HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

HÁLÓZATOK ALAPJAI ÉS ÜZEMELTETÉSE

Socketek, szállítási réteg

2023. március 13.

Mészáros András

BME Hálózati Rendszerek és Szolgáltatások Tanszék

meszarosa@hit.bme.hu

MŰEGYETEM 1782

AZ INTERNET PROTOKOLLSZERKEZETE

A TCP/IP protocol stack rétegei:

- **alkalmazási (application)**
  - a hálózati alkalmazásokat támogatja
  - FTP, SMTP, HTTP
- **szállítási (transport)**
  - Adatátvitel processztől processzig
  - TCP, UDP
- **hálózati (network)**
  - adatok (csomagok) mozgatása a forrás és nyelő hosztok között
  - IP, útvonalválasztó protokollok
- **adatkapcsolati (link)**
  - adatok (csomagok) továbbítása a szomszédos hálózatelemek között
  - PPP, Ethernet
- **fizikiai (physical)**
  - bitek továbbítása a szomszédos csomópontok közti összeköttetéseken

MA MIRŐL LESZ SZÓ?

1. Socketek programozása
2. A szállítási réteg szolgáltatásai
3. Nyalábolás és nyalábbontás
4. Összeköttetés nélküli szállítás: UDP

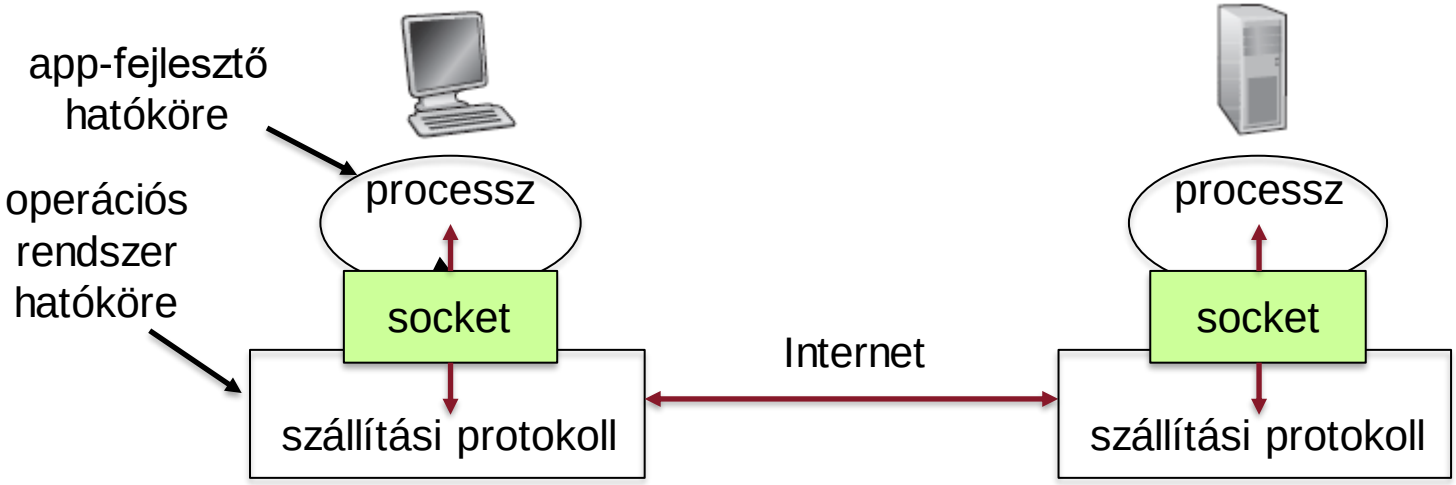
A fóliák elkészítéséhez felhasználtuk Jim Kurose és Keith Ross „Számítógép hálózatok működése” című könyvéhez készült fóliákat.

SOCKETEK HASZNÁLATA

- **Socket** (ismétlés)
  - A hálózati applikáció processze nyitja meg
  - A hoszt operációs rendszere kezeli
  - Egyfajta ajtószerű interfész a processz számára
  - Üzenetek **küldésére** és **fogadására** is képes
  - Szállítási réteg szintű kapcsolatot teremt távoli processzek között
- **Socket API**
  - 1981-ben vezetik be a BSD4.1 UNIX-ban
  - Az applikáció felelőssége
    - Létrehozás
    - Használat
    - Megszüntetés
  - Kliens-szerver kommunikációra
  - Alapvető típusok
    - Megbízható, **kapcsolat-orientált**, **adatfolyam (stream)** átvitelére
    - Megbízhatatlan, **adatcsomag (datagram)** orientált

KAPCSOLAT-ORIENTÁLT SOCKET

- Megbízható szállítás TCP-vel (ismétlés)
  - Kapcsolatorientált
  - Nyugtázott (minden megérkezik)
  - Sorrendhelyes
- Átvitt adatok
  - Összetartozó adatrészek, szegmensek
  - Nem az applikáció feladata az újraküldetés vagy átsorrendezés



TCP ALAPÚ SOCKET

- Szerver
  - **Folyamatosan futó** processz
  - Várakozik a kliensre
  - Hallgatózik egy adott porton: Egy nyitott **szerver-socketet** tart fenn a kliensek fogadására
- Kliens
  - **Kezdeményezi a csatlakozást** a szerverhez
  - Egy **kliens-socketet** nyit
  - A szerver-socket adatait használja: IP cím, portszám
  - A kliens-sockethez az operációs rendszer rendel portszámot általában
- Adatátvitel
  - A kliens-socket felől kapcsolat épül fel a szerver-socket felé
  - A **szerver fogadja a klienst**, de…
  - Az egyes kliensekhez újabb és újabb **kapcsolat-socketeket** nyit meg
  - A kliens oldali IP cím és portszám alapján tudja a szerver a megfelelő kapcsolat-sockethez irányítani az adatokat
  - Így kommunikálhat **több kliens** is a szervernek ugyanazt a portját használva

TCP ALAPÚ SOCKET-KOMMUNIKÁCIÓ LÉPÉSEI

szerver
IP cím: `1.1.1.1`

`socket.SOCK_STREAM`

Szerver-socket nyitása
és hozzákötése a **22222** porthoz
`servsock = socket.socket()`
`servsock.bind()`

kliens

Hallgatózás a **22222** porton
`servsock.listen()`

TCP kapcsolat felépítés

Kliens-socket nyitása, kapcsolódás
az **1.1.1.1** szerver **22222** portjához
`clientsock = socket.socket()`
`clientsock.connect()`

Bejövő kapcsolat fogadása:
új kapcsolat-socket nyitása
`connsock = servsock.accept()`

Kérés küldése
`clientsock.sendall()`

Kérés fogadása
`connsock.recv()`

Válasz küldése
`connsock.sendall()`

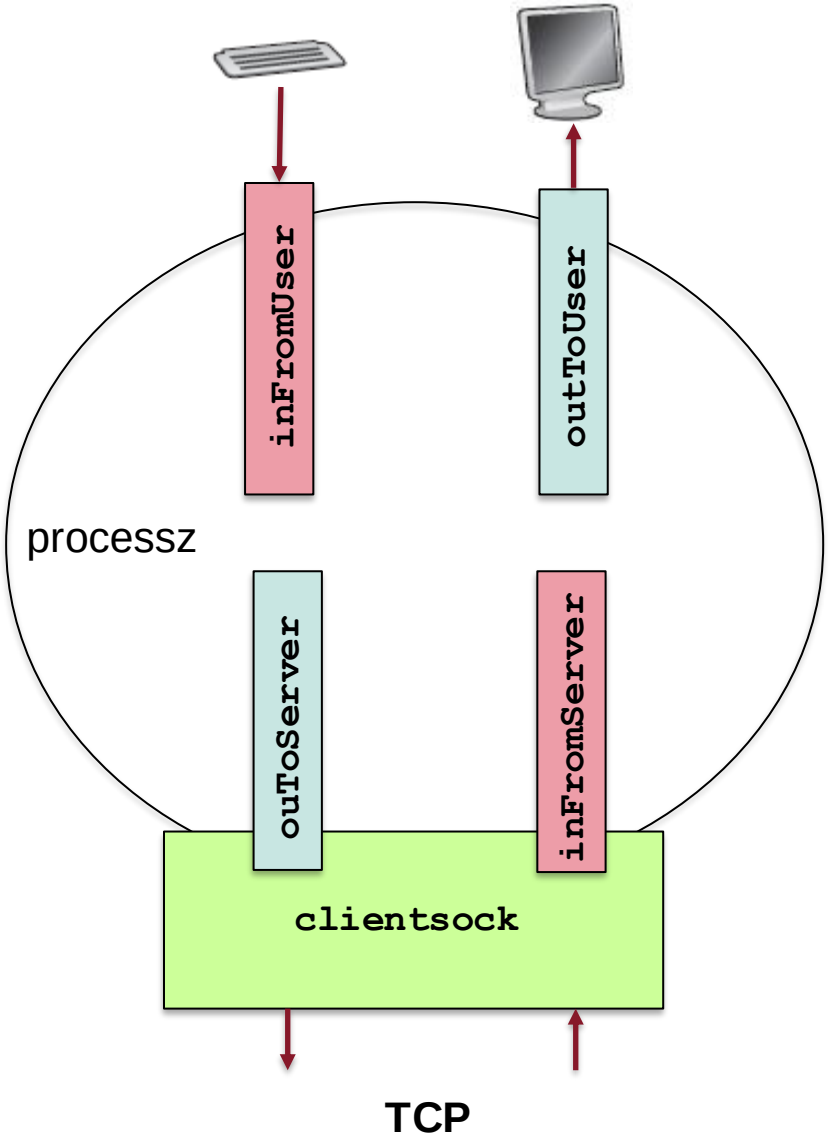
Válasz fogadása
`clientsock.recv()`

Lezárás
`connsock.close()`

Lezárás
`clientsock.close()`

ADATFOLYAMOK

- Egy folyam (stream) egy processzből kijövő, vagy abba bemenő **bájtsorozat**
- Egy bemeneti (input) stream egy **forráshoz** van kötve, pl. billentyűzet, vagy socket
- Egy kimeneti (output) stream egy **nyelőhöz** van kötve, pl. képernyő, vagy socket



TCP SOCKET – PYTHON PÉLDA

- Egyszerű hálózati alkalmazás
- Kliens
  - Beolvas egy sort és elküldi a szervernek
  - Kiírja amit a szervertől kapott
- Szerver
  - Kiolvassa a socketből amit a klienstől kapott
  - Konvertálja csupa nagybetűre
  - Az eredményt visszaküldi a kliensnek

SZERVER OLDALI SZKRIPT – TCP

- Szerver-socket létrehozása
- Hallgatózás indítása
- Kliens fogadása: új kapcsolat-socket
- Szöveg olvasása a stream-ből
- Konverzió és visszaküldés
- Kapcsolat-socket lezárás

```
import socket

def serverMainTCP():
    servsock = socket.socket(
        socket.AF_INET, socket.SOCK_STREAM)

    servsock.bind(('1.1.1.1', 22222))
    servsock.listen(1)
    connsock,addr = servsock.accept()

    while True:
        text = connsock.recv(512).decode()
        if not text: break

        connsock.sendall(
            text.upper().encode())

    connsock.close()

serverMainTCP()
```

KLIENS OLDALI SZKRIPT – TCP

- Kliens-socket létrehozása
- Kapcsolat a szerverrel
- Szöveg beolvasása a billentyűzetről
- Küldés a szervernek
- Válasz kiolvasása
- Kiírás a képernyőre
- Kapcsolat-socket lezárás

```
import socket
import fileinput

def clientMain():
    clientsock = socket.socket(
        socket.AF_INET, socket.SOCK_STREAM)
#   clientsock.bind(('4.4.4.4', 33333))
    clientsock.connect(('1.1.1.1', 22222))

    for text in fileinput.input():
        clientsock.sendall(text.encode())

    retext=clientsock.recv(512).decode()

    print(retext)

    clientsock.close()

clientMain()
```

UDP ALAPÚ SOCKET

- UDP
  - Nincs felépített kapcsolat a kliens és a szerver között
  - Nincs nyugtázás
  - Nincs sorrendezés
- Datagramok küldése (kliens)
  - A socket a processztől kapott datagramhoz hozzáteszi a portszámot és az IP címet
- Datagramok fogadása (szerver)
  - A kliens azonosítása csak közvetlenül a küldési IP cím és portszám alapján történhet
- Következmény
  - Egy processz egy socketen keresztül több kliens-socket felől kaphat datagramokat
  - A különböző kliensek datagramjai keverve érkeznek a processzhez

UDP ALAPÚ SOCKET-KOMMUNIKÁCIÓ LÉPÉSEI

szerver
IP cím: `1.1.1.1`

`socket.SOCK_DGRAM`

Datagram-socket nyitása
és hozzákötése a **22222** porthoz
`dsock_s = socket.socket()`
`dsock_s.bind()`

kliens
IP cím: `4.4.4.4`

Datagram-socket nyitása
és hozzákötése a **33333** porthoz
`dsock_c = socket.socket()`
`dsock_c.bind()`

Kérés küldése `1.1.1.1:22222` felé
`dsock_c.sendto()`

Kérés fogadása
`dsock_s.recvfrom()`

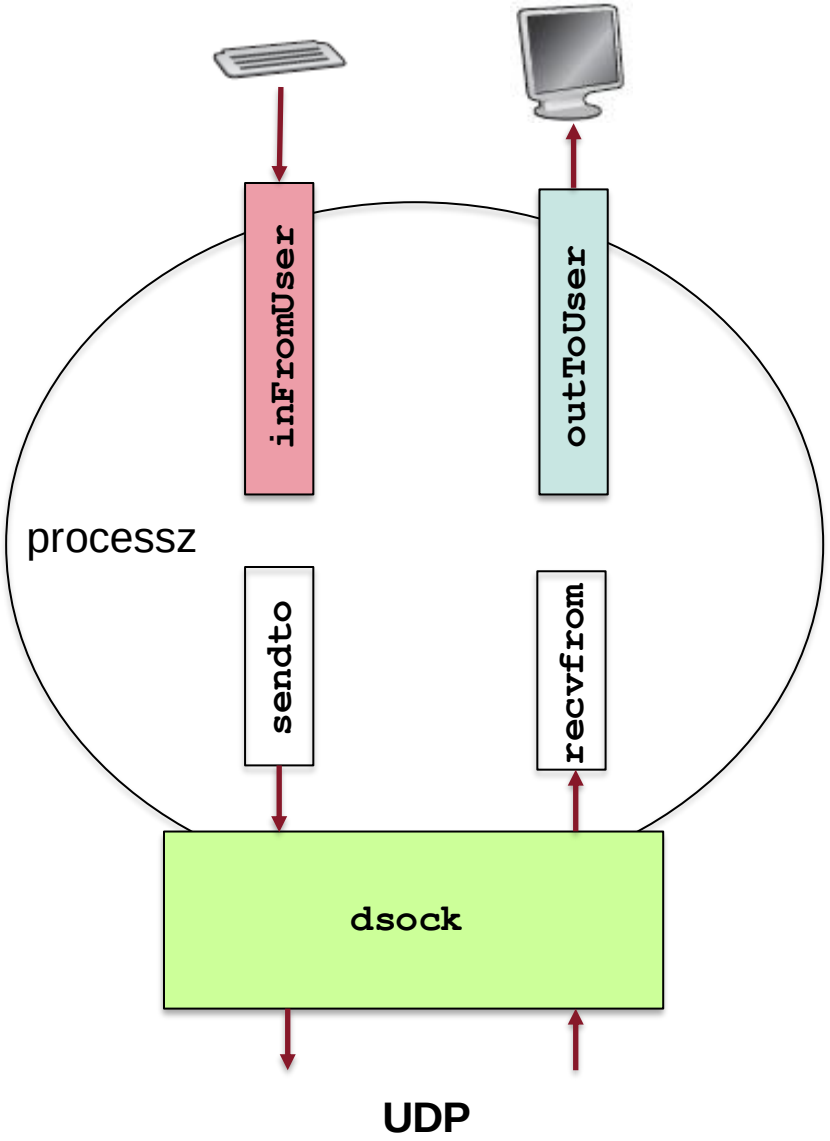
Válasz küldése `4.4.4.4:3333` felé
`dsock_s.sendto()`

Válasz fogadása
`dsock_c.recv()`

Lezárás
`dsock_c.close()`

DATAGRAMOK

- A socketbe nem stream jelleggel megy az adat, hanem egyesével címezve
- A socketből nem stream jelleggel jön az adat, hanem egyesével
- A példa ugyanaz



SZERVER OLDALI SZKRIPT – UDP

- Datagram-socket létrehozása
- Szöveg és küldő kiolvasása a datagramból
- Konverzió és visszaküldés a küldőnek

```
import socket

def serverMainUDP():
    dsock_s = socket.socket(
        socket.AF_INET, socket.SOCK_DGRAM)
    dsock_s.bind(('1.1.1.1', 22222))

    while True:
        text, client = dsock_s.recvfrom(512)

        dsock_s.sendto(
            text.decode().upper().encode(),
            client)

serverMainUDP()
```

KLIENS OLDALI SZKRIPT – UDP

- Datagram-socket létrehozása
- Szöveg beolvasása a billentyűzetről
- Küldés a szervernek
- Válasz kiolvasása
- Kiírás a képernyőre
- Datagram-socket lezárás

```
import socket
import fileinput

def clientMainUDP():
    dsock_c = socket.socket(
        socket.AF_INET, socket.SOCK_DGRAM)
    dsock_c.bind(('4.4.4.4', 33333))

    for text in fileinput.input():
        dsock_c.sendto(text.encode(),
            ('1.1.1.1', 22222))

        retext =
            dsock_c.recv(512).decode()

        print(retext)

    dsock_c.close()

clientMainUDP()
```

ALKALMAZÁS-FEJLESZTÉS

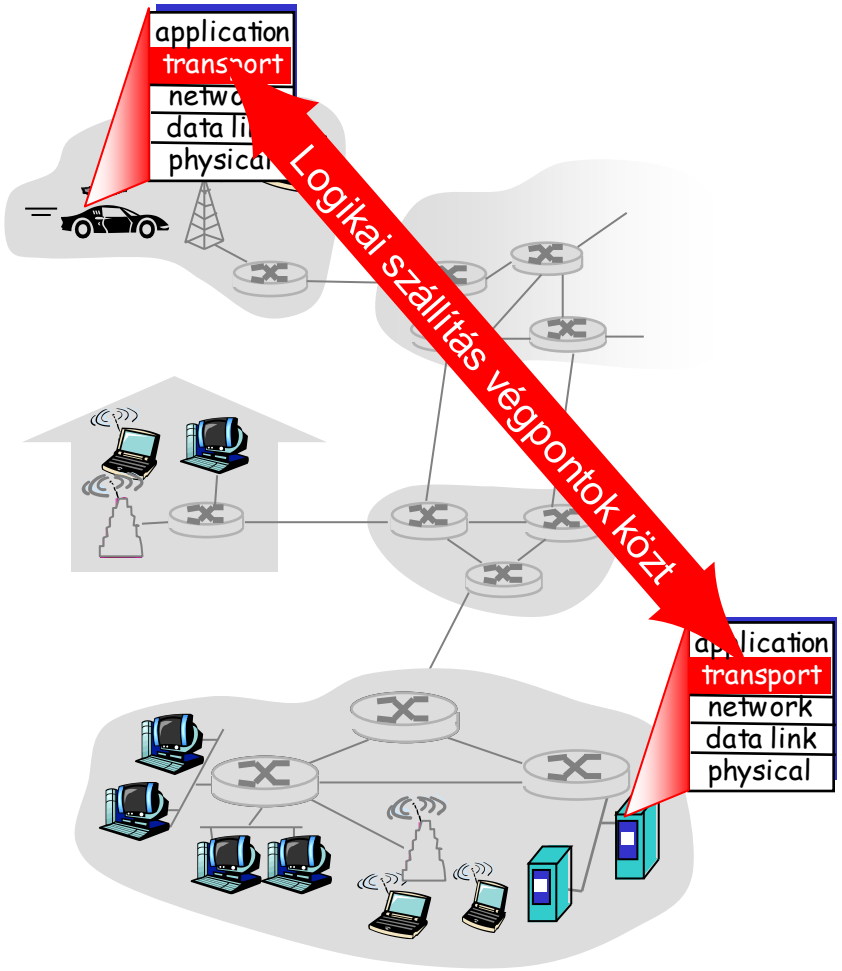
- TCP és UDP socketek
- Akkor az alkalmazás-fejlesztőnek kell minden magasabb szintű protokollt implementálni?
  - Nyilván nem.
  - Internet protokollok implementációi szabványos könyvtárakban, pl:
    - http, webbrowser
    - smtplib, smtpd, imaplib poplib
    - ftplib
    - ssl (biztonság)
- A python beépített modulokkal támogat
- Csak python szkriptekben írhatunk alkalmazást?
  - Nyilván nem.
  - Más nyelvek is támogatják a socketeket
    - Java, C++, C#, Ruby, stb.
  - És sok közülük a protokollokat is

MA MIRŐL LESZ SZÓ?

1. Socketek programozása
2. A szállítási réteg szolgáltatásai
3. Nyalábolás és nyalábbontás
4. Összeköttetés nélküli szállítás: UDP

SZÁLLÍTÁSI SZOLGÁLTATÁSOK ÉS PROTOKOLLOK

- A különböző hosztokon futó alkalmazások processzei közti **logikai kommunikációt** biztosítják
- A szállítási protokollok a végponti rendszereken, hosztokon futnak
  - Küldő oldal: az alkalmazásprotokoll üzeneteit **szegmensekre darabolva** adja át a hálózati rétegnek
  - Fogadó oldal: összeállítja az eredeti üzenetet a szegmensekből, és átadja az alkalmazási rétegnek
- Szállítási protokollok
  - Internet: TCP és UDP (és QUIC)



SZÁLLÍTÁSI ÉS HÁLÓZATI RÉTEG

- Szállítási réteg: logikai kommunikáció a (hosztokon futó) **processzek között**
- Hálózati réteg: logikai kommunikáció a **hosztok között**
- Minden réteg az alatta lévő szolgáltatásait használja:
  - Alkalmazási réteg a szállítási rétegét
  - Szállítási réteg a hálózati rétegét

TRANSZPORT SZOLGÁLTATÁSOK AZ INTERNETEN (ISMÉTLÉS)

TCP szolgáltatás

- **Kapcsolatorientált**: összeköttetés felépítése (set up) szükséges a kliens és szerver processz között
- **Megbízható szállítás**: a küldőtől minden megérkezik sorrendezve fogadóhoz
- **Forgalomszabályozás**: a küldő nem terheli túl a fogadót
- **Torlódáskezelés**: a küldő lefojtása ha hálózat túlterhelt
- Nem biztosít sem időzítést, sem minimális garantált sávszélességet

UDP szolgáltatás

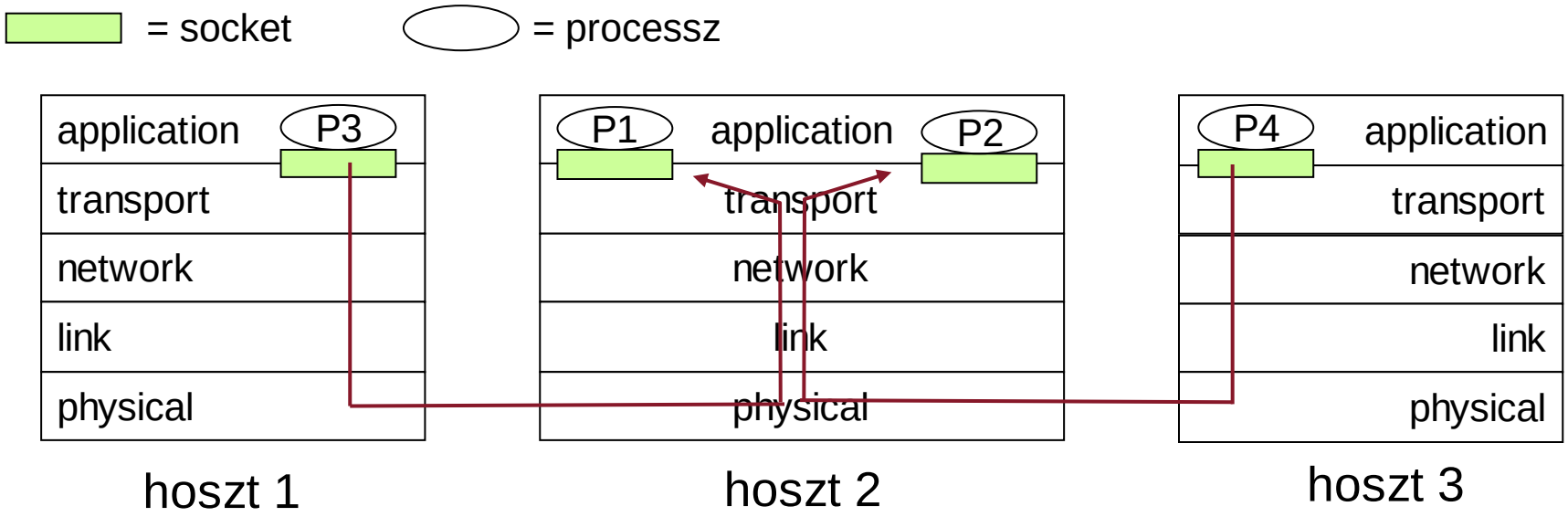
- **Megbízhatatlan szállítás**: nem biztos, hogy minden és hogy helyes sorrendben érkezik meg
- Nem biztosít összeköttetés felépítést, forgalomszabályozást, torlódáskezelést, időzítést, vagy garantált sávszélességet

MA MIRŐL LESZ SZÓ?

1. Socketek programozása
2. A szállítási réteg szolgáltatásai
3. Nyalábolás és nyalábbontás
4. Összeköttetés nélküli szállítás: UDP

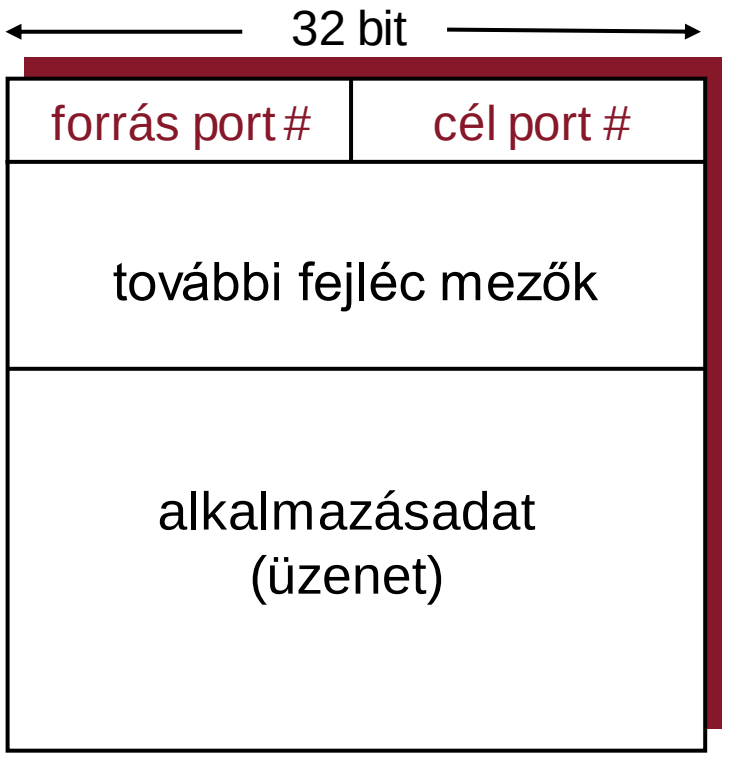
NYALÁBOLÁS/NYALÁBBONTÁS

- Nyalábolás (**multiplexálás**) a küldő hosztnál
  - Különböző socketek adatainak összegyűjtése,
  - Fejlécezése (a demultiplexáláshoz)
- Nyalábbontás (**demultiplexálás**) a fogadó hosztnál
  - A vett szegmensek továbbítása a megfelelő socketnek



A DEMULTIPLEXÁLÁS MŰKÖDÉSE

- A hoszt megkapja az **IP datagramot (csomag, packet)**
  - Minden datagramnak van forrás és cél IP-címe
  - Minden egyes datagram pontosan egy darab szállítási rétegbeli **szegmenst** hordoz
  - Minden szegmensnek van forrás és cél portszáma
- A hoszt a szegmenst az IP-cím és a portszám alapján irányítja a megfelelő sockethez



a TCP/UDP szegmens szerkezete

KAPCSOLATMENTES DEMULTIPLEXÁLÁS

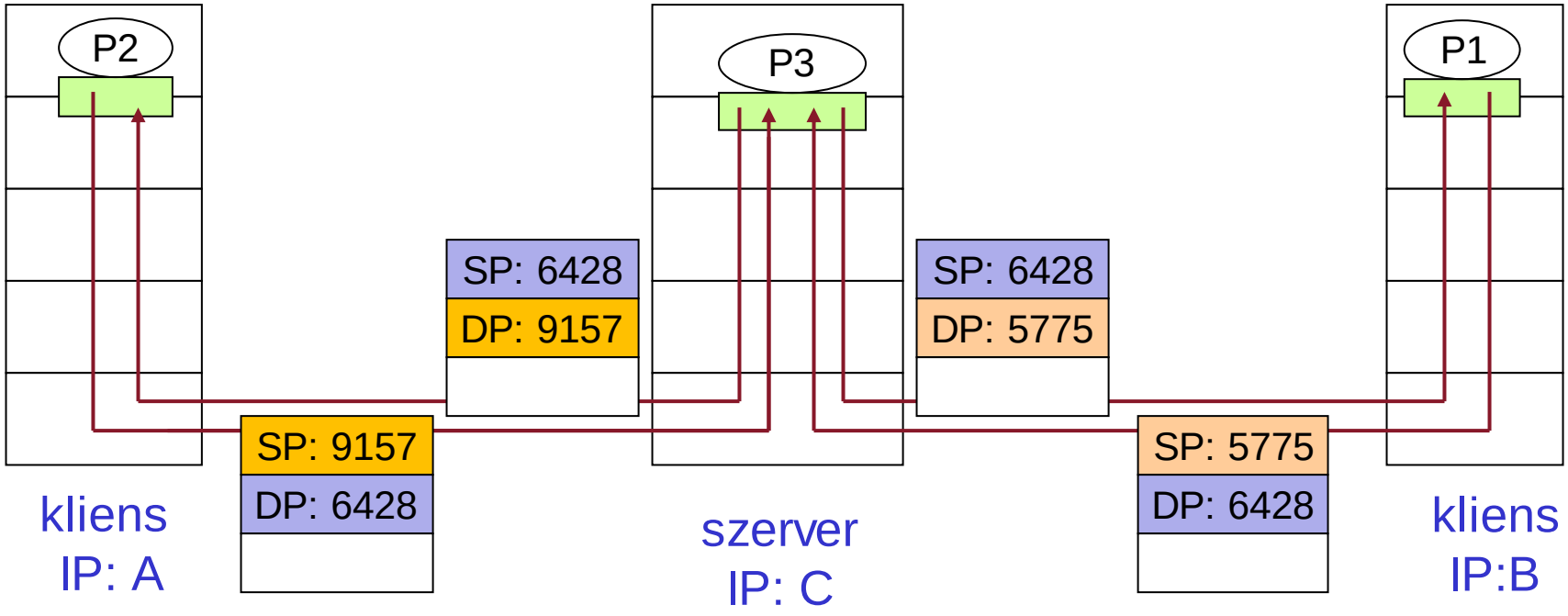
- Socket létrehozása és portszám-hozzárendelés

```
dsock_s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
dsock_s.bind(('1.1.1.1', 22222))
```

- Az UDP socketet egy értékpár azonosítja
  - (cél IP cím, cél portszám)
- Amikor a hoszt kap egy UDP szegmenst:
  - megvizsgálja a szegmensben szereplő cél portszámot
  - az ennek megfelelő portszámú socketnek adja át a szegmenst
- A különböző forrás IP-című és/vagy forrás portszámot tartalmazó szegmensek a cél portszámuk alapján ugyanahhoz a sockethez kerülhetnek

UDP DEMULTIPLEXÁLÁS ILLUSZTRÁCIÓ

- A kérésben lévő forrás portszám (SP) lesz majd a válasz célportja (DP)
  - A szervernél vett SP, DP fordítva lesz a válaszban

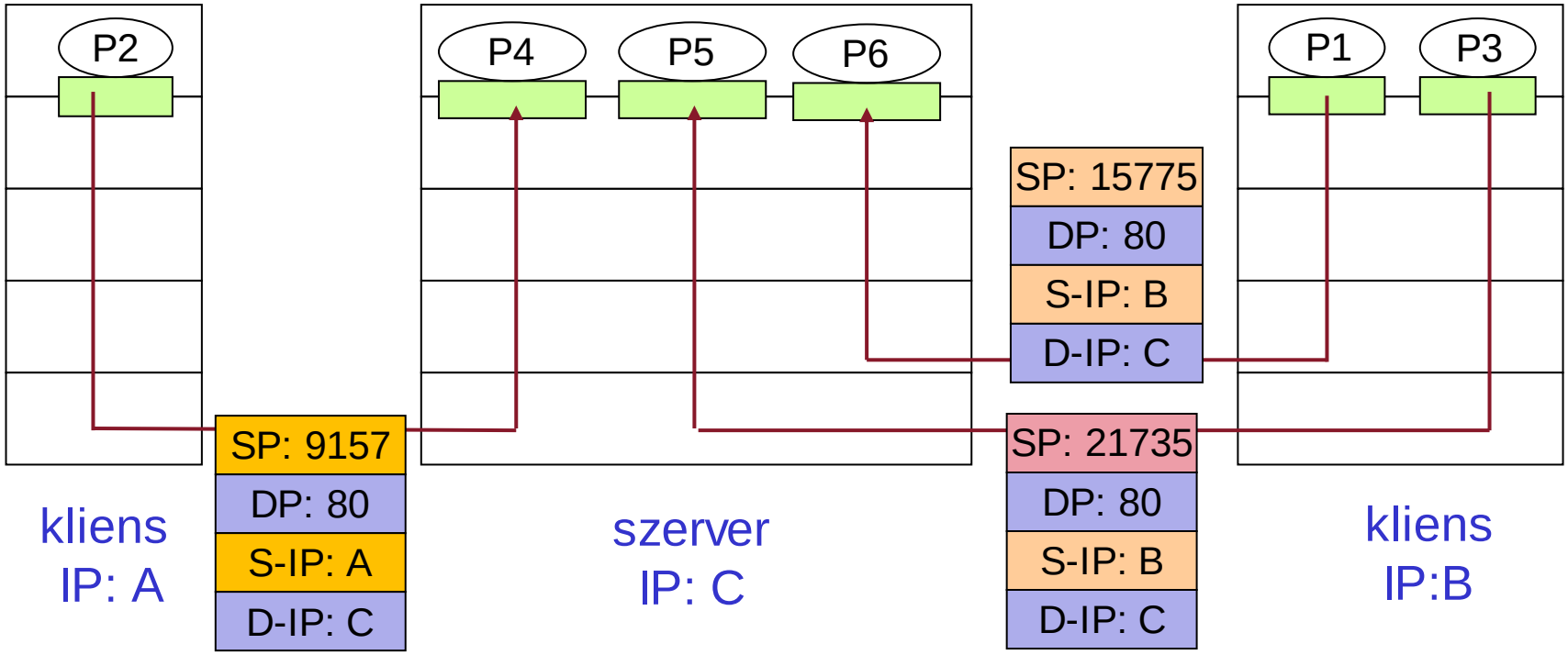


KAPCSOLAT ALAPÚ DEMULTIPLEXÁLÁS

- A TCP kapcsolat-socketet egy értéknégyes azonosítja:
  - Forrás IP cím
  - Forrás portszám
  - Cél IP cím
  - Cél portszám
- A vevő oldali hoszt mind a négy értéket felhasználja a szegmens megfelelő sockethez továbbítására
- Egy szerver szerepű host egyszerre számos TCP socketet kezelhet:
  - Minden egyes socketet saját értéknégyese azonosít
- Egy web szerver más-más socketen keresztül kezeli az egyes kapcsolódott klienseket
  - A nem perzisztens HTTP külön socketen keresztül kezel minden egyes kérést

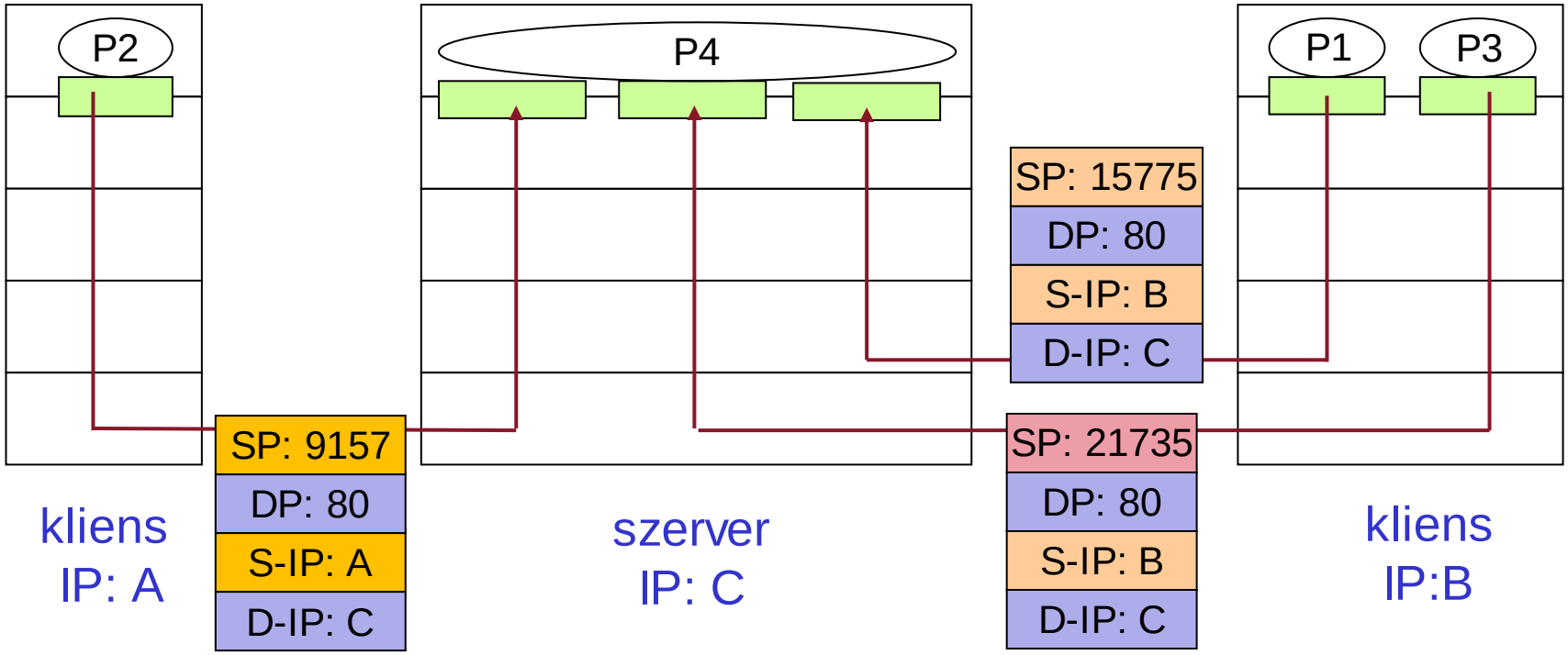
TCP DEMULTIPLEXÁLÁS ILLUSZTRÁCIÓ – EGY SOCKET

- Külön processz minden sockethez



TCP DEMULTIPLEXÁLÁS ILLUSZTRÁCIÓ – TÖBB SOCKET

- Egy processzhez több socket is tartozik



MA MIRŐL LESZ SZÓ?

1. Socketek programozása
2. A szállítási réteg szolgáltatásai
3. Nyalábolás és nyalábbontás
4. Összeköttetés nélküli szállítás: UDP

USER DATAGRAM PROTOCOL

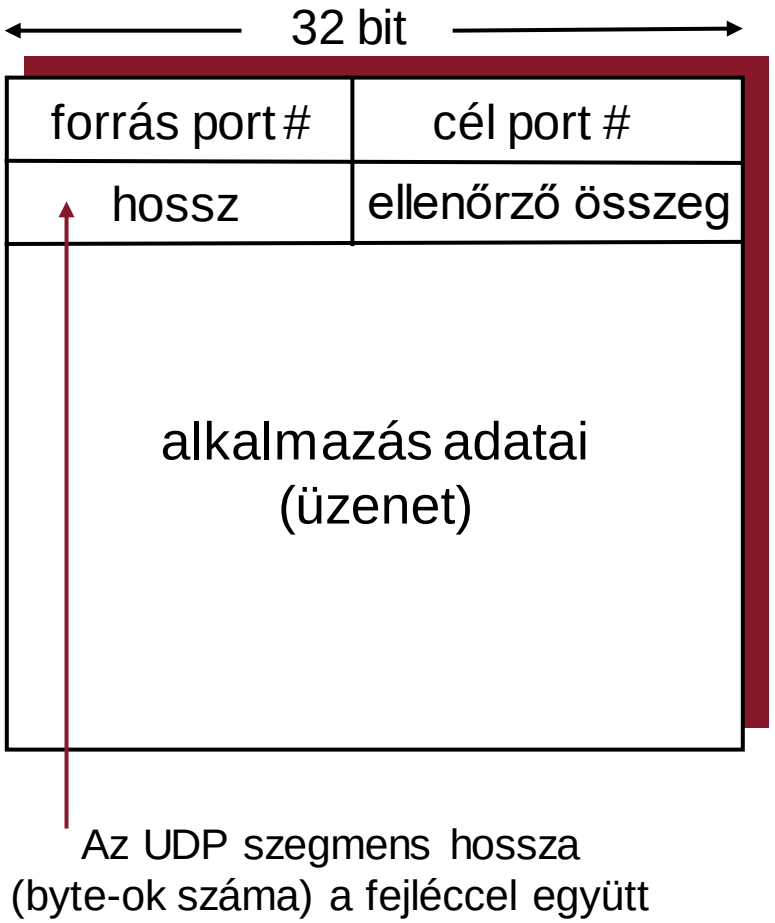
- “Lecsupaszított” Internet transzport protokoll
- “Best effort” szolgáltatás, a szegmensek
  - Elveszhetnek
  - Összekeveredhetnek
- Összeköttetés nélküli
  - Nincs „kézfogás” az UDP küldő és fogadó oldala között
  - Minden egyes UDP szegmens kezelése független a többitől

- Miért kell az UDP (is)?
  - Nem szükséges összeköttetés felépítés: **nincs kezdeti késleltetés**
  - **Egyszerű**: nincs kapcsolatállapot-kezelés sem a küldő, sem a fogadó oldalon
  - **Kis méretű** szegmensfejléc
  - Nincs kézfogás és torlódáskezelés: az UDP a lehető **leggyorsabban továbbítja** az adatokat

UDP HASZNÁLATA

- Gyakori alkalmazása a multimédia folyam szállítás (streaming)
  - Veszteségtűrő
  - Sávszélesség-érzékeny
- További UDP felhasználás
  - DNS
  - SNMP (később)
- Az UDP feletti megbízható átvitelről az alkalmazási rétegben kell gondoskodni
  - Pl. alkalmazás specifikus hibakezelés

UDP szegmens formátuma



UDP ELLENŐRZŐÖSSZEG

- Cél a **hibák detektálása** (pl. „átbillent” bitek) az átvitt szegmensben
- Küldő oldal:
  - A szegmenst 16 bites egész számok sorozataként kezeli
  - Ellenőrzőösszeg
    - A számok összege
    - Invertálva
  - Beleteszi a fejlécbe
- Fogadó oldal:
  - Azonos módszerrel kiszámítja az összeget
  - Hozzáadja az ellenőrzőösszeg mezőt
  - Az eredmény csak egyeseket tartalmazhat
    - Ha nem, akkor hiba volt, eldobható a szegmens
    - Egyébként elfogadja (Biztos nem volt hiba?)

ELLENŐRZŐÖSSZEG SZÁMÍTÁSA

- Binárisan összeadjuk a számokat, és a legmagasabb helyi értékű bitek összeadásakor keletkező átvitelt a végén még hozzá kell adni az eredményhez

- Példa: két 16-bites egész összeadása

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
+	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
visszacsatolás: átvitel jobbról beléptetése	(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
összeg		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
ellenőrző összeg		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK