

# Számítógépes Hálózatok 2011

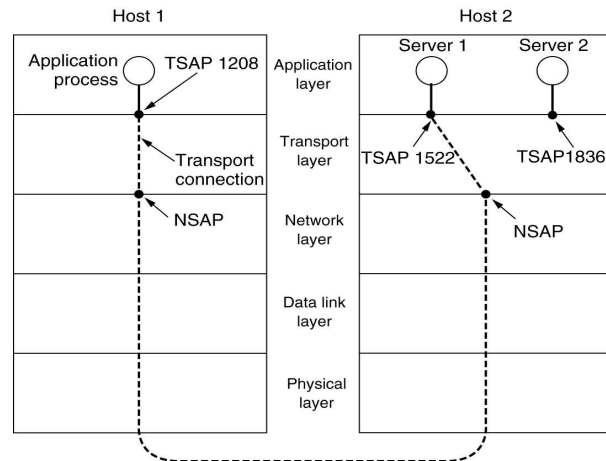
## 11. Szállítói réteg – TCP, Tahoe, Reno, AIMD, hatékonyság, fairness

### A szállítói réteg (transport layer) szolgáltatásai

- Kapcsolat nélküli vagy kapcsolat orientált (connectionless/connection oriented)
  - Gondoljunk az ISO/OSI ülés rétegére
- Megbízható vagy nem megbízható (reliable/unreliable)
  - „Best effort” vagy „Quality of Service”
  - Hibafelügyelet
- Torlódás felügyelet (congestion control) vagy torlódás felügyelet nélkül
- Lehetőség több végpontra egy végrendszeren (host)
  - Demultiplexálás
- Több interakciós modell támogatása
  - Byte-áram, üzenetek, „Remote Procedure Call“

## Multiplexálás a szállítói rétegben

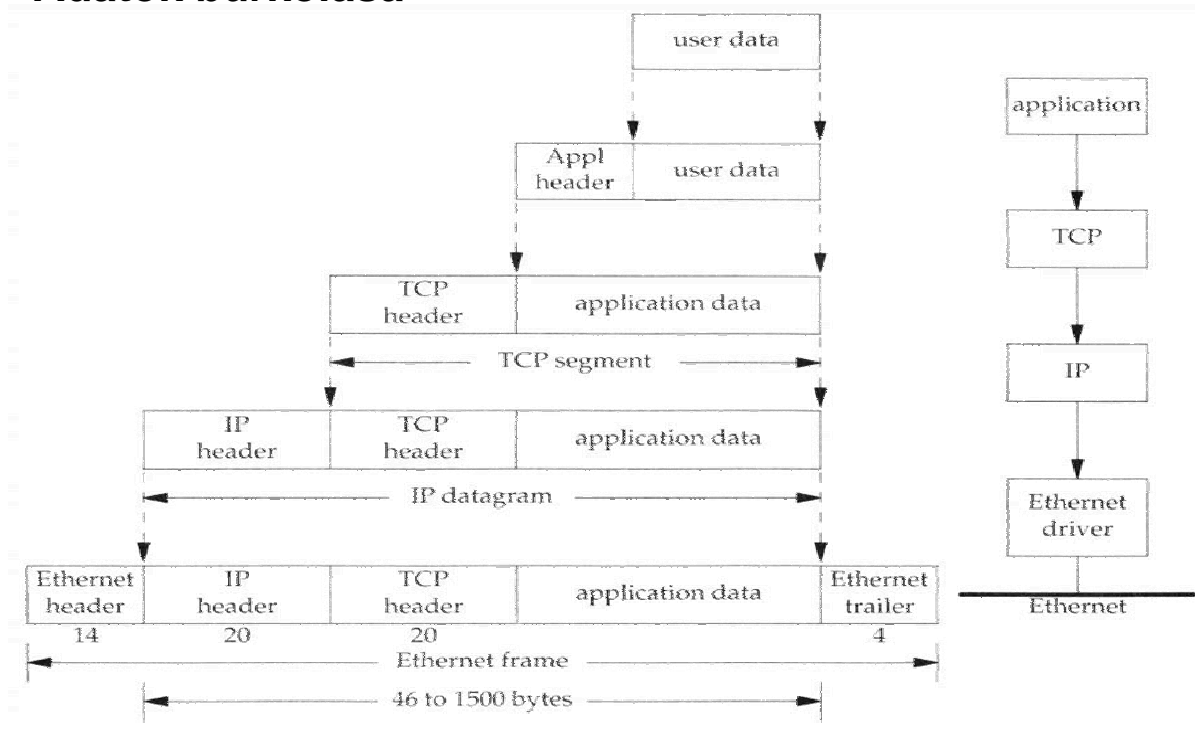
- A hálózati réteg az adatokat kontroll nélkül továbbítja a szállítói rétegnek
- A szállítói rétegnek az adatokat különböző felhasználásokhoz kell hozzárendelni:
  - pl. Web, Mail, FTP, ssh, ...
  - TCP/UDP ezt port-szám alapján teszi
  - pl. port 80 a Web-szerverhez



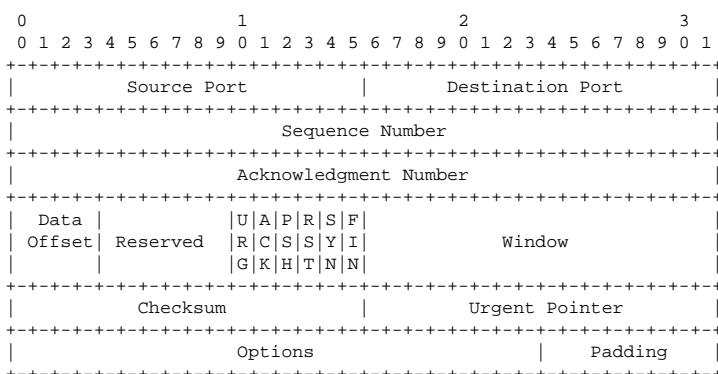
## Szállítói réteg (transport layer)

- TCP (transmission control protocol)
  - Megbízható adatfolyamot hoz létre két végpont között
  - A felhasználói réteg adatáramát csomagokra osztja
  - A másik oldal a csomagok fogadásától nyugtákat küld (Acknowledgment)
- UDP (user datagram protocol)
  - Egyszerű nem megbízható szolgáltatás csomagok küldésére
  - Az inputot egy datagrammá alakítja
  - A felhasználói réteg határozza meg a csomag méretét
- A csomagokat a hálózati réteg által küldi
- Routing nincs: végpont-végpont protokollok

# Adatok burkolása

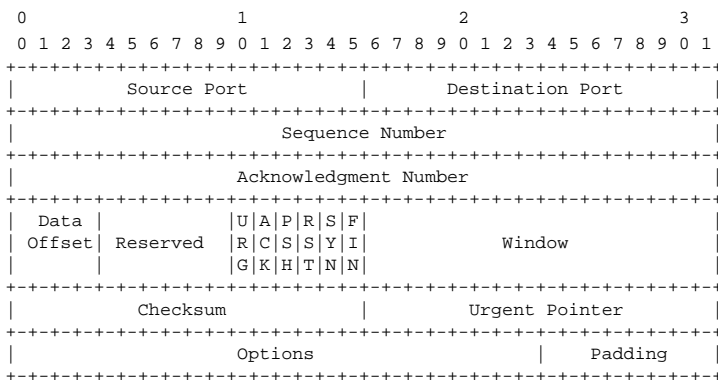


## TCP-fejléc (I)



- **Küldő-Port + Cél-Port-Nr.**
  - Megenged több TCP-kapcsolatot IP-címenként
- **Sorszám**
  - Minden adatbyte meg van számozva modulo  $2^{32}$
  - = a szegmens első byte-jának a száma
- **Nyugta szám**
  - Az ACK-Flag által aktivivált
  - Az első még nem feldolgozott adatbyte száma
  - = utolsó sorozatszám + utolsó adatmennyiség
- **„Checksum“**
  - Fejléchez és adatokhoz
- **Fejléchoossz (data offset)**
  - A változó hosszúságú opció-mező miatt

## TCP-Fejléc (II)



- Opció-mező pl. MSS (maximum segment size):

- A fogadó megadja a kívánt csomagméretet
- Tekintet nélkül az IP MTU-ra (max. transmission unit)
- Fragmentálás lehetséges az IP által

- FLAGS (függetlenül felhasználhatók)

- URG: sürgős (urgent)
- ACK: nyugta (acknowledgment)
  - Aktiválja a nyugta számot
- PSH: Push
  - Gyors adattovábbítás a felhasználói rétegnek
- RST: Reset
  - A válasz hiba esetén: connection reset by peer
- SYN: Synchronize
  - Kapcsolatfelépítés és a kezdő sorszám megadása
- FIN: Finished
  - (Egy) adatfolyam befejezése

## TCP (I)

TCP (Transmission Control Protocol) egy kapcsolatorientált megbízható szolgáltatás bidirekcionális byte-folyamokhoz

TCP Kapcsolatorientált

- Két résztvevő. Egy egy résztvevő **socket** által azonosított: socket: **IP-cím** és **port**
- TCP-kapcsolat egyértelműen azonosított egy **socketpár** által
- Nincs broadcast sem multicast
- Kapcsolatfelépítés és lezárás szükséges
- Amíg egy kapcsolat nincs (rendesen) lezárva, addig aktív

## TCP (II)

TCP egy kapcsolatorientált megbízható szolgáltatás bidirekcionális byte-folyamokhoz

TCP megbízható

- Minden adatcsomag megérkezését nyugtázza (*acknowledgment*)
- A nem nyugtázott adatcsomagokat újraküldi
- "Checksum" a fejléchez és csomaghoz
- TCP számozza a csomagokat és sorbarendezi a fogadónál
- Törli a duplikált csomagokat

## TCP (III)

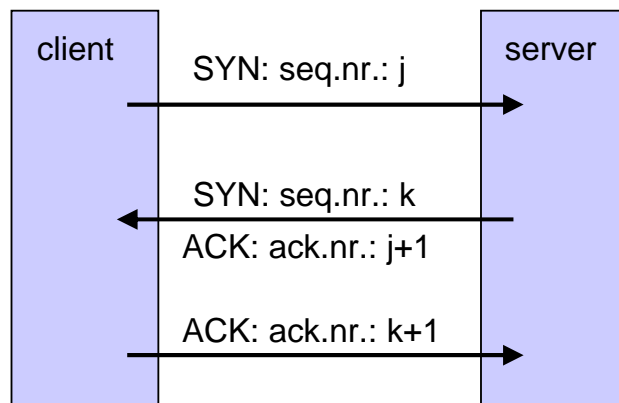
TCP egy kapcsolatorientált megbízható szolgáltatás bidirekcionális byte-folyamokhoz

TCP egy szolgáltatás bidirekcionális byte-folyamokhoz

- Az adatok két egymással ellentétes irányú byte-sorozatként (=8 bit) kerülnek átvitelre
- A tartalom nem interpretálódik
- Az adatcsomagok időbeli viselkedése megváltozhat: átvitel sebessége növekedhet, csökkenhet, más késés, más sorrendben is megérkezhetnek
- Megpróbálja az adatcsomagokat időben egymáshoz közel kiszállítani
- Megpróbálja az átviteli közeget hatékonyan használni  
= kevés csomag

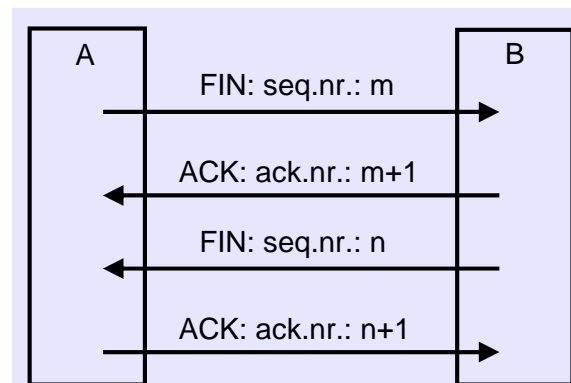
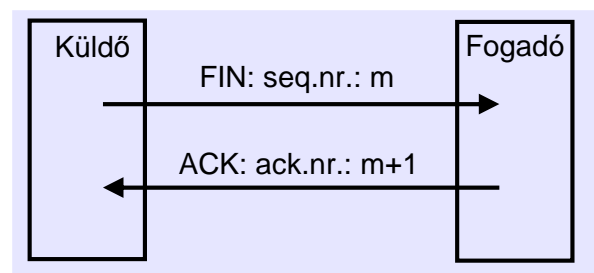
## Kapcsolatfelépítés

- Rendszerint Client-Server-kapcsolat
  - Ekkor felépítés 3 TCP-csomaggal (=3 szegmens)
  - Az első SYN-szegmensben az MSS (maximum segment size) is átvitelre kerül



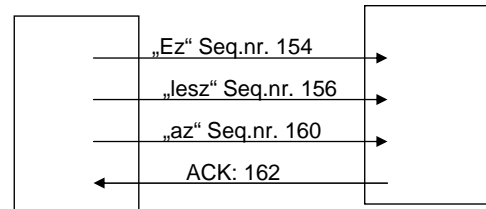
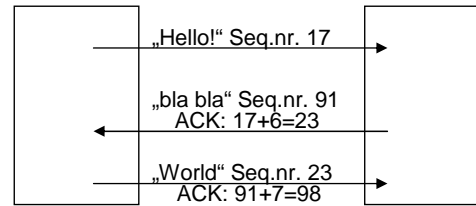
## Kapcsolat lezárása

- Félig lezárás (half-close)
  - A küldő jelzi a kapcsolat befejezését egy FIN-szegmensben és vár annak nyugtájára
  - Az ellenkező irányban továbbra is lehet küldeni
- Két félig lezárás lezárja a TCP-kapcsolatot

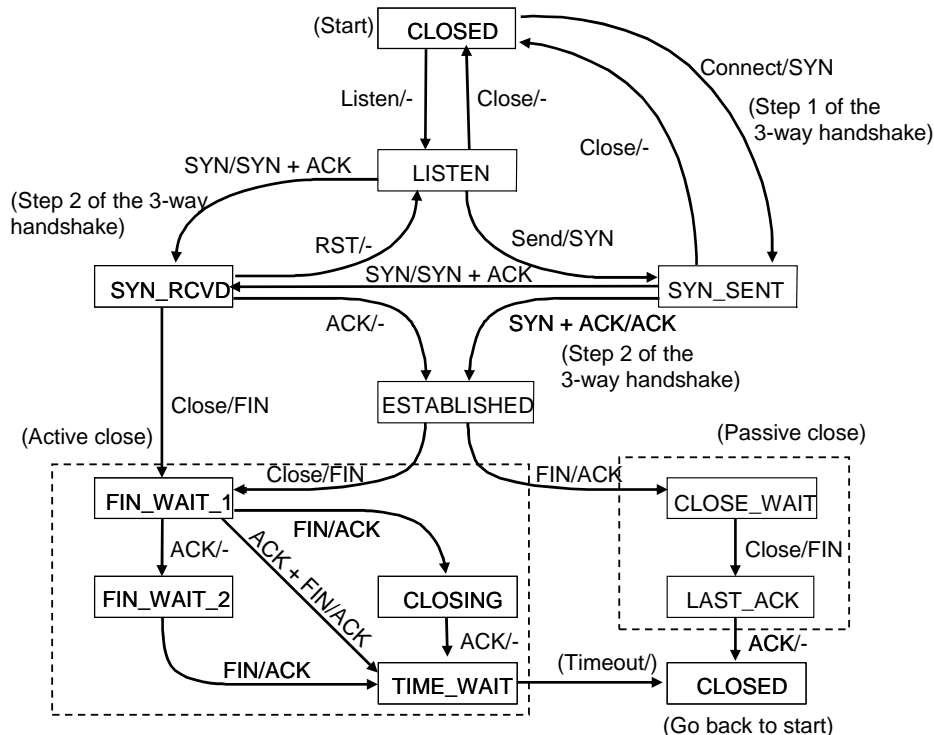


## Nyugták (acknowledgement -- ACK)

- Hátizsák technika „piggybacking“
  - A nyugták (ACK) az ellenkező irány adatszegmensein „utaznak“
- Egy nyugta több adatszegmenst is nyugtázhat
  - Ha nincs küldeni való adat, késleteti az ACK-kat

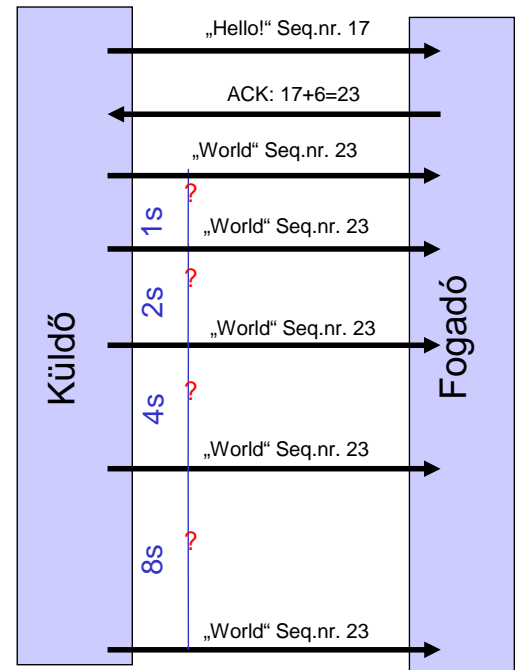


## TCP állapot átmeneti diagramm



## Exponenciális visszavétel (exponential backoff)

- **Retransmission Timeout (RTO)**
  - szabályozza az időközt a küldés és egy duplikátum újraküldése között, ha egy nyugta kimarad
- Mikor nem kerül nyugtázásra egy TCP-csomag?
  - Ha a nyugta lényegesen több időt vesz igénybe, mint az átlagos „round trip time” (RTT)
    - 1. Probléma: RTT mérés
    - 2. Probléma: Csak a nyugta jön túl későn
  - Küldő
    - Vár az RTO-nak megfelelő ideig
    - Ha nem érkezett nyugta, újraküldi a csomagot és növeli
$$RTO \leftarrow 2 RTO \quad (RTO = 64 \text{ másodpercig})$$
- RTO újraszámolása, ha a csomagok nyugtázódnak



## A Round Trip Time (RTT) becslése

- A TCP-csomag nem nyugtázottnak számít, ha a nyugta „lényegesen” tovább tart, mint az RTO
- RTT nem számítható on-line (csak visszatekintve)
- RTT erősen ingadozik
- Ezért: Retransmission Timeout Value nagyvonalú becsléssel:
  - RFC 793: ( $M :=$  utoljára mért RTT)
$$R \leftarrow \alpha R + (1 - \alpha) M, \quad \text{ahol } \alpha = 0,9$$
$$RTO \leftarrow \beta R, \quad \text{ahol } \beta = 2$$
  - Jacobson 88: a becslés nem elég robusztus, ezért
$$A \leftarrow A + g (M - A), \quad \text{ahol } g = 1/8$$
$$D \leftarrow D + h (|M - A| - D), \quad \text{ahol } h = 1/4$$
$$RTO \leftarrow A + 4D$$
- Többszörösen elküldött csomagoknál nem aktualizálunk

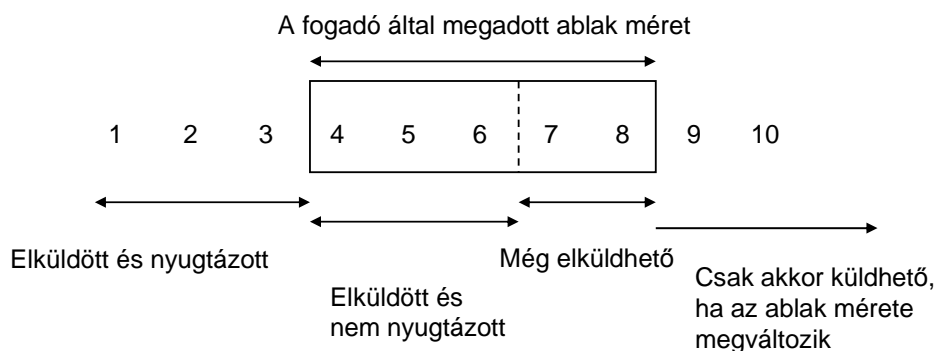


## TCP – Nagle algoritmusa

- Hogyan biztosíthatjuk,
  - hogy kis csomagok időben egymáshoz közel kerüljenek kiszállításra
  - és hogy sok adat esetén nagy csomagok előnyben részesüljenek?
- **Nagle algoritmusa:**
  - Kis csomagok nem kerülnek addig küldésre, amíg nyugták hiányoznak
    - egy csomag kicsi, ha az adathossz  $< \text{MSS}$
  - Ha a korábban küldött csomag nyugtája megérkezik, küldi a következőt
- Tulajdonsága
  - Önmagát ütemező: Gyors kapcsolat = sok kis csomag

## Csúszó Ablakok (sliding windows)

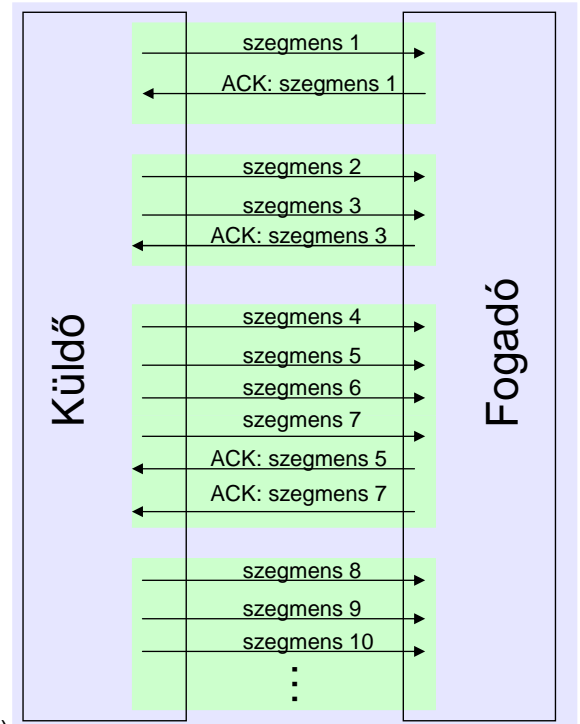
- Adatátráta szabályozása ablak segítségével
  - A fogadó meghatározza az ablak méretet (wnd) az ACK-szegmensek TCP-fejlécében
  - Ha a fogadó fogadási puffere tele van, akkor  $\text{wnd}=0$  -t küld
  - Máskülönben a fogadó  $\text{wnd}>0$  -t küld
- A küldőnek be kell tartani:
  - Az elküldött nem nyugtázott adatcsomagok száma  $\leq$  ablak mérete



## Lassú Start (slow start)

- A küldőnek nem szabad a fogadó által felajánlott ablakméretet azonnal kihasználni
- Második ablak: Congestion-ablak (cwnd: congestion window)
  - A küldő választja
  - Az ablak amiben küld:  $\min\{wnd, cwnd\}$
  - Kezdetben:
$$cwnd \leftarrow MSS$$
  - Minden csomagnál a megkapott nyugta után nő
$$cwnd \leftarrow cwnd + MSS$$
(azaz megduplázódik minden RTT után)
  - Addig, amíg egyszer egy nyugta kimarad

Slow start = exponenciális növekedés  
(hisztórikus elnevezés: korábban még agresszívebb sémák)



## Torlódás elkerülés (congestion avoidance) TCP Tahoe

- Jacobson 88:
  - Paraméter: cwnd és ssthresh (= slow-start-küszöb, slow start threshold)
- 1. Kapcsolatfelépítés:
$$cwnd \leftarrow MSS \quad ssthresh \leftarrow 65535$$
- 2. Csomagvesztésnél, azaz nyugta ideje > RTO: **multiplicatively decreasing**
$$cwnd \leftarrow MSS \quad ssthresh \leftarrow \max \left\{ 2 MSS, \frac{\min\{cwnd, wnd\}}{2} \right\}$$
- 3. Nyugta jön a szegmenshez és  $cwnd \leq ssthresh$ : **slow start**
$$cwnd \leftarrow cwnd + MSS$$
- 4. Nyugta jön a szegmenshez és  $cwnd > ssthresh$ : **additively increasing**
$$cwnd \leftarrow cwnd + MSS \frac{MSS}{cwnd}$$

## TCP Tahoe

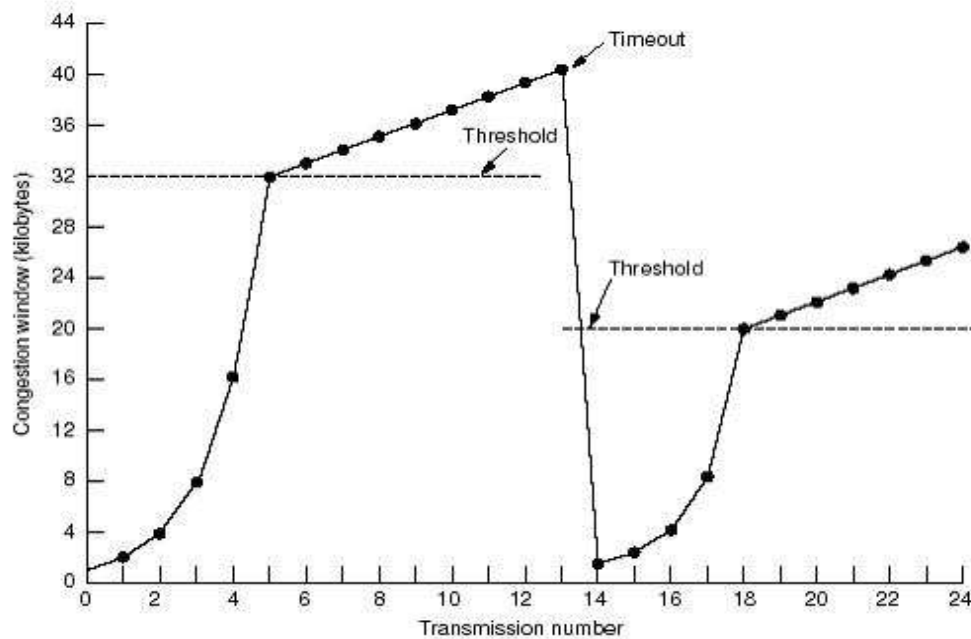


Fig3

pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

## Fast Retransmit és Fast Recovery

- TCP Tahoe [Jacobson 1988]:
  - Ha csak egy csomag veszik el, akkor
    - a csomagot újraküldi + a fennmaradó ablakot
    - és egyidejűleg slow start
  - Fast retransmit
    - ha ugyanazon csomaghoz 3 nyugta-duplikátum (azaz 4 azonos nyugta) érkezik (triple duplicate ACK),
    - újraküldi az elvesztet csomagot, egyidejűleg slow start
- TCP Reno [Stevens 1994]
  - Fast retransmit után:
    - $ssthresh \leftarrow \max(\min(wnd, cwnd)/2, 2 \text{ MSS})$
    - $cwnd \leftarrow ssthresh + 3 \text{ MSS}$
  - Fast recovery a fast retransmit után
    - Miden további nyugta után növeli a rátát:
    - $cwnd \leftarrow cwnd + \text{MSS}$
  - Congestion avoidance: amikor új adat nyugtája megérkezik (ujraküldés ACK-ja):
    - $cwnd \leftarrow ssthresh$

## Torlódás elkerülési elv: AIMD

- A TCP a „fast recovery” mechanizmussal lényegében a következőképp viselkedik:

$x$ : csomagok száma per RTT

- Kapcsolatfelépítés:

$$x \leftarrow 1$$

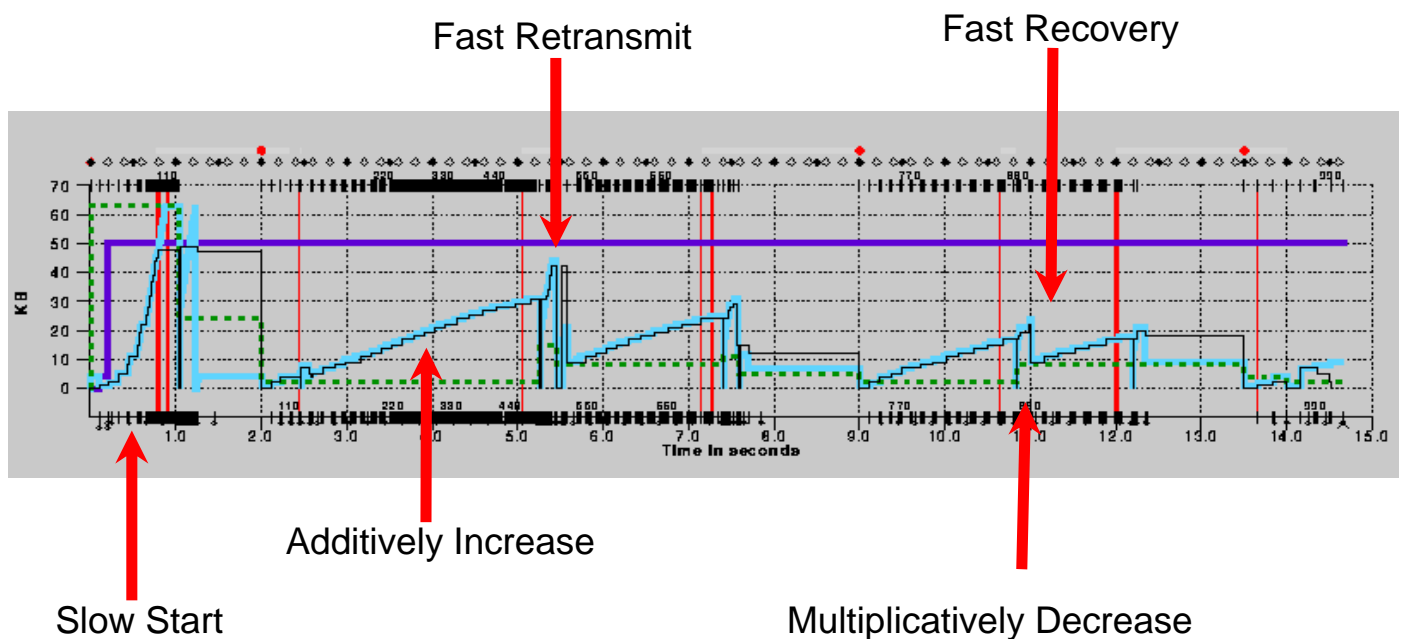
- Csomagvesztésnél, MD: multiplicative decreasing

$$x \leftarrow x/2$$

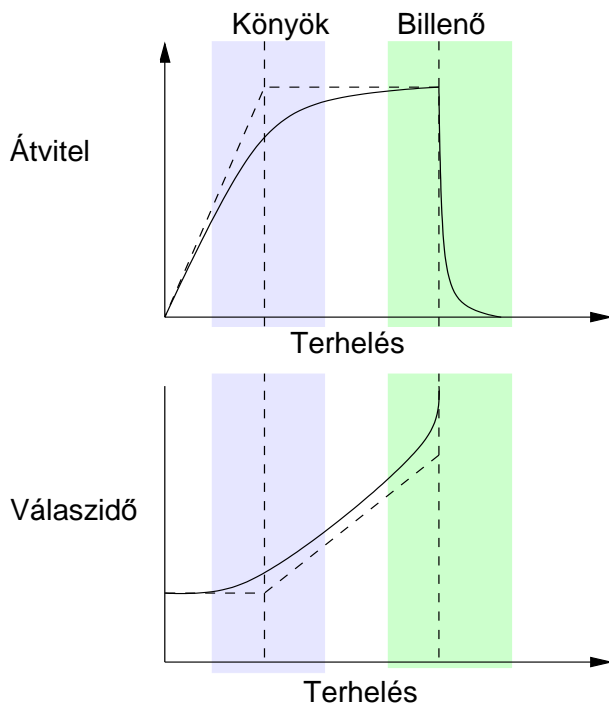
- Nyugtázott szegmenseknél, AD: additive increasing

$$x \leftarrow x + 1$$

## Példa: TCP Reno „in akción”



## Additive Increase Multiplicative Decrease (AIMD): Fairness és Hatékonyság



A hálózati terhelés az átvitelrel és a válaszidővel kölcsönösen hat egymásra.

- Az átvitel maximális, ha a terhelés a hálózat kapacitását majdnem eléri.
- Ha a terhelés tovább nő, túlcserélnek a pufferek, csomagok vesznek el, újra kell küldeni, drasztikusan nő a válaszidő. Ezt a toródást **congestion**-nak nevezzük.
- Ezért a maximális terhelés helyett, ajánlatos a hálózat terhelését a könnyök közelében beállítani. Itt a válaszidő csak lassan emelkedik, míg az adatátvitel már a maximum közelében van.
- Egy jó torlódáselkerülési (*congestion avoidance*) stratégia a hálózat terhelését a könnyök közelében tartja: **hatékonyság**. Emellett fontos, hogy minden résztvevőt egyforma rátával szolgáljunk ki: **fairness**.

## AIMD Fairness és Hatékonyság – Egy egyszerű modell

- $n$  résztvevő, forduló-modell
- résztvevő  $i$  adatrátája a  $t$ -edik fordulóban  $x_i(t)$
- Kezdeti adatráták:  $x_1(0), \dots, x_n(0)$
- A visszacsatolás (feedback) forduló  $t$  után:  $y(t) = 0$ , ha  $\sum_{i=1}^n x_i(t) \leq K$   
 $y(t) = 1$ , ha  $\sum_{i=1}^n x_i(t) > K$
- Minden résztvevő aktualizálja az adatrátáját a  $t+1$ -edik fordulóban:

$$x_i(t+1) = f(x_i(t), y(t))$$

- Increase-stratégia  $f_0(x) = f(x, 0)$
- Decrease-stratégia  $f_1(x) = f(x, 1)$
- Tekintsük a következő lineáris függvényeket :

$$f_0(x) = a_I + b_I x, \quad f_1(x) = a_D + b_D x$$

## AIMD Fairness és Hatékonyság– A Modell

- A következő lineáris függvényeket vizsgáljuk:

$$f_0(x) = a_I + b_I x, \quad f_1(x) = a_D + b_D x$$

- Érdekes speciális esetek:

- MIMD: Multiplicative Increase/Multiplicative Decrease

$$f_0(x) = b_I x, \quad f_1(x) = b_D x, \quad \text{ahol } b_I > 1, b_D < 1.$$

- AIAD: Additive Increase/Additive Decrease

$$f_0(x) = a_I + x, \quad f_1(x) = a_D + x, \quad \text{ahol } a_I > 0, a_D < 0.$$

- AIMD: Additive Increase/Multiplicative Decrease

$$f_0(x) = a_I + x, \quad f_1(x) = b_D x, \quad \text{ahol } a_I > 0, b_D < 1.$$

## AIMD Fairness és Hatékonyság

- Hatékonyság

- Terhelés:  $X(t) := \sum_{i=1}^n x_i(t)$

- Mérték:  $|X(t) - K|$

- Fairness:  $x=(x_1, \dots, x_n)$  esetén:

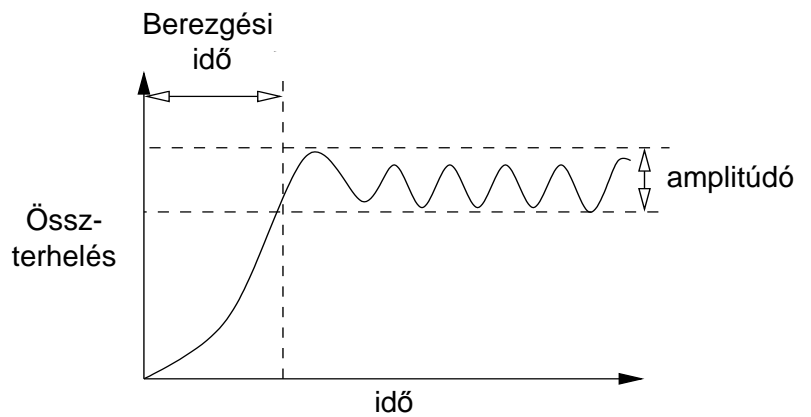
$$F(x) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n (x_i)^2}.$$

- $1/n \leq F(x) \leq 1$
- $F(x) = 1 \leftrightarrow$  absolut Fairness
- skálázástól független
- Folytonos, differenciálható
- Ha  $n$  közül  $k$  fair, a többi 0, akkor  $F(x) = k/n$

# Konvergencia

- Konvergencia nem lehetséges
- Legjobb esetben oszcilláció az optimális érték körül
  - Az oszcilláció amplitúdója A

$$A = \inf_{t_0 \geq 0} \sup_{t \geq t_0} |X(t) - K| .$$

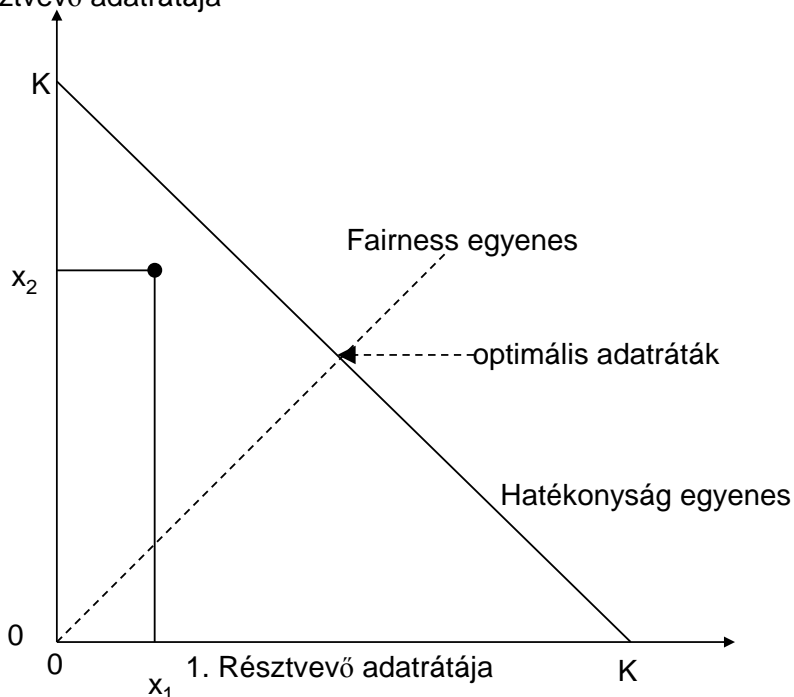


- Berezgési idő T

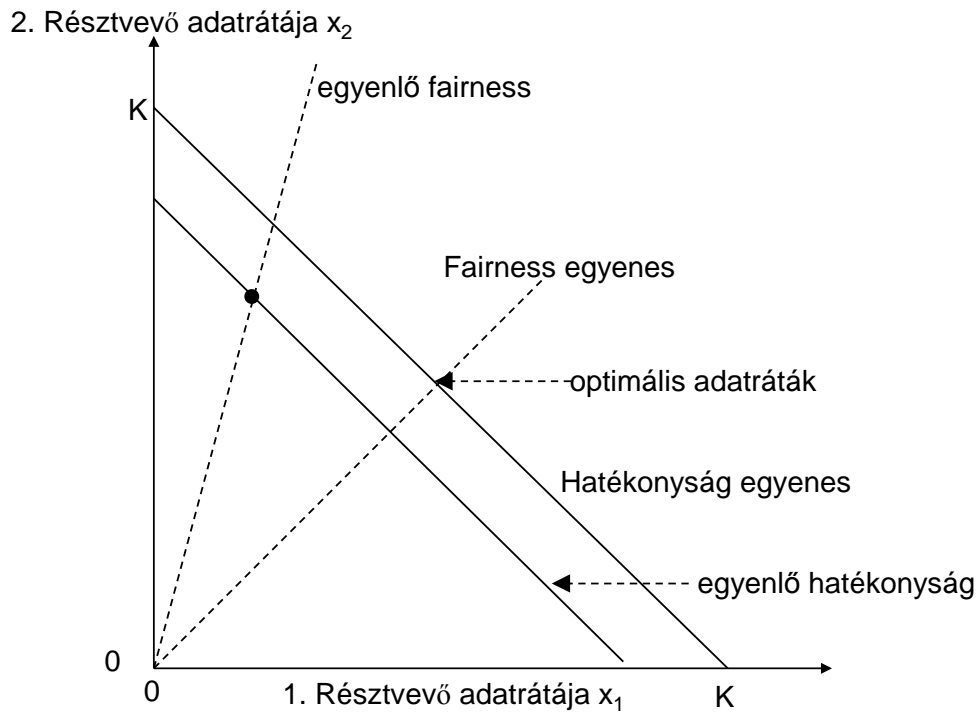
$$T = \min\{t_0 \mid \forall t \geq t_0 : |X(t) - K| \leq A\} .$$

# Vektor Ábrázolás (I)

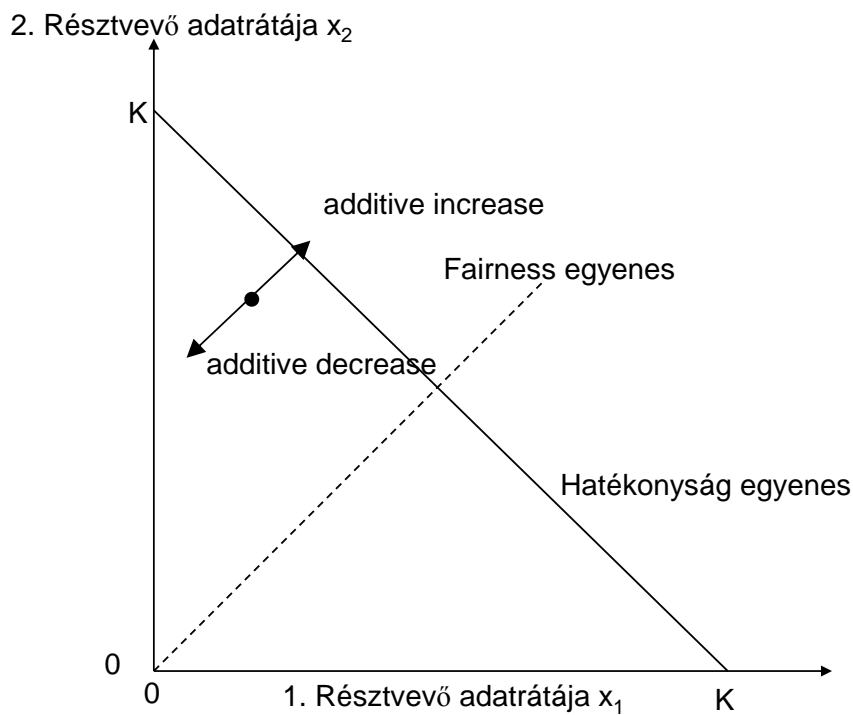
2. Résztevő adatrátája



## Vektor Ábrázolás (I)

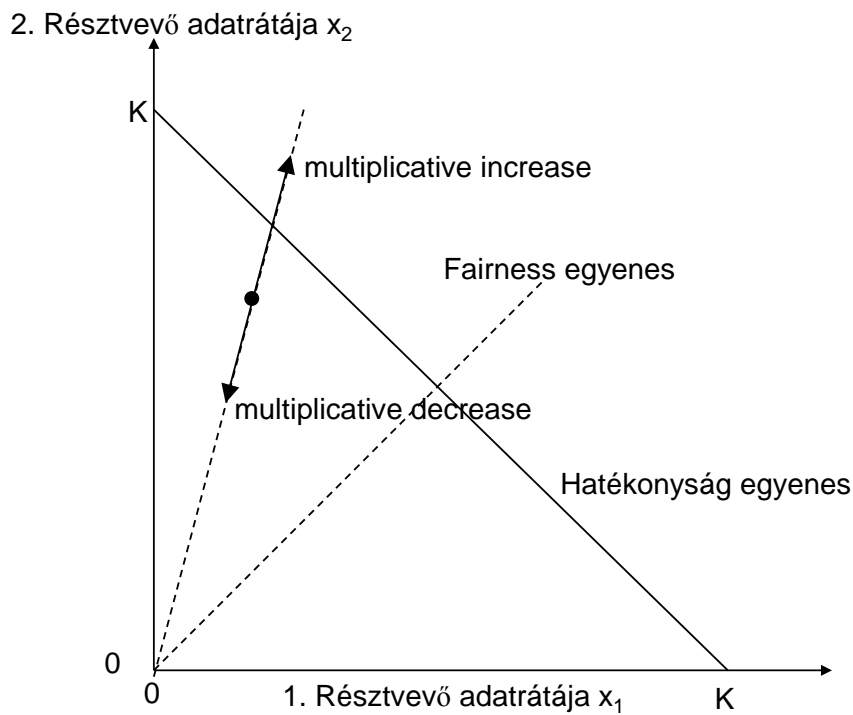


## Vektor Ábrázolás (I)

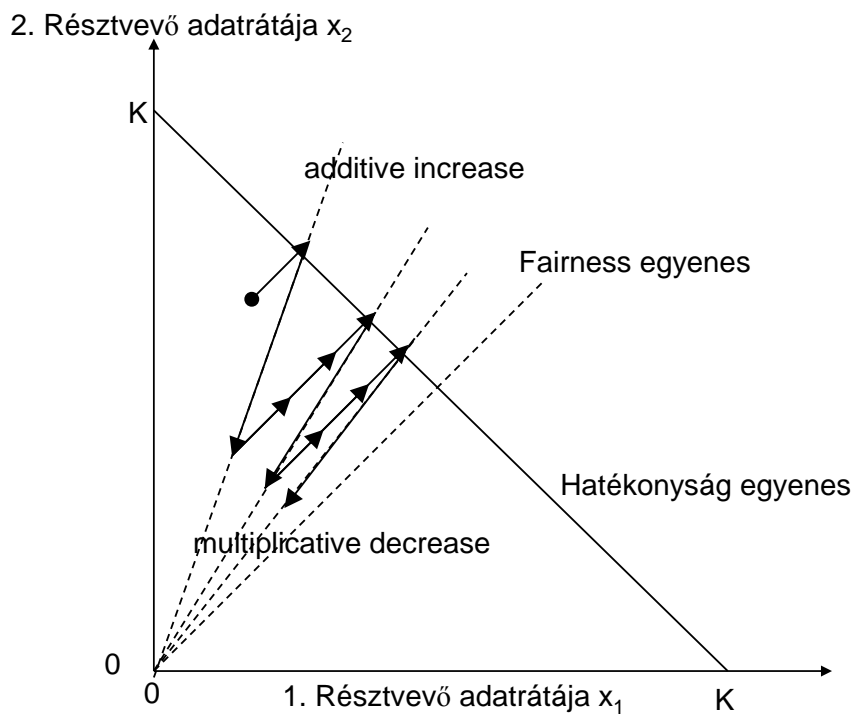




# Vektor Ábrázolás (I)



# Vektor Ábrázolás (I)



## TCP összefoglalás

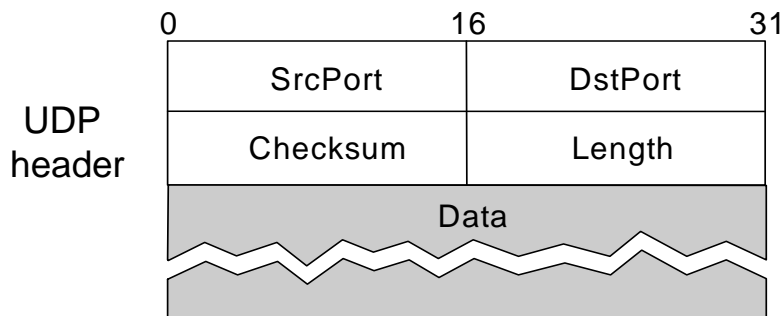
- TCP egy megbízható byte-folyamot hoz létre
  - Hibafelügyelet "Go-Back-N" által
- Congestion control
  - Ablak alapú
  - AIMD, Slow start, *Congestion Threshold*
  - Folyamfelügyelet *Window* által
  - Kapcsolatfelépítés
  - Nagle algoritmus

## TCP fairness & TCP friendliness

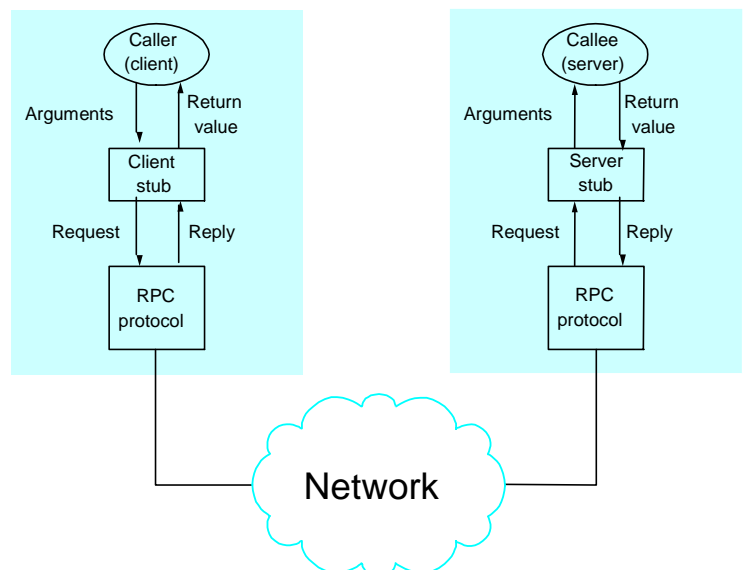
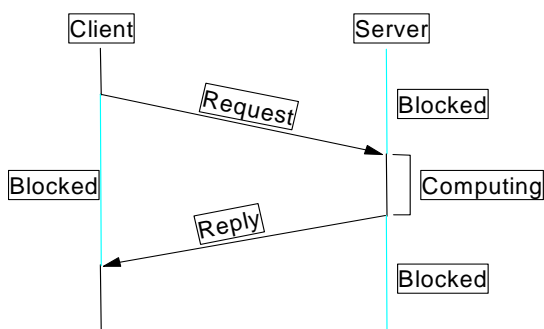
- TCP
  - Dinamikusan reagál a rendelkezésre álló sávszélességre
  - A sávszélesség fair felosztása
    - Ideális esetben:  $n$  TCP-kapcsolat mindegyike  $1/n$  részt kap
- TCP más protokollokkal
  - Reakció más szállítói protokollok terhelésétől függ
    - pl. UDP-ben nincs congestion control
  - Más protokollok mindenkor felhasználhatók
  - UDP és más protokoll el tudja nyomni a TCP kapcsolatokat
- Véggövetkeztetés
  - A szállítói protokolloknak TCP-kompatibilisnek kell lenni (TCP friendly)

# UDP

- User Datagram Protocol (UDP)
  - Egy nem megbízható kapcsolat nélküli szállítói réteg protokoll csomagoknak
- Fő funkció:
  - A hálózati réteg csomagjainak demultiplexálása
- Egyéb funkció (opcionális):
  - Checksum: UDP header + data



## Remote procedure call – Struktúra



- Komplexebb interakcióhoz:
  - Egy függvény hívása egy másik számítógépen
- Cél: Transzparens protokoll a hívónak/hívottnak