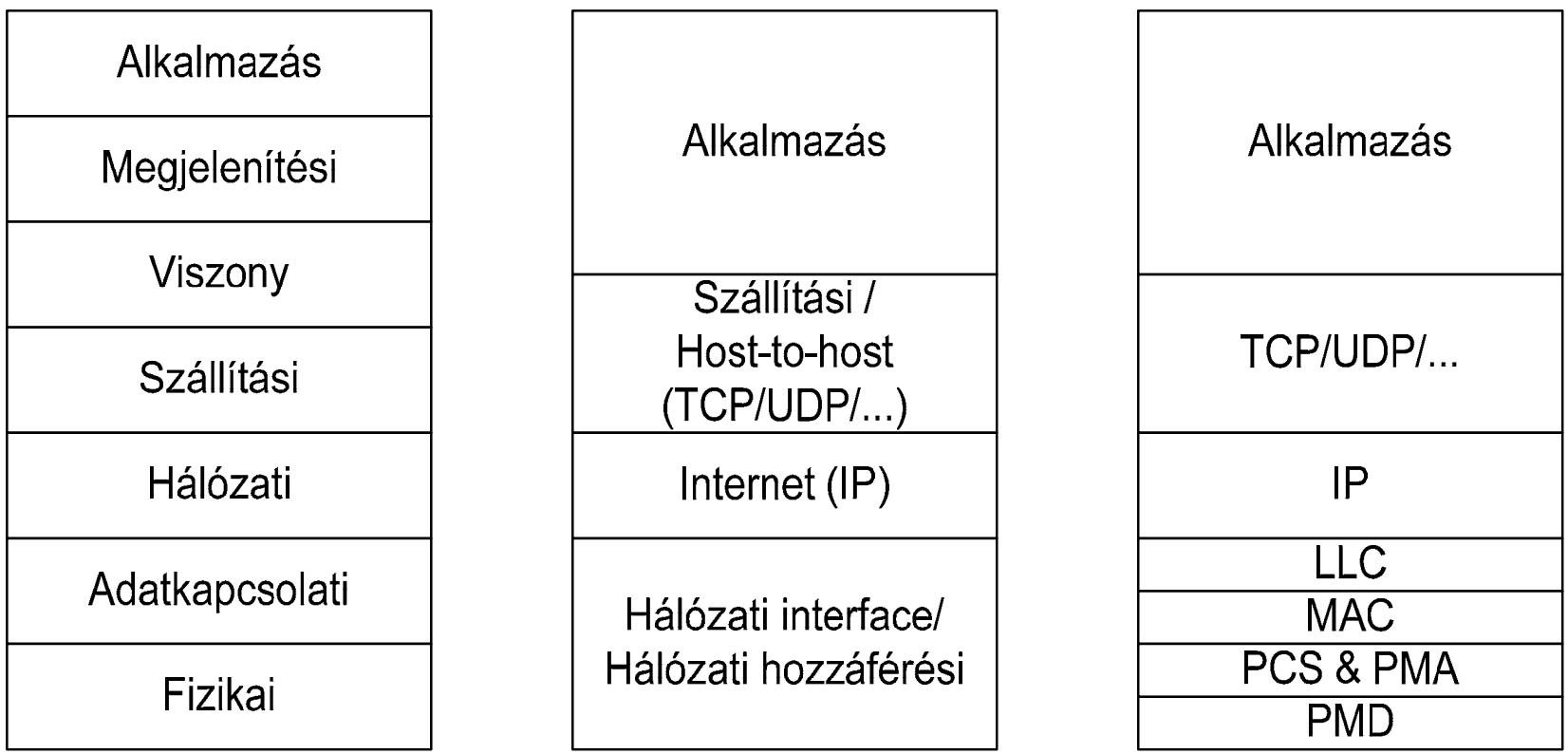
SZÁLLÍTÁSI (TRANSPORT, HOST-TO-HOST) PROTOKOLLOK

UDP és TCP

Dr. Simon Vilmos
adjunktus
BME Hálózati Rendszerek és Szolgáltatások Tanszék
svilmos@hit.bme.hu

2013.Április 16.

A TCP/IP architektúra és az ISO/OSI rétegmodell



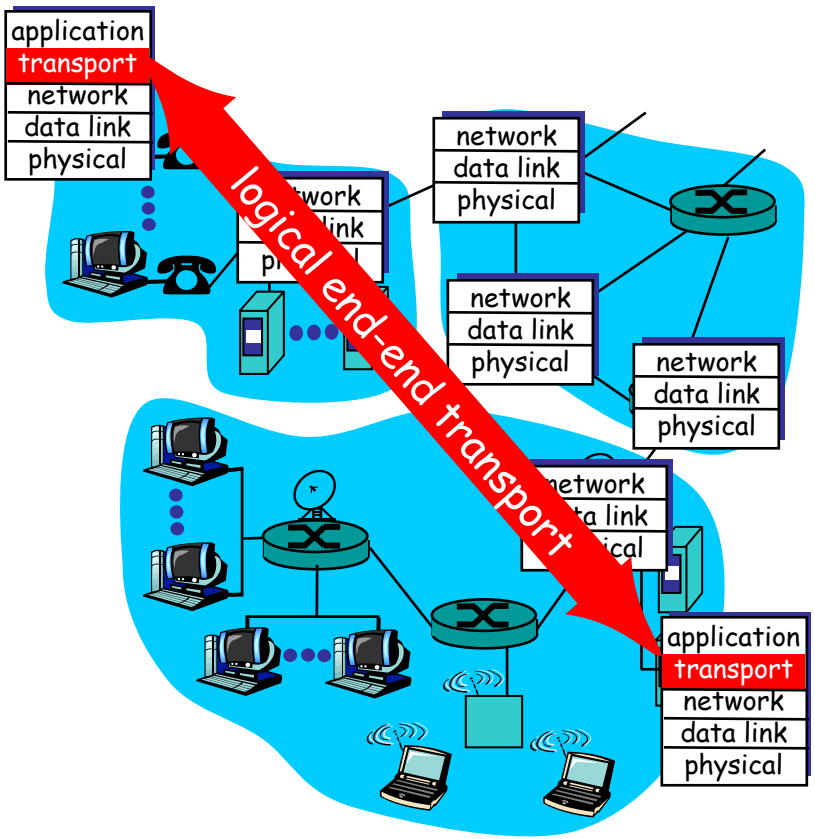
IP: Internet Protocol
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
LLC: Logical Link Control

MAC: Medium Access Control
PCS: Physical Coding Sublayer
PMA: Physical Medium Attachment
PMD: Physical Medium Dependent

A hálózati és a szállítási réteg

- *hálózati réteg:* végpontok („host”-ok) közötti logikai kapcsolatok
- *szállítási réteg:* alkalmazások (process) közötti logikai kapcsolatok
  - a hálózati réteg szolgáltatásainak igénybevételére alapozva **megbízható** átvitel a **transzport entitások** között (transport entity)
  - Feladatai:
    - forgalom szabályozás
    - multiplexelés
    - hibadetektálás, javítás (pl. Automatic Repeat reQuest – ARQ)
    - sorrendhelyes átvitel
    - csomagképzés (szegmensek) és csomagok visszaállítása a felsőbb rétegek számára
      – byte alapú átvitel

A szállítási réteg



Logikai kapcsolatok a végpontokban futó **alkalmazások között**

A szállítási protokollok a **végpontokban futnak**, a csomópontokban nem

Az alkalmazások adategységeit szállítási protokoll-adategységekbe tördeljük, a kapottakból pedig összerakjuk

Szállítási protokollok: UDP és TCP

- **UDP – User Datagram Protocol**
- **TCP – Transmission Control Protocol**

- Az UDP és TCP közös képességei:
  - Portok kezelése
  - Multiplexelési képesség

- Alapvető különbség az UDP és a TCP között:
  - UDP összeköttetésmentes (connectionless),
  - TCP összeköttetés-alapú (connection–oriented)

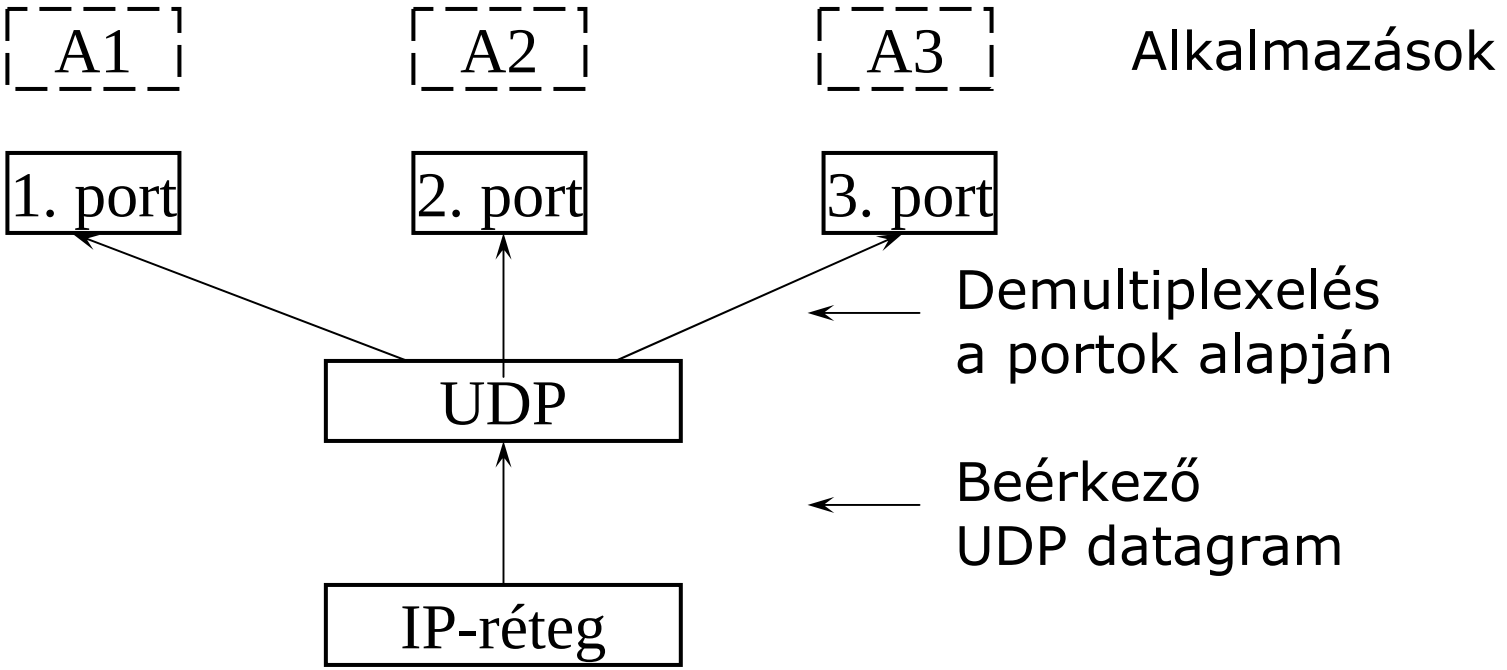
  transzport-szolgáltatást nyújt

Az UDP és TCP közös képességei (1)

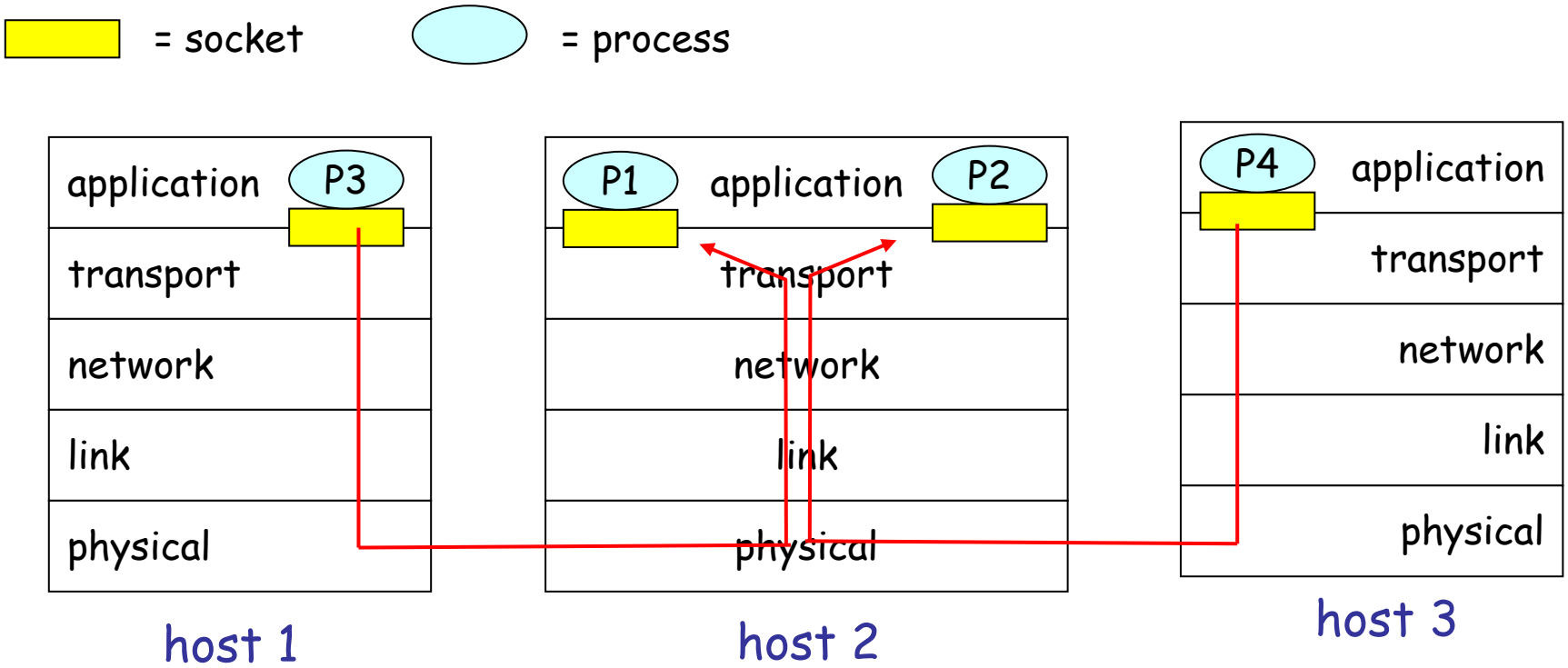
- Portok kezelése:
  - Az IP-rétegben a csomagok végpontnak, „host”-nak vannak címezve
  - A végpontokon belül: több alkalmazás, folyamat
  - Megkülönböztetésük: portok használatával
  - **Foglalt** (reserved, „well-known”) és **rendelkezésre álló** (available) portszámok
  - Foglalt portok: ide mindig lehet küldeni datagrammokat
    - 0...1023 közötti portszámok
    - pl. 80: HTTP, 21: FTP, 69: TFTP (ezek főként TCP-re)
  - Az UDP-en belül megállapításra kerülnek az alkalmazandó portszámok
- Multiplexelés /demultiplexelés
  - A portmechanizmus segítségével

Az UDP és TCP közös képességei (2)

Multiplexelés és demultiplexelés
Példa:



Multiplexelés-demultiplexelés



Socket: magyarázat

- Socket: interfész, „ajtó” az alkalmazás és a hálózat között
- A socket-et az alábbiak jellemzik:
  - transzportprotokoll (UDP v. TCP)
  - saját IP cím
  - saját portszám

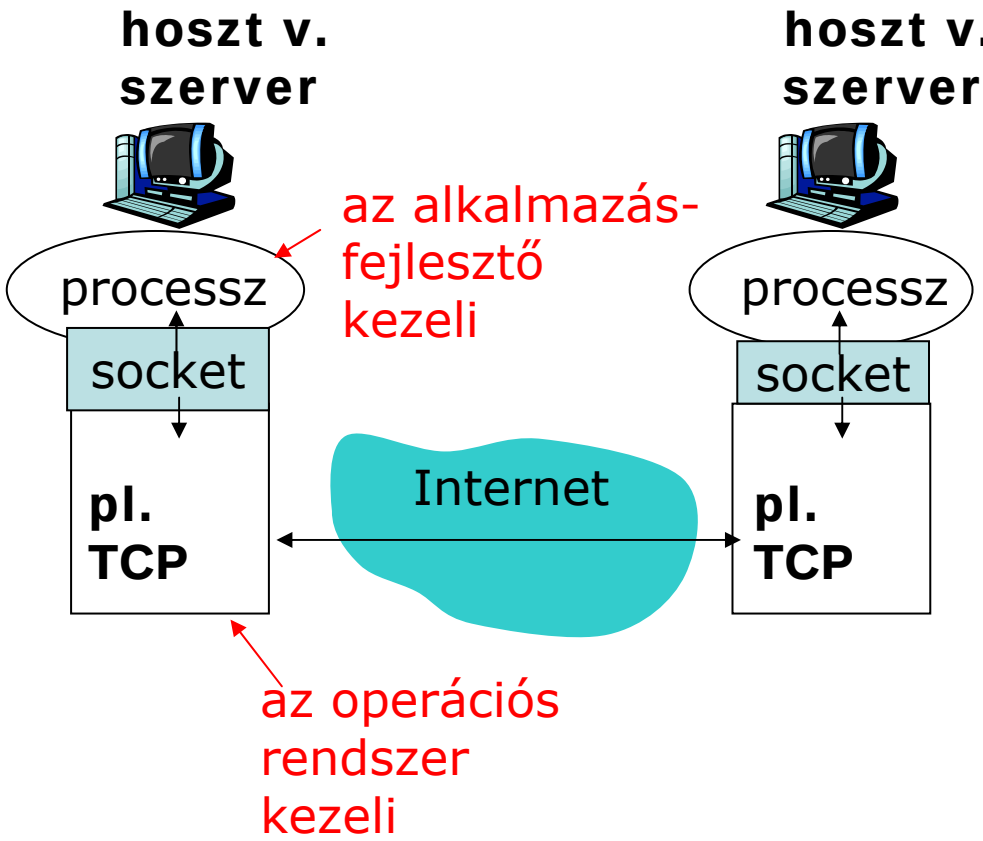
  } helyi socket cím

  - *(opcionális) távoli host IP címe*
  - *(opcionális) távoli alkalmazás portszáma*

  } távoli socket cím

  } socket pár
- Leegyszerűsítve: socket = IP cím + portszám
- Az operációs rendszer a bejövő IP csomagokat a fentiek alapján továbbítja az alkalmazásnak, kiszedve ezeket a megfelelő PDU-kból

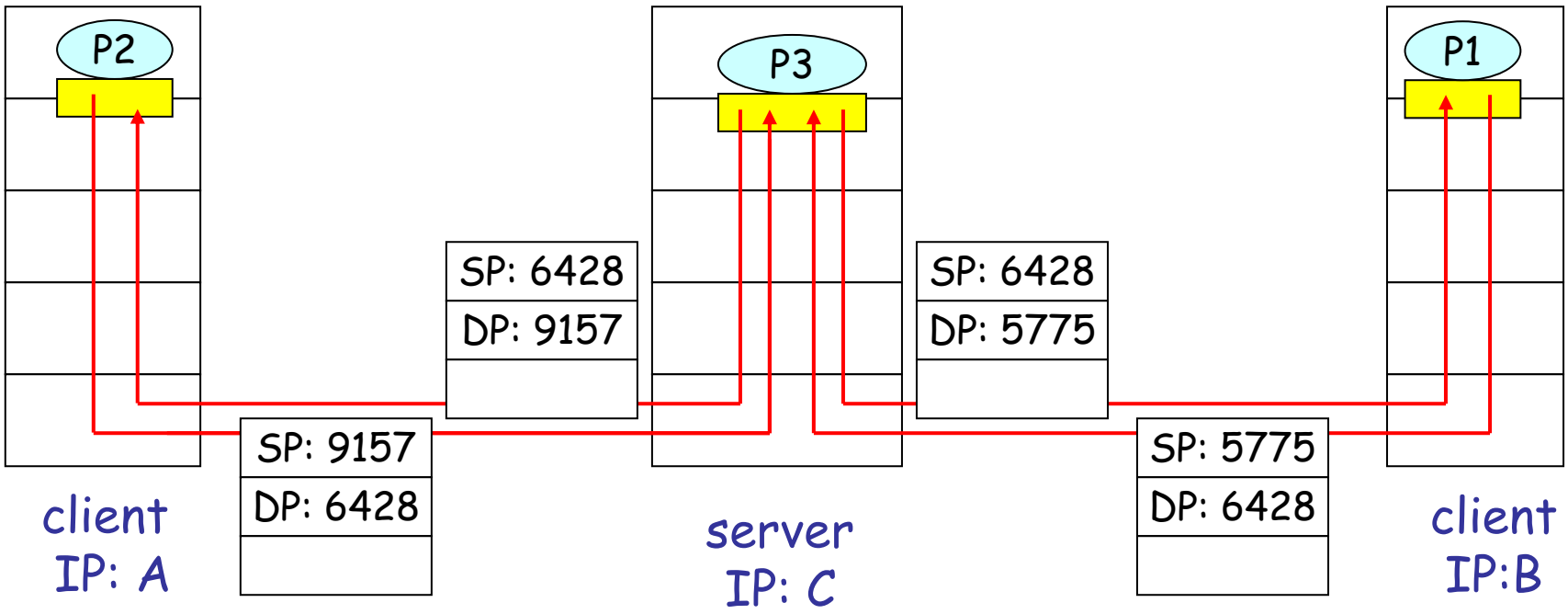
Socket: illusztráció



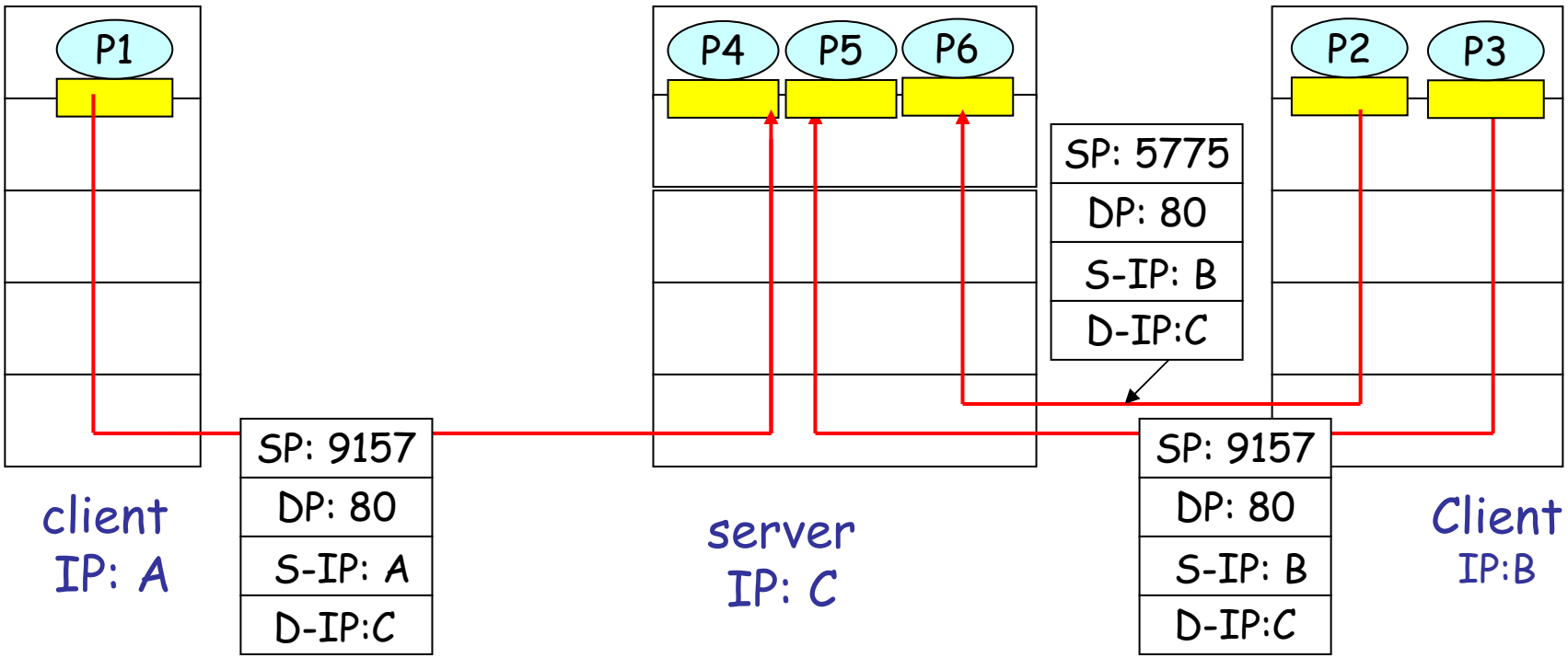
Socket típusok

- Datagramm socket
  - Összeköttetésmentes socket, UDP használja
- Stream socket
  - Összeköttetés-alapú socket, TCP használja
- Raw (IP) sockets
  - Routerekben és hálózati eszközökben
  - Pl. ICMP, IGMP, OSPF

Multiplexelés-demultiplexelés, összeköttetésmentes eset (UDP)



Multiplexelés-demultiplexelés, összeköttetés-alapú eset (TCP)



TCP kliens-szerver socket kezelés

- Szerver létrehoz **„hallgató” módban** lévő socketeket
  - várja a kliensek kapcsolatfelvételét
  - ilyenkor távoli cím: 0.0.0.0, távoli port: 0
- Kapcsolatfelvétel után: **dedikált socket** minden kapcsolathoz
  - Socket-socket közötti virtuális áramkörkapcsolás: TCP kapcsolat, duplex byte folyam
- Szerver különböző TCP socketeket hozhat létre ugyanazzal a helyi IP címmel és port számmal
  - mivel a távoli host IP címével, portjával **más socket párt alkot**
  - szerver gyerek processz összerendelése a kliens processzével
- UDP-ben **nincs dedikált socket**
  - Nincs külön gyerek processz minden távoli processzhez, egy processz kommunikál velük

És ahol nincs transzport réteg implementálva?

- Egyes hálózati elemekben nincs implementálva a transzport réteg
  - Router hálózati rétegben
  - Switch adatkapcsolati rétegben
- De tűzfalak, NAT-ok és proxy szerverek figyelik az aktív socket párokat és fenntartanak socket interfészeket
- Továbbá: ütemezéshez (WFQ), QoS támogatáshoz routerekben a csomagfolyamokat a socket párokkal lehet azonosítani

Megoldás: Egyszerű (raw) socket

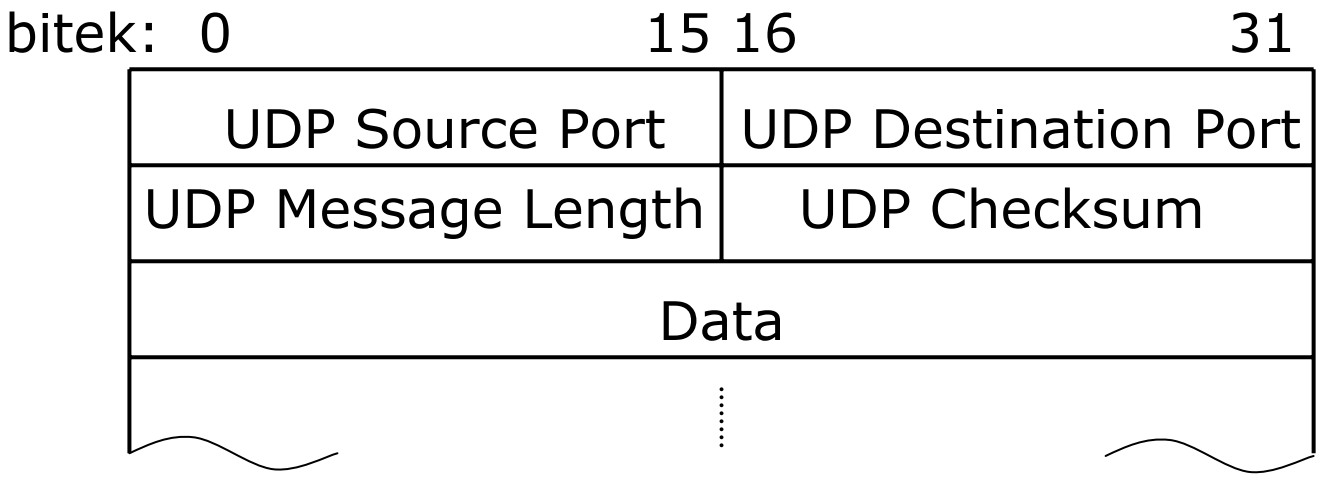
- **Közvetlenül** az alkalmazásnak továbbítja a csomagot a **fejléccel**
  - Nincs TCP/IP feldolgozás, alkalmazás látja el fejléccel/veszi le a fejlécet
  - Pl. routerekben ICMP-hez

- Windows XP-ben 2001-ben jelent meg: biztonsági aggályok (Unixban régóta)
  - Félelem **TCP reset támadástól**: „lelövi” a TCP kapcsolatot
    - +:gyanús kapcsolatok megszakítása
    - -: harmadik fél beékelődve megszakítja a kapcsolatot
    - +/- ? Peer-to-peer forgalom szűrése (Comcast-NNSquad eset)

UDP (User Datagram Protocol)

- Nincs kapcsolatfelépítés, kézfogás
- Semmi garancia:
  - a csomagok sorrendjére
  - adatintegritására
  - elvesztésének detekciójára
- A megbízható átvitel garantálását az alkalmazási rétegre bízza
- *Unreliable* Datagram Protocol ☺
- Multiplexálást és opcionálisan adatintegritás ellenőrzést nyújt

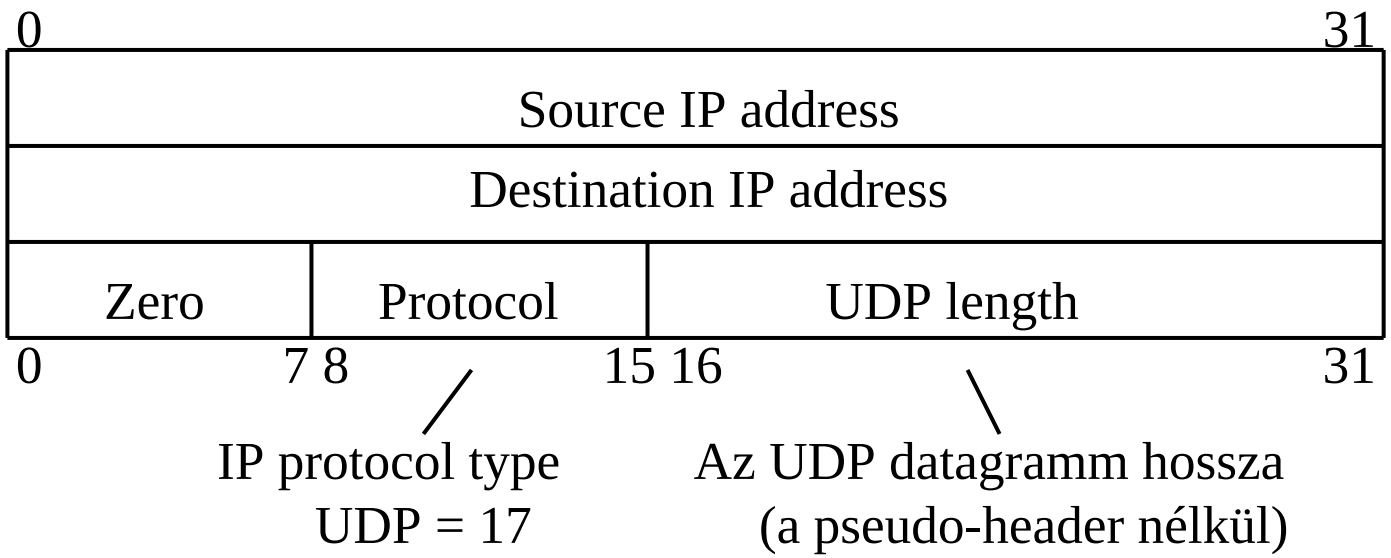
UDP – User Datagram Protocol (2)



- **Source port** opcionális
- **Length**: oktettben, max (65 535-8-20)
- **Checksum**: opcionális (nincs: 0, IPv6-nál már nem opcionális)
- Az UDP checksum az egyetlen lehetőség annak ellenőrzésére, hogy a datagram helyesen érkezett-e meg
  - adatra és fejlécre!
  - az IP-csomag ellenőrzése csak a fejrészt fogja át

UDP – User Datagram Protocol (3)

- A checksum számítási elve
  - 1-es komplemens 16 bites szavakra
- Tartalmazza az IP-címeket is
  - annak ellenőrzésére, hogy a datagramm elérte a helyes címzettet, nemcsak a helyes portot
- „Pseudo-header” hozzáadásával (IPv4-re):



UDP – User Datagram Protocol
Összefoglalás: funkcionalitás és a költsége

- Portkezelés
  - a különböző alkalmazások/folyamatok megkülönböztethetők
- Több alkalmazás egyidejű kezelése
  - port-hozzárendeléssel és multiplexeléssel/demultiplexeléssel
- Hibajelzés az UDP datagram tartalmára és az IP csomag további részeire
- A fentiek költsége:
  minimum 8 oktettnyi overhead

UDP alkalmazása

- Broadcast, multicast (TCP nem képes rá)
- Média: streaming, valós idejű játékok, VoIP, IPTV
- Gyors és rövid lekérdezések:
  - DNS
  - DHCP
  - RIP
- Tipikusan a hálózati forgalom pár százaléka
  - De növekszik a részaránya és nincs torlódás szabályozás: aggályok!
  - Datagram Congestion Control Protocol (DCCP)

TCP –Transmission Control Protocol
Fő jellemzői

- **Célkitűzés:** megbízható szállítási szolgáltatás nyújtása az IP nem megbízható datagram-szolgáltatásán

- **Jellemzői:**
  - ***Virtuális összeköttetések:*** összeköttetés épül fel és marad fenn a kommunikáció tartamára
  - ***Stream–típusú szolgáltatás:*** byte- (oktett-) streamek sorrendhelyes átvitele
  - ***Strukturálatlan stream:*** nincsenek határolók a streamen belül
  - ***Pufferelt átvitel:*** a streamből a datagramm megtöltéséhez szükséges mennyiséget várja össze
  - ***Duplex kapcsolatok:*** két független stream
  - ***Vezérlő információk küldése:*** az ellenkező irányban folyó streambe ágyazva (**piggybacking**)

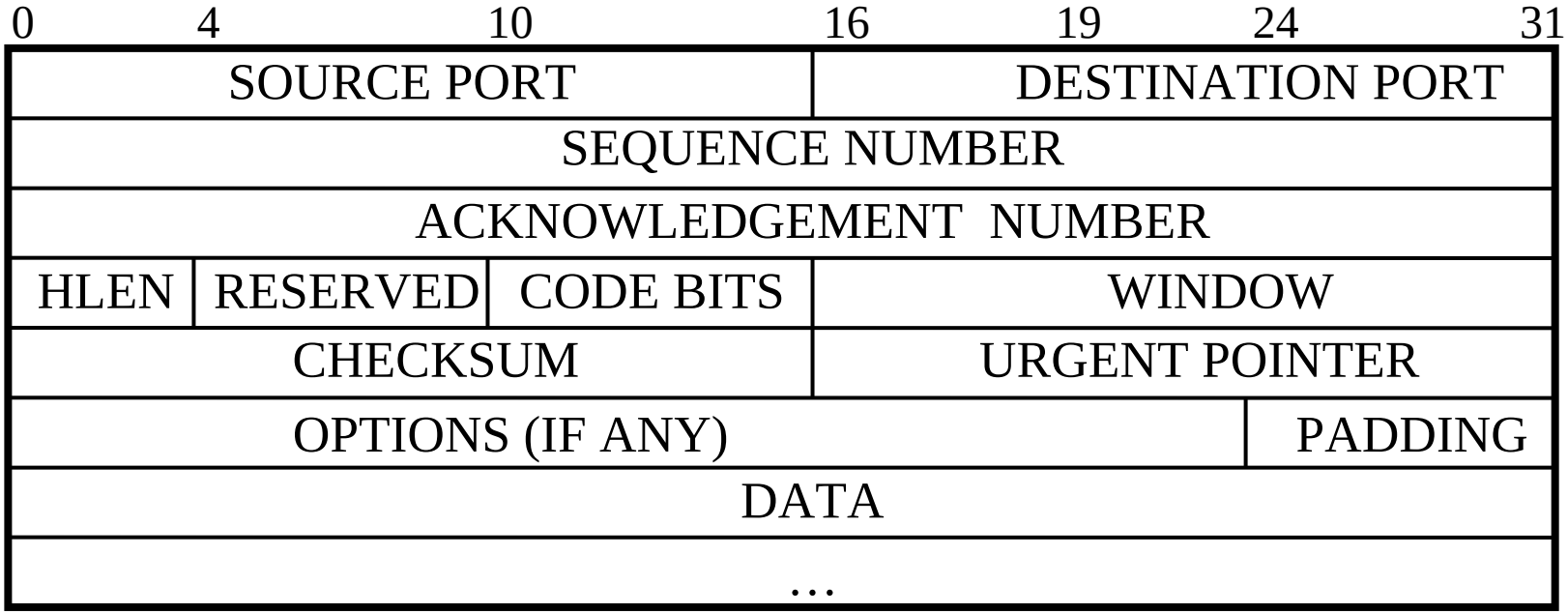
TCP – Transmission Control Protocol
Miről lesz szó?

- **Adategység**: szegmens; szegmensstruktúra
- **Megbízható átvitel** sorszámozás és pozitív nyugtázás segítségével
- Összeköttetés-alapú kommunikáció: **kapcsolatfelépítés és -lebontás**
- **Forgalomszabályozás** (flow control) ablakmechanizmus segítségével
- **Torlódásvezérlés** (congestion control)

TCP kialakulása

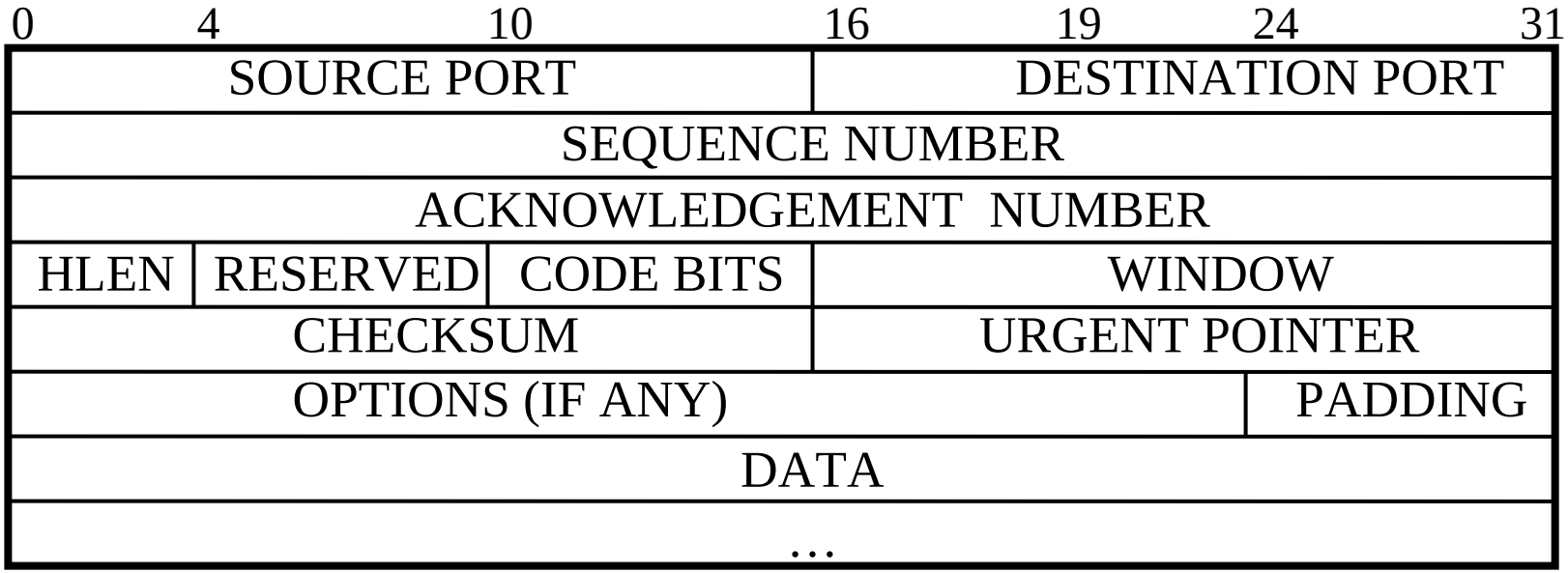
- 1974: "*A Protocol for Packet Network Interconnection.*" Vint Cerf és Bob Kahn
- Itt vezetik be a *Transmission Control Program*-ot, ez válik szét később TCP-re és IP-re
- Eleinte Internet Protocol Suite, majd TCP/IP elnevezés

A PDU – Protocol Data Unit (a TCP-ben: „segment”) struktúrája



- **Sequence no**.: a szegmensben levő adat első byte-jának pozíciója a küldő byte stream-jében
- **Ack no**.: annak a byte-nak a sorszáma, amelyet a forrás legközelebb vár

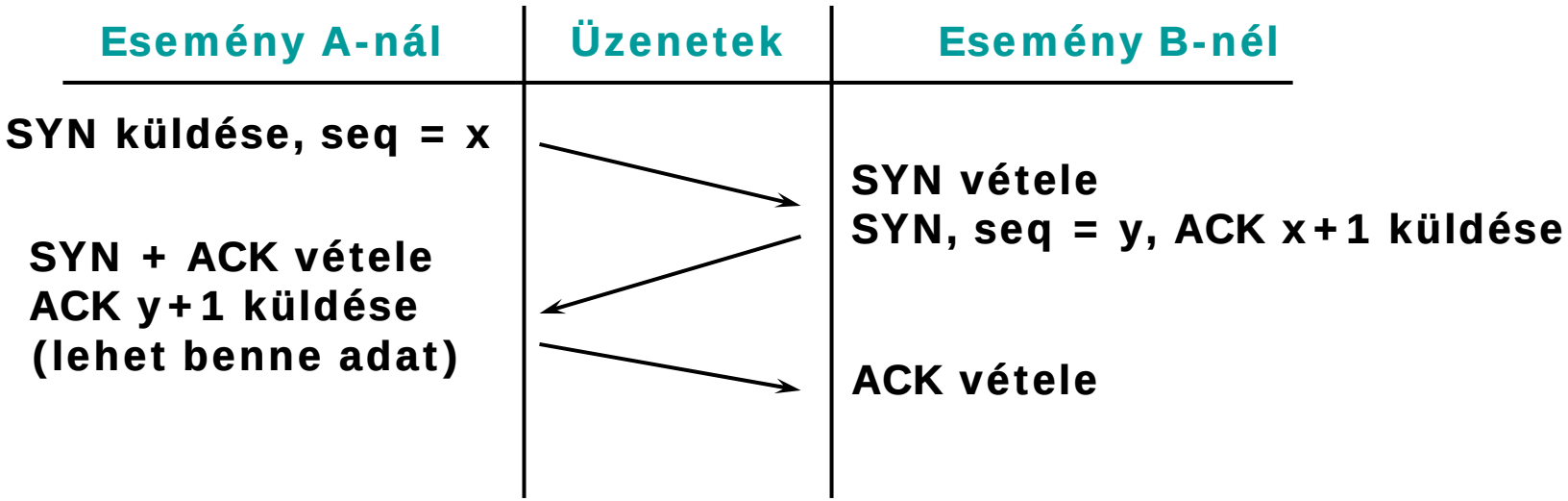
TCP – Transmission Control Protocol
Szegmensformátum (folyt.)



- **HLEN**: fejléc mérete, minimum 20 byte, max 60 byte
- **Code bits** (flags): URG, PSH, ACK, RST, SYN, FIN bitek
  a kapcsolat kezeléséhez használt jelzőbitek
- **Window**: a küldő ismertté teszi a **vételi pufferének méretét**
- **Checksum**: mint az UDP-ben (pszeudorandom)
- **Urgent pointer**: ha URG=1, a szegmens „urgent” részt tartalmaz, ilyenkor a végére mutat (pl. jelszóküldés)

Hívásfelépítés a TCP-ben

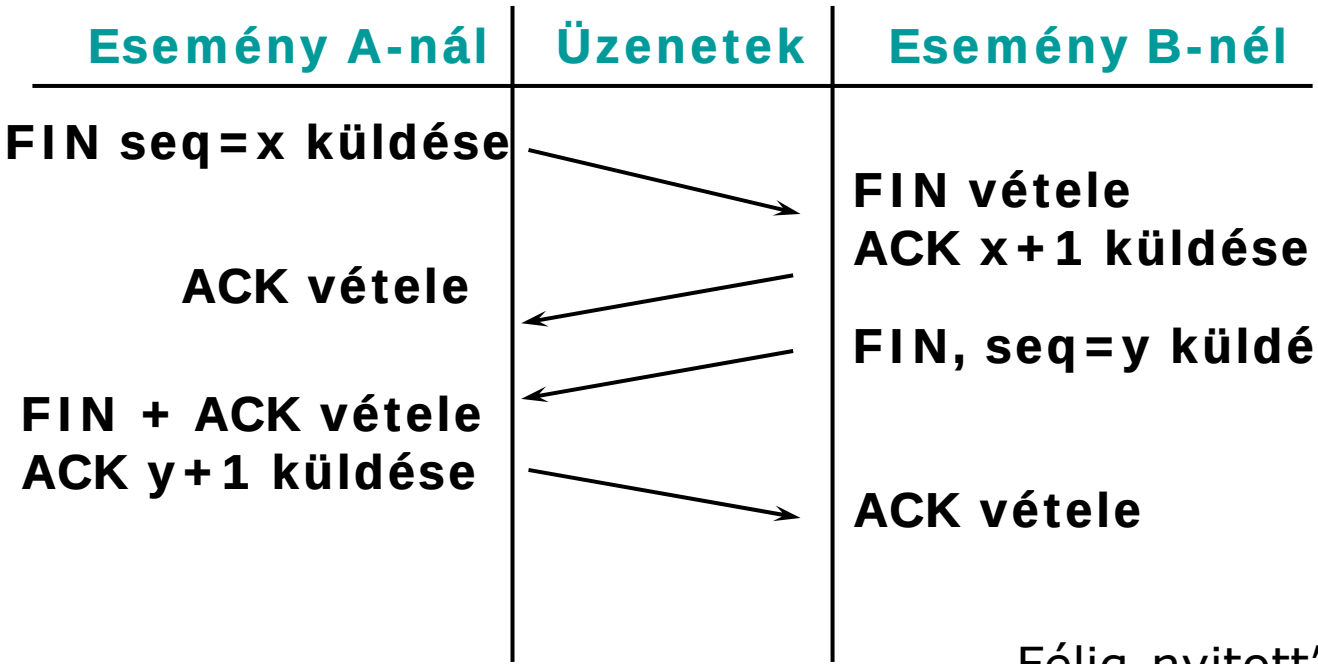
„3-way handshake” eljárás (3 utas kézfogás)



Elején véletlenszerű sorszám: a TCP sorszám predikciós támadás elkerülése

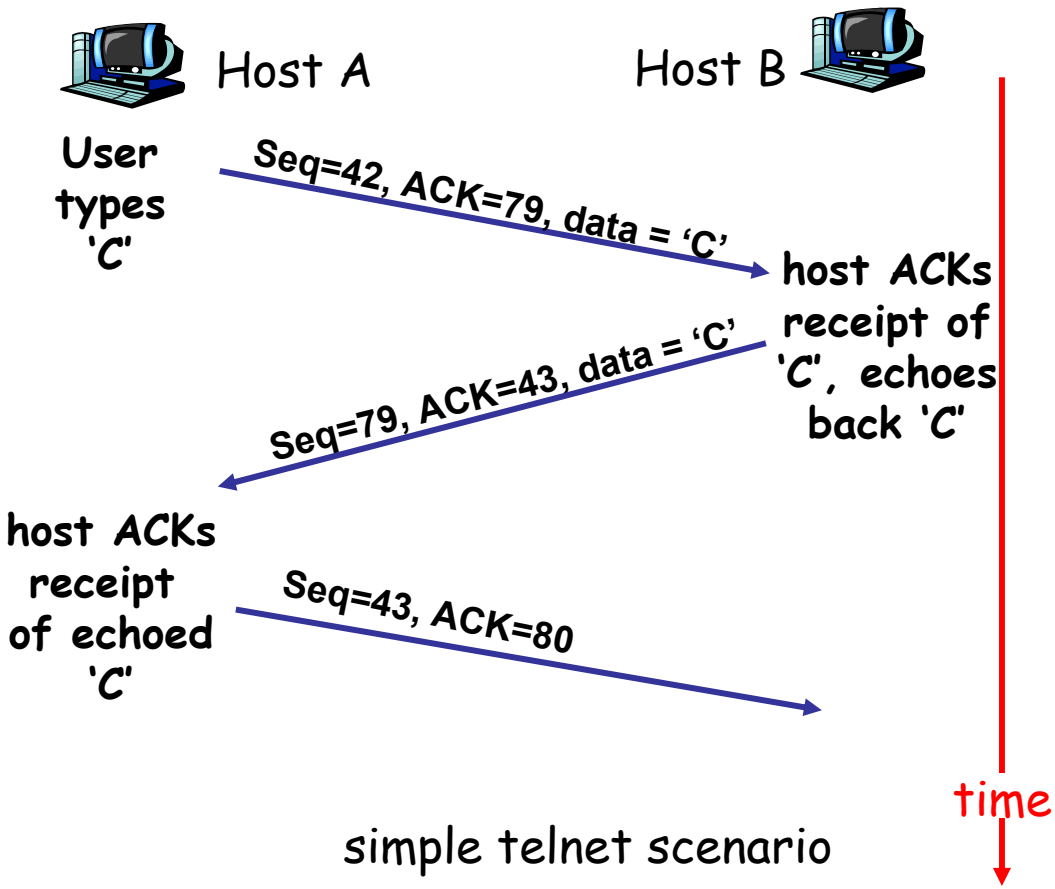
Híváslebontás a TCP-ben

„Modified 3-way handshake” eljárás



„Félig-nyitott” kapcsolat elkerülése

Sorszámok és nyugtaszámok használata a TCP-ben



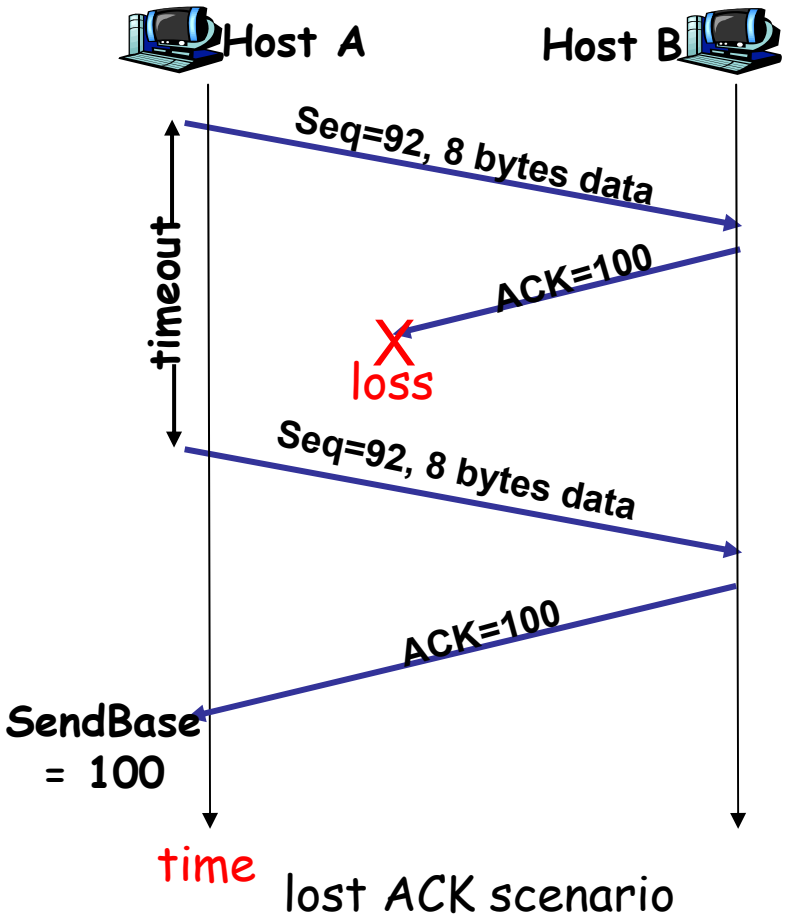
Várakozás nyugtázásra

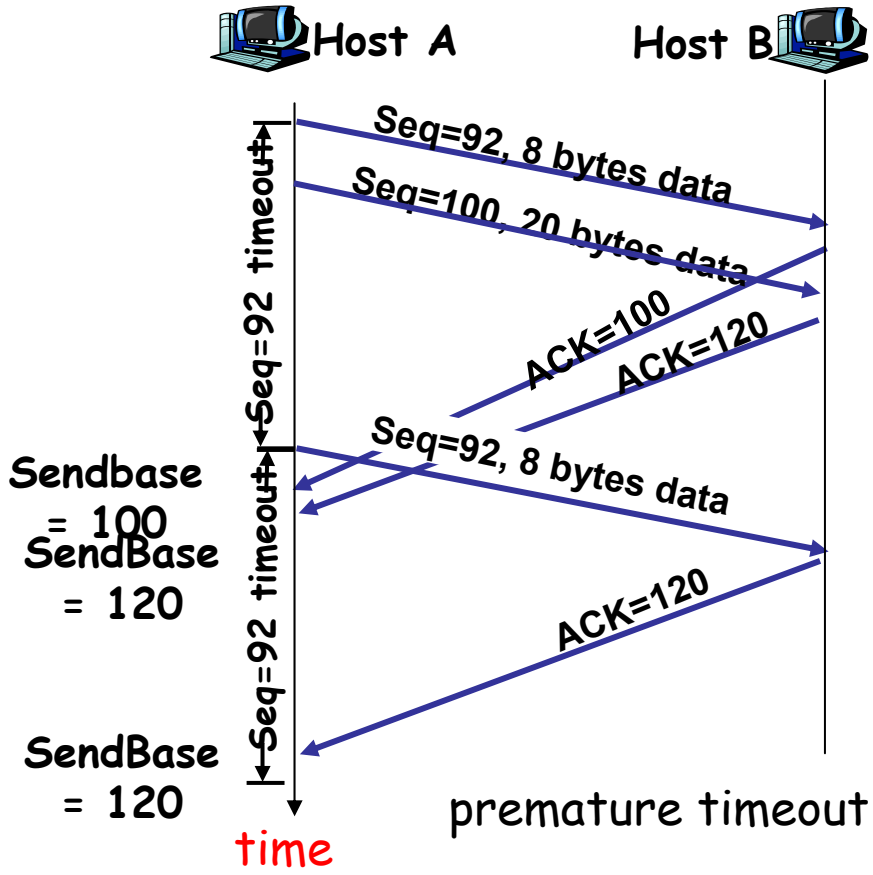
- Várakozás ACK-ra: **time–out**
- Mekkorára válasszuk a time–out-ot?
- Probléma a túl kicsivel és a túl naggyal

- Megoldás:
  a teljes terjedési időhöz
  (RTT - round-trip time) igazítani, **adaptívvá** tenni

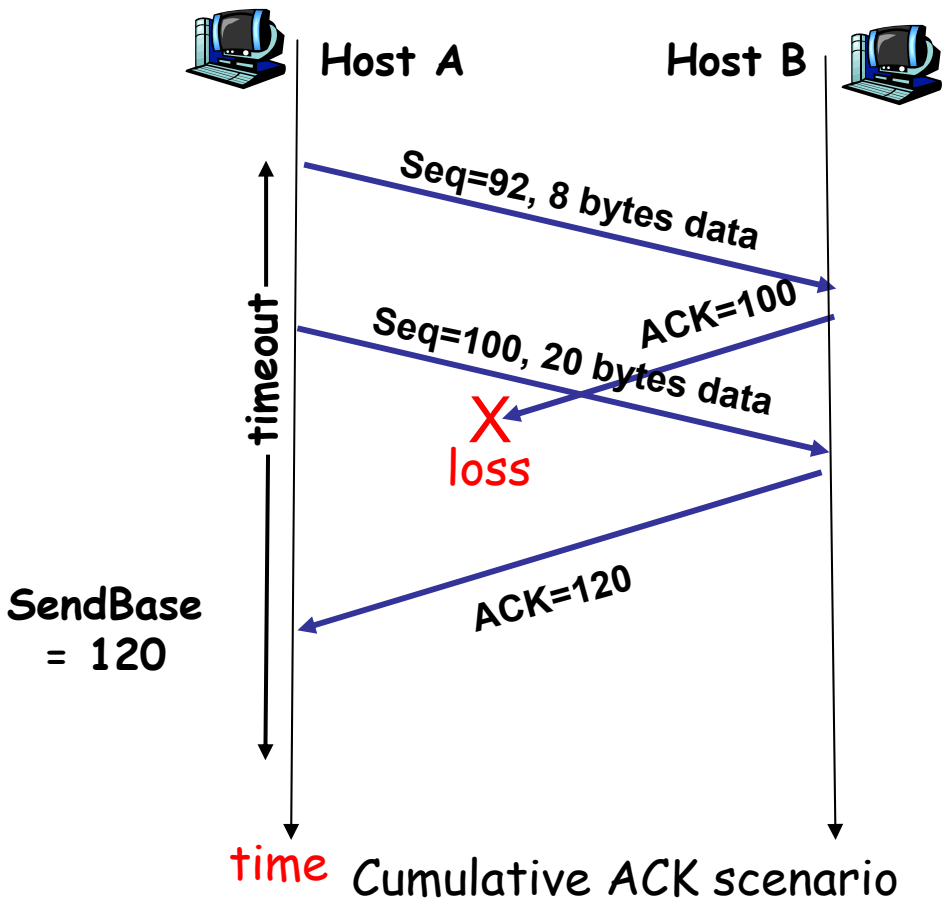
- Szabályok arra, hogy mit tegyünk, ha nem jön ACK a time-out alatt

Újraküldési esetek a TCP-nél: elveszett nyugta és korai timeout





Újraküldési esetek a TCP-nél: összevont nyugta



Cumulative ACK scenario

Gond összevont nyugtával

- Pl. 1000 bájt kerül elküldésre 10 szegmensben
  - Ha elveszik az első szegmens, a fogadó nem tudja visszajelezni, hogy 100-999 megjött, de 0-99 nem
  - Újra elküldi a teljes 1000 bájtot

- Megoldás: **szelektív nyugtázás**
  - SACK-ban meg tudja mondani, hogy 100-999 megjött sikeresen
  - Opcionális fejrészmezőben
  - Népszerű, minden TCP implementációban

Gond sorrend keveredéssel

- Sorrendkeveredés miatt elveszett csomagnak hiszi a küldő ->újraküldés->forrás visszafogás

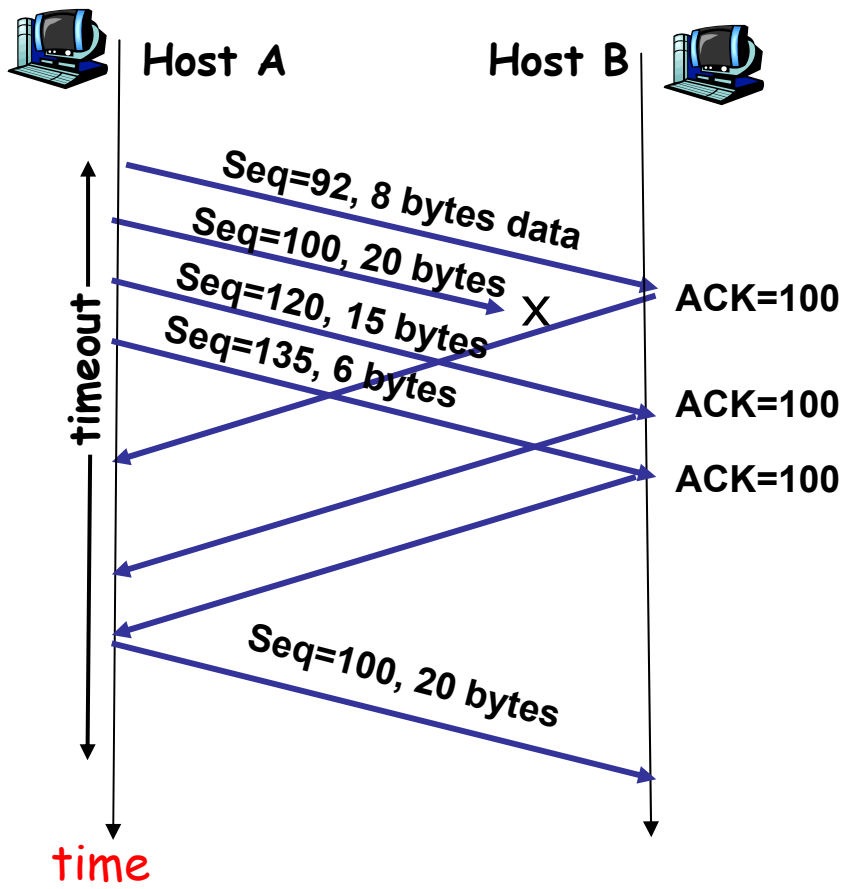
- Megoldás: **D-SACK (duplikált nyugta)**
  - Fogadó szól, hogy az újraküldött csomag duplikátum

  ->visszagyorsul a forrás

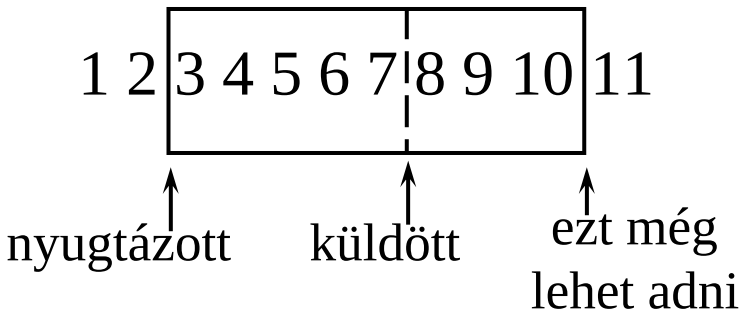
Fast retransmit (1)

- A time-out idő gyakran túl hosszú:
  - nagy késleltetés mielőtt az elveszett csomagot az adó újra tudná küldeni.
- De hogy értesülhetne az elveszett csomagról előbb, mintsem hogy letelt volna a timeout?
  - Az elveszett szegmensekre utalhatnak a duplikált ACK-ok
  - Ha a vevő hézagot vesz észre a vett szegmensek sorozatában (elveszett csomag) akkor újból lenyugtázza a megelőző helyesen vett szegmenst.
  - Több egymást követő duplikált ACK érkezhet.
- Fast retransmit szabály: ha az **adó 3 egymást követő ACK-t kap** ugyanarra a szegmensre, feltételezi, hogy az azt követő szegmens elveszett és
  - **újraküldi azt még mielőtt lejárna a timeout**.

Fast retransmit (2)

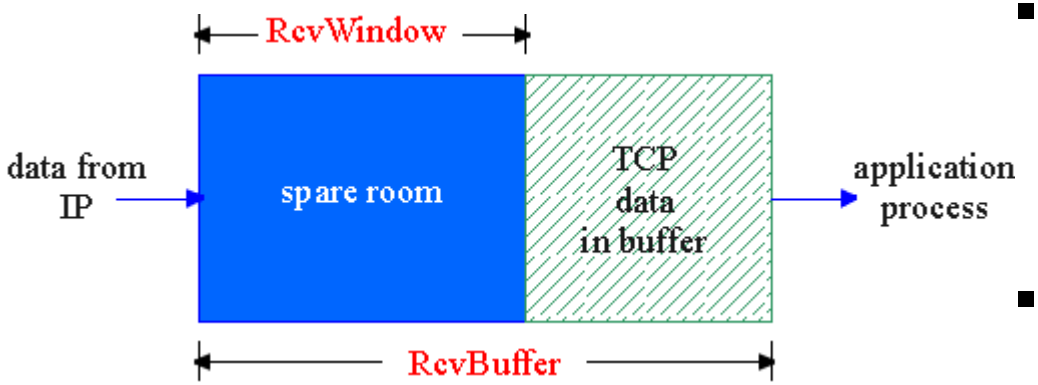


Flow control: a „sliding window” módszer elve



- Csúszóablakos („sliding window”) mechanizmus
  - az ablak mérete megadja a „kintlevő”, nyugtázatlan csomagok max. számát. (Pl.: w=8)
- A TCP-ben: az ablak-mechanizmus oktetteken működik

TCP Flow control: hogy működik?



- Spare room:

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

- A vevő közli a szabad helyének a méretét (**RcvWindow**) a küldött szegmensben
- Az Adó legfeljebb **RcvWindow** mennyiségű nyugtázatlan adatot küld

MSS (Maximum Segment Size)

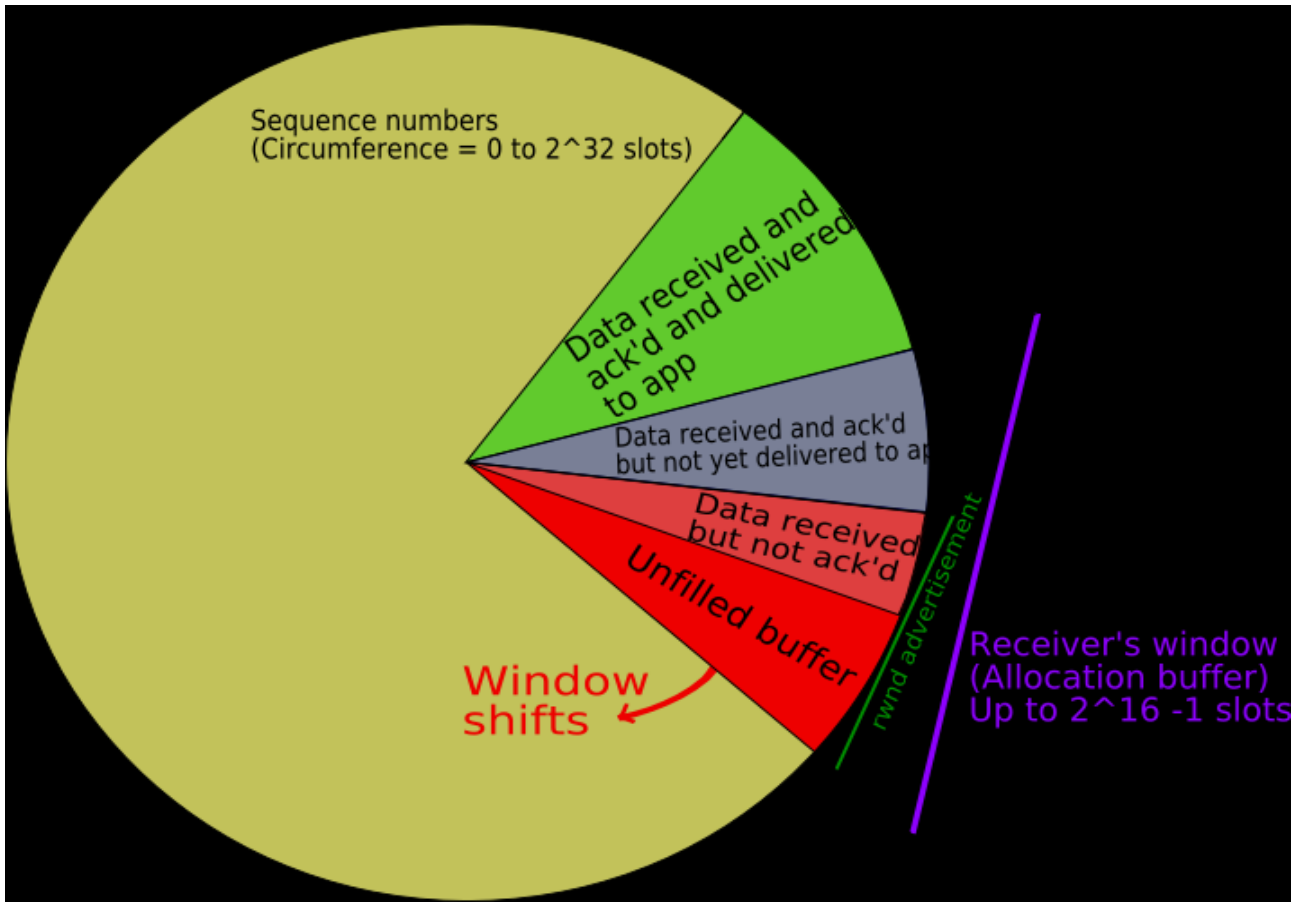
- A legnagyobb adatméret bátjban, amit a TCP hajlandó küldeni egy szegmensben
- Össze kell egyeztetni az adatkapcsolati réteg MTU-jával
  - elkerülendő a tördelést
- TCP kapcsolatfelépítésnél kell egyeztetni, **MSS opció a fejlécben**
- TCP adó használhat Path MTU discovery-t is: dinamikus MSS változtatás

Csúszóablakos problémák

I. Ha a vevő **nullás csúszóablak** méretet hirdet: adó leáll a küldéssel
- Ha **elveszik a vevő csomagja az új csúszóablak méretről**: adó vár hiába
- Megoldás: adó egy **timert** indít, lejárta után felkéri a vevőt, hogy küldjön ACK-ot az új ablakméretről

II. **Buta ablak jelenség**
- Ha a vevő oldalon kicsi (akár 1 byte) szabadul fel, lehet 1 byte az ablak
- A küldő 1 byte-ot küld, a vevő megint 1 byte-tal nyit
- Erőforráspocsékolás: kisebb az adat mint a fejléc!
- Megoldás:
  - A vevő nem nyitja az ablakot, csak akkor, **ha MSS nagyságrendűt nyithat**
- A küldő nem küld, hacsak nem
  - MSS-nyit küldhet
  - Mindent küldhet, amit az alkalmazás kért

A TCP „óra”



Torlódásvezérlés a TCP-ben

- A torlódásvezérlésről általában:
  - Azok a módszerek, amelyekkel linkek, csomópontok időszakos túlterheltségét megkíséreljük megszüntetni.
- Két fő módszer:
  - „Hálózat által segített” (**network assisted**) torlódásvezérlés
    - A hálózati elemek szolgáltatnak információt a túlterhelésről az adónak
  - Végpontok közötti (**end-to-end**) torlódásvezérlés
    - nincs visszacsatolás a hálózatból, a végpont következtet arra, hogy torlódás léphetett fel
- A TCP az utóbbit csinálja!

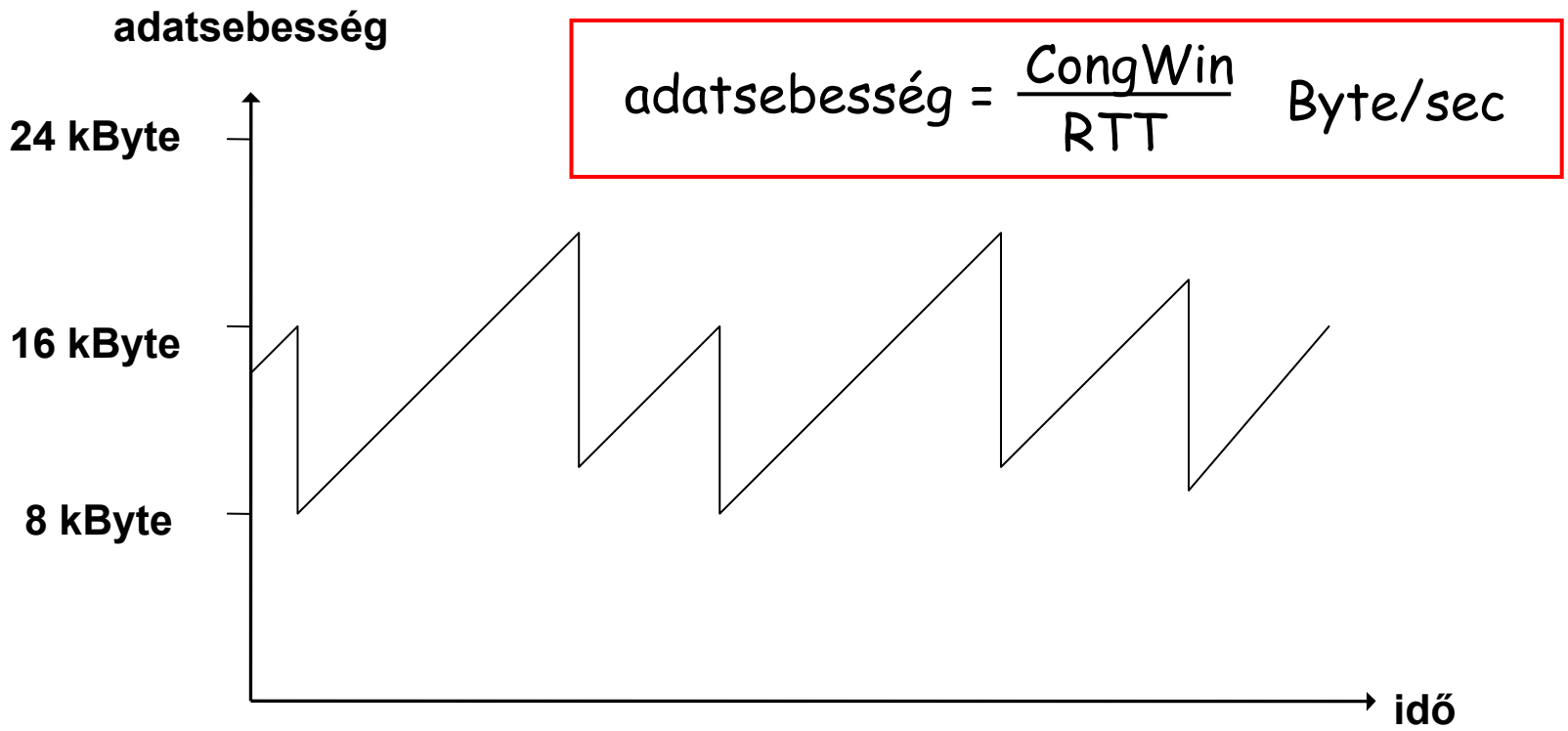
Torlódásvezérlés a TCP-ben

- Módszer:
  - **növeljük az adatsebességet**, ha úgy érzékeljük, van elég átbocsátóképesség
  - amíg **torlódásra utaló jeleket** nem tapasztalunk, ha igen, csökkentsük.

- Hogyan:
  - congestion window, **CongWin**
  - növeljük a CongWin-t minden RTT alatt MSS-sel, amíg vesztést nem érzékelünk
  - csökkentsük a CongWin-t felére minden vesztéskor
  - = **Additive increase – multiplicative decrease (AIMD)** módszer

- Hogyan érzékeljük a torlódást (vesztést)?
  - timeout letelt, nem jött nyugta
  - többszörös nyugta érkezett ugyanarra a szegmensre

Az adatsebesség alakulása AIMD esetén



$$\text{adatsebesség} = \frac{\text{CongWin}}{\text{RTT}} \quad \text{Byte/sec}$$

TCP torlódásvezérlés: további részletek

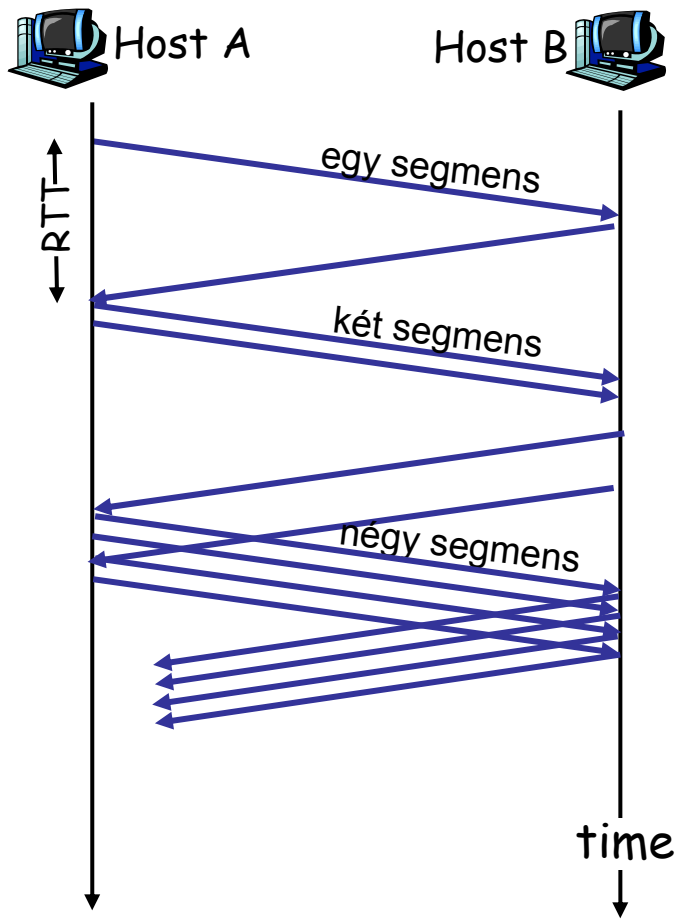
- Az AIMD kiegészítései:

- „Slow Start”
  - az összeköttetés kezdetén a sebesség gyors (exponenciális) növelése az első vesztésig, utána AIMD
  - Gyors növelés: minden ACK után kétszerezzük

- Eltérő viselkedés timeout és többszörös (3-szoros) nyugták esetén

TCP Slow Start

- Az elején: **`CongWin`** = 1 MSS
  - Példa: MSS = 500 byte & RTT = 200 msec
  - kezdeti sebesség = 20 kbps
- Mivel a rendelkezésre álló átb. képesség >> CongWin/RTT lehet,
  - célszerű gyorsan elérni, ezért
  - induláskor exponenciálisan növelni az első vesztésig



Csomagvesztés

- 3 duplikált ACK után:
  - **CongWin** felére csökken
  - Utána lineáris növekedés
    - torlódás elkerülés
- De timer lejárta esetén:
  - **CongWin** 1 MSS lesz;
  - Utána exponenciálisan nő,
  - Egy korlátig, majd onnan lineáris

Filozófia:

- timer lejárta rosszabb torlódási helyzetet jelez, mint a 3 duplikált ACK

Összefoglaló: TCP torlódás szabályozás

- Ha a **CongWin** egy korlát alatt van, adó slow-start fázisban, exponenciális ablak növelés
- Ha **CongWin** a korlát felett, az adó torlódás elkerülési fázisban, lineáris ablak növelés
- Ha 3 duplikált ACK, a korlát **CongWin/2** lesz és a **CongWin** pedig a korlát
- Ha timer lejárt, a korlát **CongWin/2** lesz, míg a **CongWin** 1 MSS lesz

TCP sender congestion control *

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to “Congestion Avoidance”	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP átbocsátóképessége

- TCP átlagos átbocsátó képessége az ablak méret és RTT függvényében?
  - Nem foglalkozunk slow-start-al

- *W* az ablakméret a csomagvesztés pillanatában
- Csomagvesztés előtt az átbocsátás *W/RTT*
- Vesztés után az ablak *W/2* lesz, átbocsátás *W/2RTT*.
- Átlagos átbocsátás: *.75 W/RTT*

TCP és UDP: összefoglalás

- Mindkettő host layer/transport layer protokoll
- Mindkettő portokat kezel
  - multiplexelés/demultiplexelés
  - ezáltal interface nyújtása az alkalmazói folyamatok felé
- Az UDP összeköttetés-mentes, best effort szolgáltatás
  - nem garantál célba juttatást, csak hibajelzést nyújt
  - gyorsan célba juttat
- A TCP összeköttetés-alapú, megbízható transzport szolgáltatás
  - sorrendhelyes, hibamentes szállítást nyújt
  - ára: késleltetés

Néhány gyakori alkalmazás és az használt transzportprotokollok

Alkalmazás	*Alkalmazási rétegbeli protokoll*	*Használt transzport-protokoll*
E-mail	SMTP	TCP
Távoli elérés	Telnet	TCP
Web-elérés	HTTP	TCP
File-átvitel	FTP	TCP
Routing	RIP	UDP
Hálózatmenedzsment	SNMP	UDP, TCP
VoIP, média-streaming	Többnyire nem szabványos	UDP

Kérdések?

KÖSZÖNÖM A FIGYELMET!

Dr. Simon Vilmos
adjunktus
BME Hálózati Rendszerek és Szolgáltatások Tanszék
svilmos@hit.bme.hu