosítója
- optval, optlen: opció értéke

Protokoll szintek

Érték	Szint	„man” oldal
SOL_SOCKET	socket és egyben Unix	socket(7), unix(7)
IPPROTO_IP	IPv4	ip(7)
IPPROTO_IP6	IPv6	ipv6(7)
IPPROTO_TCP	TCP	tcp(7)
IPPROTO_UDP	UDP	udp(7)

Hasznos socket opciók

- SO_KEEPALIVE
  - socket szint
  - Kapcsolat életben tartó csomagokat küld.
- SO_REUSEADDR
  - socket szint
  - Cím újra használható, ha nincs aktív socket, ami használja.
- TCP_CORK, UDP_CORK
  - TCP illetve UDP szint
  - Bufferben gyűjtögeti az adatokat és egyszerre küldi ki.

Többes küldés opciói (IPv4)

Opció neve	Leírás
IP_MULTICAST_LOOP	Ezzel tilthatjuk/engedélyezhetjük azt, hogy az elküldött multicast üzeneteket mi is megkapjuk.
IP_MULTICAST_TTL	Ezzel állíthatjuk a time-to-live (TTL) mezőt.
IP_MULTICAST_IF	Itt adhatjuk meg, hogy melyik hálózati interfészről küldjük a csomagokat. A rendszer-adminisztrátor alapértelmezésben megad egyet, de felülírhatjuk.
IP_ADD_MEMBERSHIP	Csatlakozás egy csoporthoz.
IP_DROP_MEMBERSHIP	Kiválás egy csoportból.

Többes küldés opciói (IPv6)

Opció neve	Leírás
IPV6_MULTICAST_LOOP	Ezzel tilthatjuk/engedélyezhetjük azt, hogy az elküldött multicast üzeneteket mi is megkapjuk.
IPV6_MULTICAST_HOPS	Ezzel állíthatjuk a hop limit mezőt.
IPV6_MULTICAST_IF	Itt adhatjuk meg, hogy melyik hálózati interfészről küldjük a csomagokat. A rendszer-adminisztrátor alapértelmezésben megad egyet, de felülírhatjuk.
IPV6_JOIN_GROUP	Csatlakozás egy csoporthoz.
IPV6_LEAVE_GROUP	Kiválás egy csoportból.

Csoport megadás struktúrái

```
struct ip_mreq
{
  struct in_addr imr_multiaddr;  /* A csoport IP cime
*/
  struct in_addr imr_interface;  /* A lokalis IP cim */
};


struct ipv6_mreq
{
  struct in6_addr ipv6mr_multiaddr;/* A csoport IP cime
*/
  unsigned int ipv6mr_interface;   /* A lokalis
interfész */
};
```

Segédprogramok

- netstat:
  - Beállítások ellenőrzése
- tcpdump:
  - Hálózati forgalom monitorozás
- wireshark:
  - Hálózati forgalom monitorozás
  - Grafikus felület