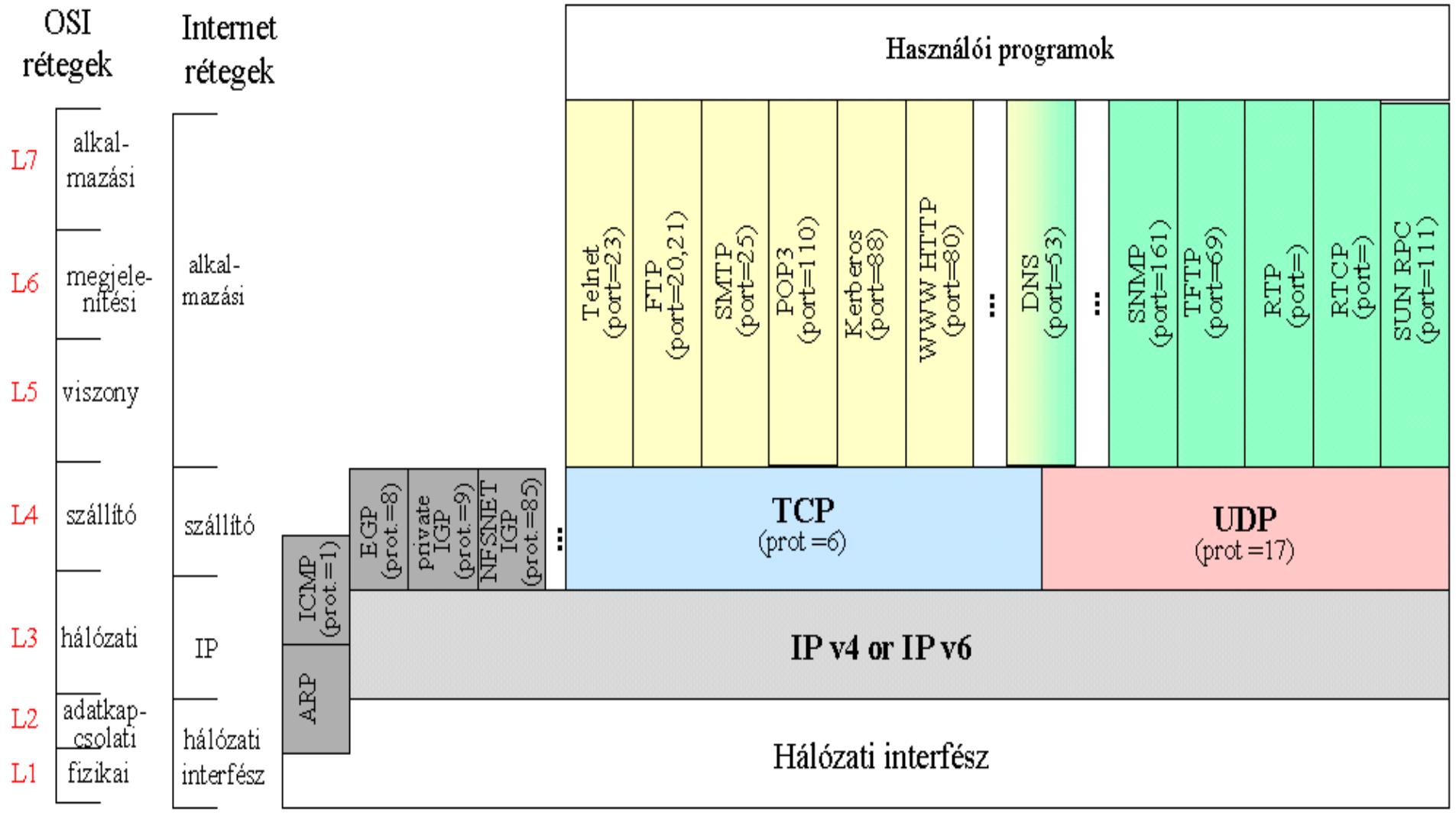
Szállítási réteg (L4)

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

Protokoll stack



Kliens – szerver modellek

- Iteratív szerver
  - Vár, hogy érkezzen egy kliens igény
  - Feldolgozza a kliens igényét
  - Elküldi a választ az igényt küldő kliensnek
  - Ugrás az első lépéshez!
- Konkurrens szerver
  - Vár, hogy érkezzen egy kliens igény
  - Elindít egy új kiszolgálót, hogy kezelje a kliens igényét (új processz, vagy új szál)
    - Ha kész, e szerver működése megáll
  - Ugrás az első lépéshez!

Kliens – szerver modellek

- TCP szerverek általában **konkurrensek**
- UDP szerverek általában **iteratívak**

Iterativitás

- Az UDP Szerver iteratívan dolgozik
  - Általános eset:
  - Egy db well-known port az UDP forgalomnak
  - A sort a küldő végtelennek tekinti
- Ha mégis megtelik a sor:
  - Nincs figyelmeztetés az alkalmazástól a küldőnek
  - Nincs figyelmeztetés a csomageldobásra
  - UDP input sora FIFO

Transzport réteg

- TCP
  - Kapcsolat-orientált
  - Megbízható kapcsolat
  - Automatikus torlódás vezérlés
  - A küldési sebesség automatikus
    - Az alkalmazás nem tudja vezérelni
- UDP
  - Kapcsolat nélküli
  - A sebességet az alkalmazás szabja meg

Transzport réteg - Portok

- Két host között egyszerre több TCP/UDP folyam lehet
- Azonosításra a portokat használják:
  - 16 bites számok a transzport fejlécben
- A forrásnál általában a számítógép választja ki
- A cél általában egy ismert szám
  - HTTP: 80, FTP: 21, SIP: 5060

Transzport réteg – Portok 2

- Egy kapcsolat azonosításához szükséges:
  - Forrás IP, forrás port
  - Cél IP, cél port
- Így egyszerre több kapcsolat lehet ugyanarra a portra – akár ugyanarról a forrás gépről
- Encapsulation – beágyazás

Szállítási réteg - UDP

UDP- User Datagram Protocol

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

User Datagram Protocol

- Egyszerű
- Datagram orientált
- Szállítási réteg beli protokoll
- RFC 768

20 bájt	8 bájt	
IP fejléc	**UDP fejléc**	**UDP adat**

UDP

- Kapcsolat nélküli protokoll
  - Nincs állapot információ
- Nem garantálja a csomagok megérkezését
  - Nem küld újra
  - Nem állít a küldési sebességen torlódás esetén
- Kis fejléc (8 byte)
- Főleg multimédia alkalmazások használják
  - Valós idejű átviteleknél jó választás

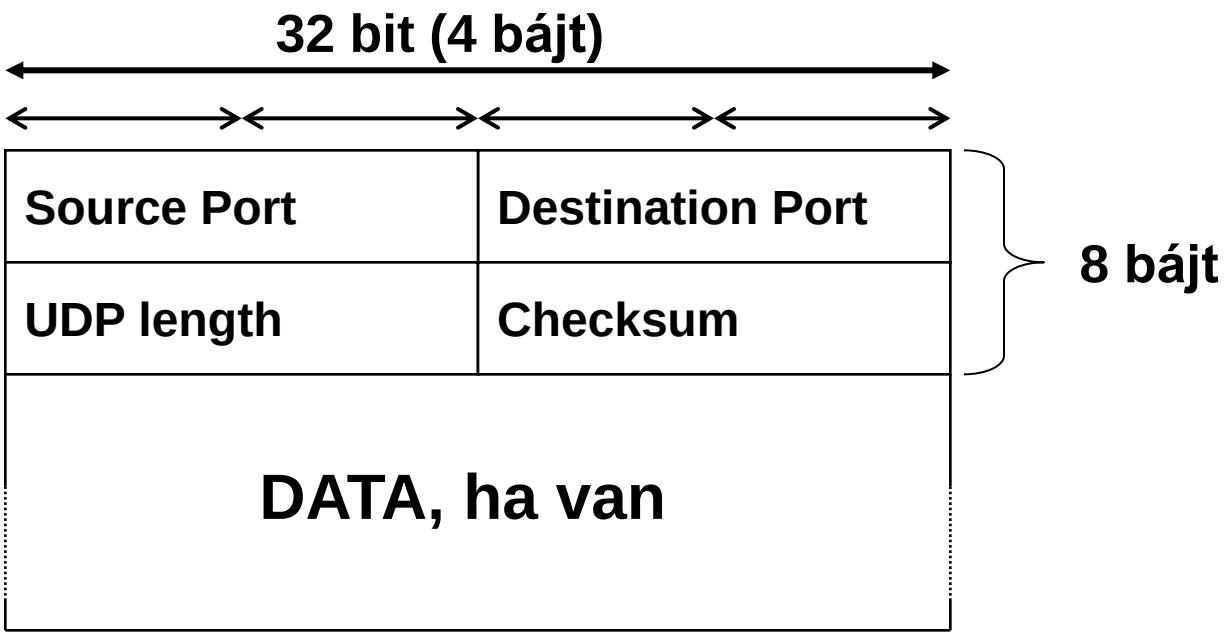
UDP kapcsolat kiépülés

- A kapcsolat kiépülése a portok hozzárendeléséből áll
- A másik számítógép nem kell válaszoljon
  - Ha hiba van ICMP üzenettel jelzi
- A protokoll szegmensekre bontja az adatot
  - Az UDP szegmensek sorszámot kapnak
  - a szegmenseket azonnal küldi

User Datagram Protocol

- Nem megbízható
  - Nincs garancia, hogy a csomag elér a célba
- Nincs folyamvezérlés
- Egy UDP csomag – egy IP datagramm
  - Az alkalmazásnak kell odafigyelnie a helyes csomagméret használatra
  - Fregmentáció!
  - DF bit nem használható
    - A fregmentáció mindig tiltott

UDP csomagformátum



UDP packet structure

Source Port	Destination Port

- Source port (16 bit): azonosítja a küldő alkalmazást
- Destination port (16 bit): azonosítja a fogadó alkalmazást
  - TCP, UDP port számok függetlenek
    - A TCP/UDP demultiplexació az IP protokoll mező alapján!

UDP packet structure

UDP length	Checksum

- UDP length (16 bit): Az UDP fejléc és az UDP adat hossza
  - bájtban
  - Minimum érték: 8 bájt – nincs UDP adat
  - Elméleti max: 65507
- Checksum (16 bit): fejléc és az adatra számítva
  - A 16 bites szavak egyes komplemensű összege
  - „pszeudo fejléc” alapján számolt, bele tartozik:
    - IP csomagból: IP címek, protokoll azonosító mezők
    - Teljes UDP fejléc

UDP Checksum

- Pszeudo fejléc:
  - Kettős ellenőrzése a helyes átvitelnek
    - Szükséges, mert nincs folyamvezérlés
- Nem kötelező (teljes 0 – nincs checksum)
  - Ha a checksum csupa 0
    - Helyettesíti 65535 – egyes komplemens aritmetika
  - Ha a checksum hibás:
    - A csomag csendben (*silently*) eldobásra kerül
      - Nincs hibaüzenet!

Szállítási réteg - TCP

TCP - Transmission Control Protocol

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

Transmission Control Protocol

- Két alkalmazás között nyújt
  - Megbízható végpont-végpont adattovábbítást
  - Kapcsolat-orientált adatfolyam szolgáltatás
  - Folyamvezérlő algoritmus
- Két végponti alkalmazás
  - Az adatközvetítés előtt fel kell építenie a TCP kapcsolatot
- Broadcastingra és multicastingra nem alkalmazható a TCP

Transmission Control Protocol

- A TCP adat IP csomagba enkapszulálva
- A TCP által az IP-hez továbbított adategység neve **szegmens**
- TCP logikai kapcsolatokat használ processz párok között:
  - TCP szegmens tartalmazza a forrás és a cél port számait
- Az IP cím és a megfelelő TCP port számok kombinációját hívjuk a kapcsolat **socket**jének, transzport címének
  - Socket párok

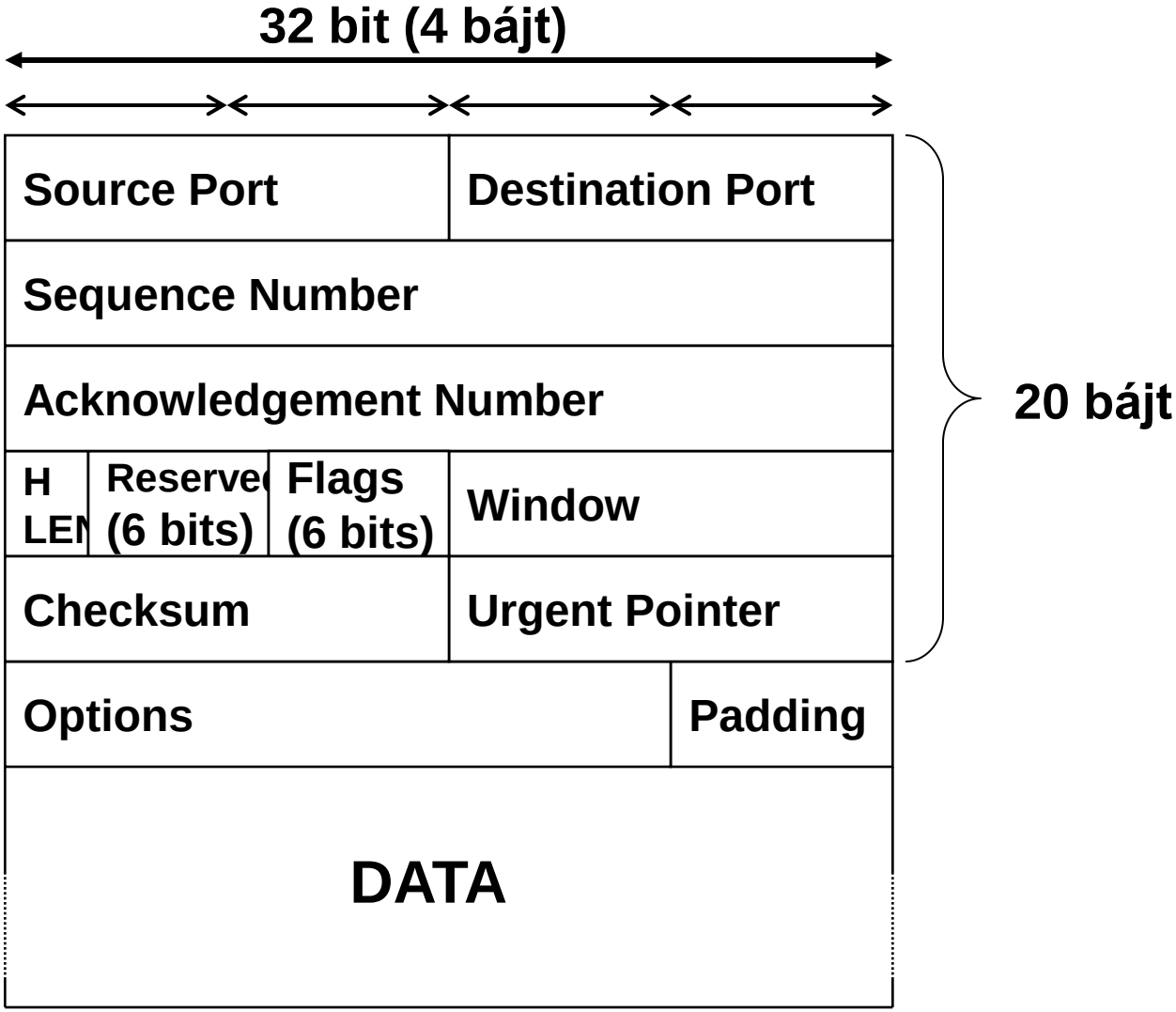
TCP

- ***full duplex*** szolgáltatást nyújt az alkalmazási rétegnek
  - Kétirányú adatátvitel
  - Mindkét végpont sorszámozást végez az adataikon
- ***nincs selective*** ACK
  - ack jelentése: eddig a bájtig (de a küldöttet nem beleértve) sikeres a vétel
- ***nincs negative*** ACK
- ***nem interpretálja*** a bájtfolyamot
  - Továbbadja az alkalmazásnak

Áttekintés - TCP

- Csomag formátum
- Kapcsolat kiépítés
- Csúszó ablakos átvitel
- Torlódás vezérlés
  - Slow Start
  - Fast Retransmit
  - Congestion avoidance

TCP csomagformátum



TCP csomagformátum

Source Port	Destination Port
Sequence Number	
Acknowledgement Number	

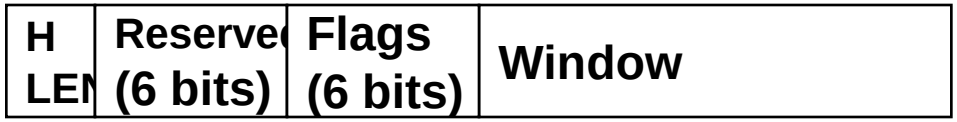
- **Source port** (16 bit):
  - A TCP port száma a küldőnél
- **Destination port** (16 bit):
  - A TCP port száma a fogadónál
- **Sequence number** (32 bit):
  - A bájtfolyam adott szegmensének sorszáma
- **Acknowledgement number** (32 bit):
  - A fogadó által következőként várt szegmens sorszáma

Port számok

- Azonosítják az alkalmazásokat (16 bit)
- well-known port számok (1-1023)
  - szerverek, pl. Telnet 23, FTP 21
- ephemeral port számok (1024-5000)

- Internet Assigned Numbers Authority, IANA

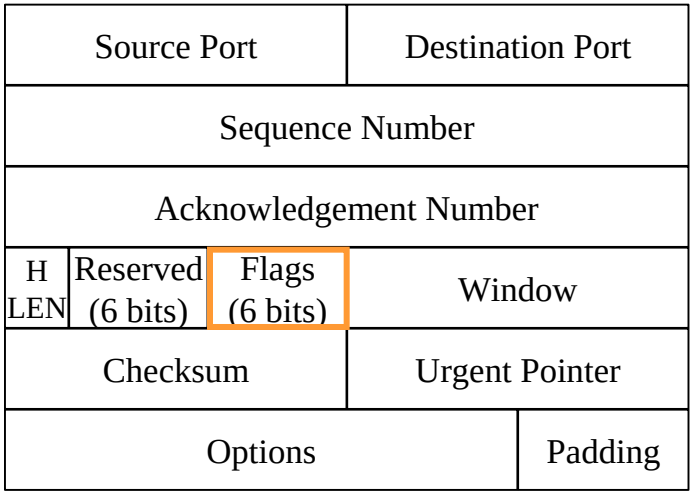
TCP csomagformátum



- **Header Length** (4 bit):
  - A TCP fejléc 32 bites szavainak száma
  - Az options mező vváltozó hossza miatt szükséges
- **Reserved** (6 bits):
  - MBZ
  - Jövőbeni használatra foglalt
- **Flags** (6 bits):
  - 6 flag, melyek szabályozzák a TCP csomag viselkedését
    1. Urgent (URG)
    2. Acknowledgement (ACK)
    3. Push (PSH)
    4. Reset connection (RST)
    5. Synchronous (SYN)
    6. Finish (FIN)

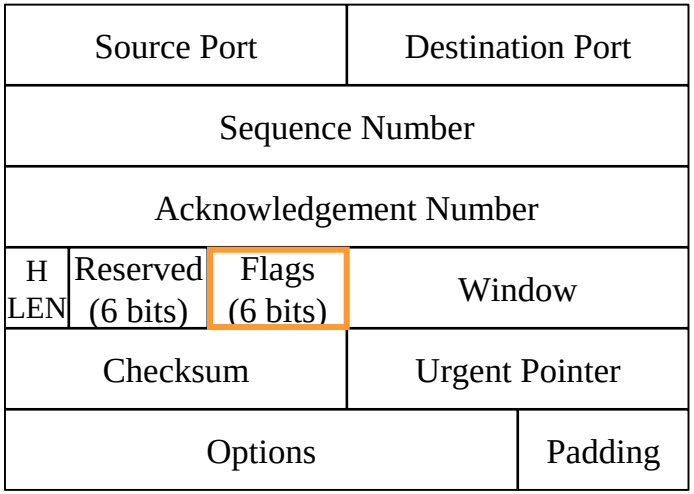
TCP flagek

- Urgent flag (URG)
  - A végpontok üzenhetnek, hogy sürgős adat van az adatfolyamban
- Acknowledgement flag (ACK)
  - Megadja, hogy a nyugtaszám a szegmensben érvényes
- Push flag (PSH)
  - A szegmens adatokat tartalmaz, melyeket az alkalmazásnak kell továbbítani



TCP flagek

- Reset flag (RST)
  - Reset szegmenst küld a TCP
    - ha nem megfelelő portra érkezik kapcsolatkérés
    - Ha az egyik fél meg akarja szakítani a kapcsolatot
- Synchronous flag (SYN)
  - A SYN flag bekapcsolt azokban a szegmensekben, melyek a kapcsolatfelépítéshez szükségesek
- Finish flag (FIN)
  - A végpontok kapcsolat lezárásra használják ezt a flaget



TCP csomagformátum

H LEN	Reserved (6 bits)	Flags (6 bits)	Window
Checksum			Urgent Pointer

- **Window** (16 bit):
  - Az adatfolyam vezérléshez szükséges
  - Megadja, hogy a fogadónak mennyi bájt adat fogadására képes a buffere.
- **Checksum** (16 bit):
  - A TCP fejléc integritásának megőrzésére
  - A checksum pszeudo fejléc alapján számítódik, információkat véve az IP fejlécből is

TCP csomagformátum

Checksum	Urgent Pointer
Options	Padding

- **Urgent Pointer** (16 bit):
  - Ha sürgős adat van a szegmensben ez a pointer mondja meg, hogy hol kezdődik az az adatrészben
- **Options**:
  - A leggyakoribb opció mező az MSS - maximum segment size
    - Megadja a legnagyobb szegmensméretet, melyet a fogadó fogadni szeretne

TCP kapcsolat felépítés és bontás

Budapest University of Technology and Economics

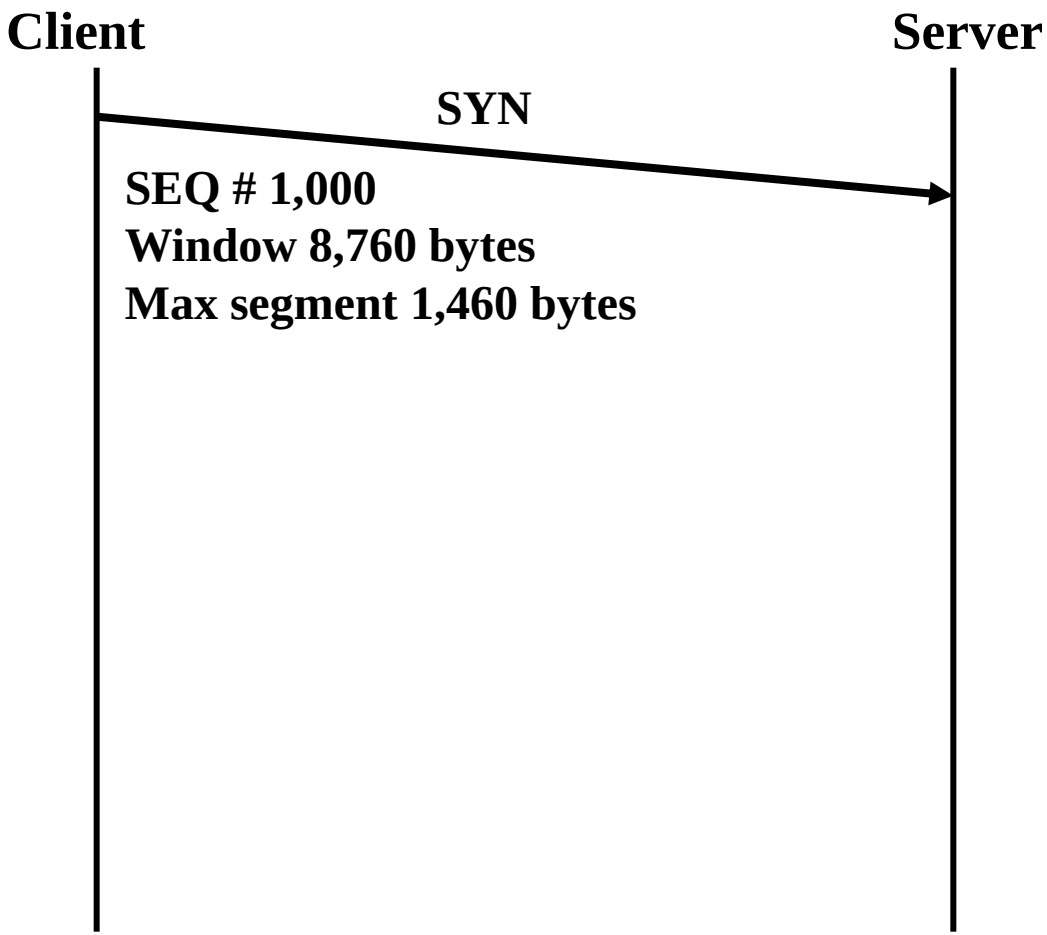
Department of
Telecommunications and Media Informatics

Transmission Control Protocol

- A TCP az adatszegmensek továbbításakor a következőket végzi:
  - Kapcsolat felépítés
  - Ablakméret (Advertised window size), Maximum szegmens méret meghirdetése
  - Adatok továbbítása
  - Nyugták küldése a fogadott szegmensekre
  - Kapcsolat lezárása

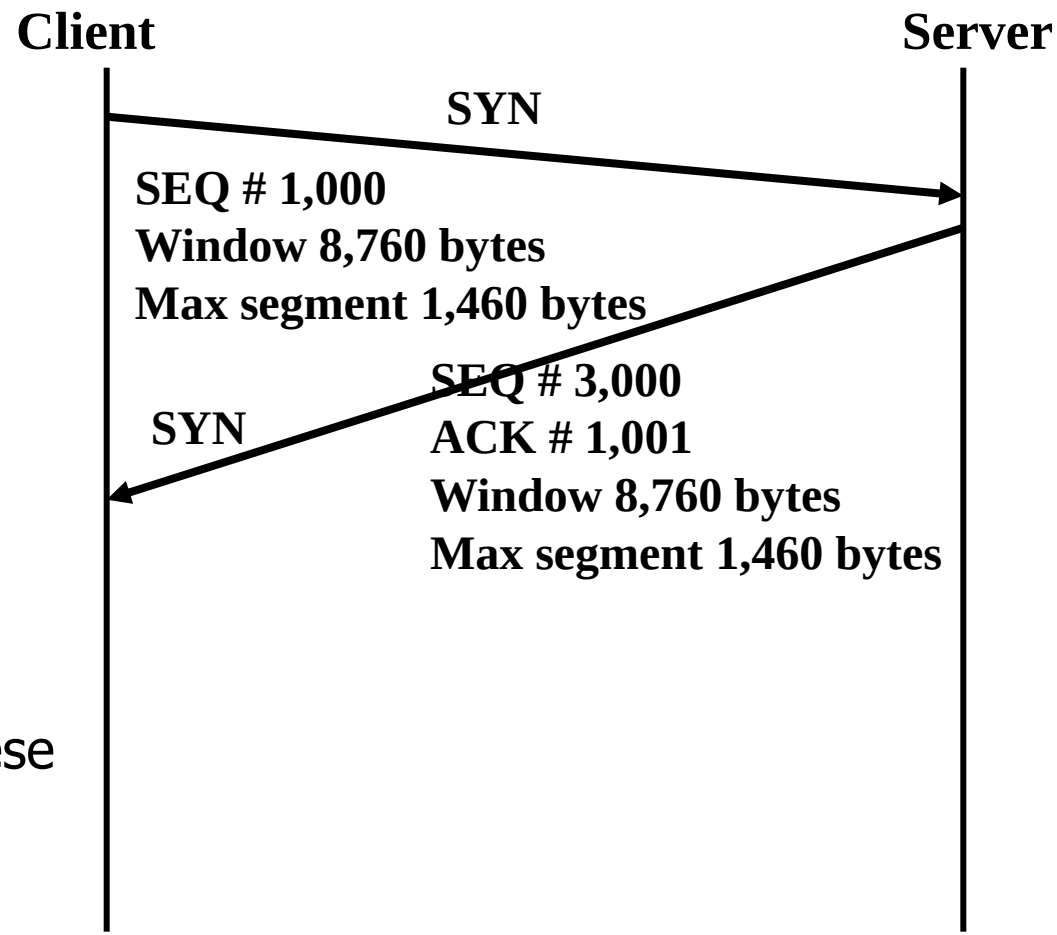
Kapcsolat felépítés

1. Kezdeményező végpont - kliens
   - SYN szegmens küldése
     - Szerve port számának megadása – ahová kapcsolódni szeretne
   - Kezdeti saját, sorszám
     - ISN – initial seq. num.
   - Saját ablakméret hirdetése
   - MSS hirdetése



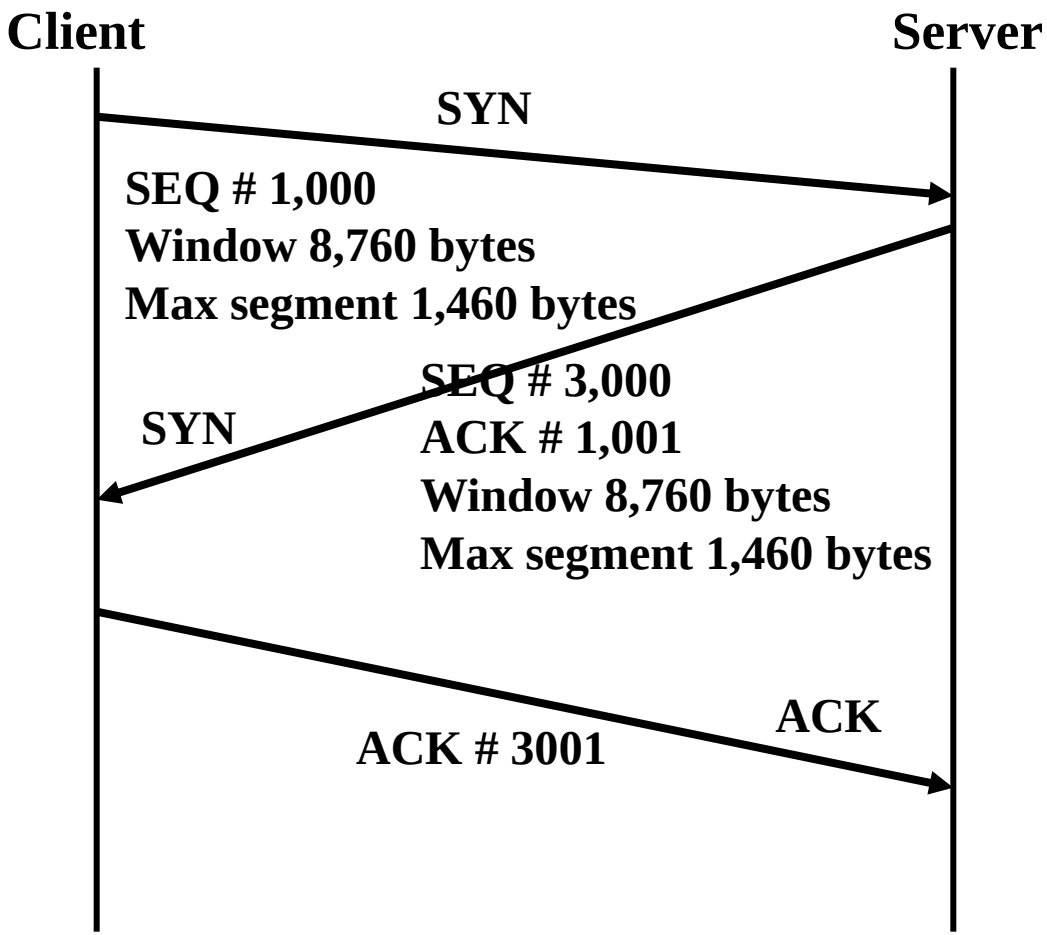
Kapcsolat felépítés 2

2. The szerver válaszol
   - SYN szegmens tartalmazza
     - Szerver ISN
     - Nyugta a kliens szegmensére
       - Várja az 1001-et
   - Saját ablakméret hirdetése
   - MSS hirdetése



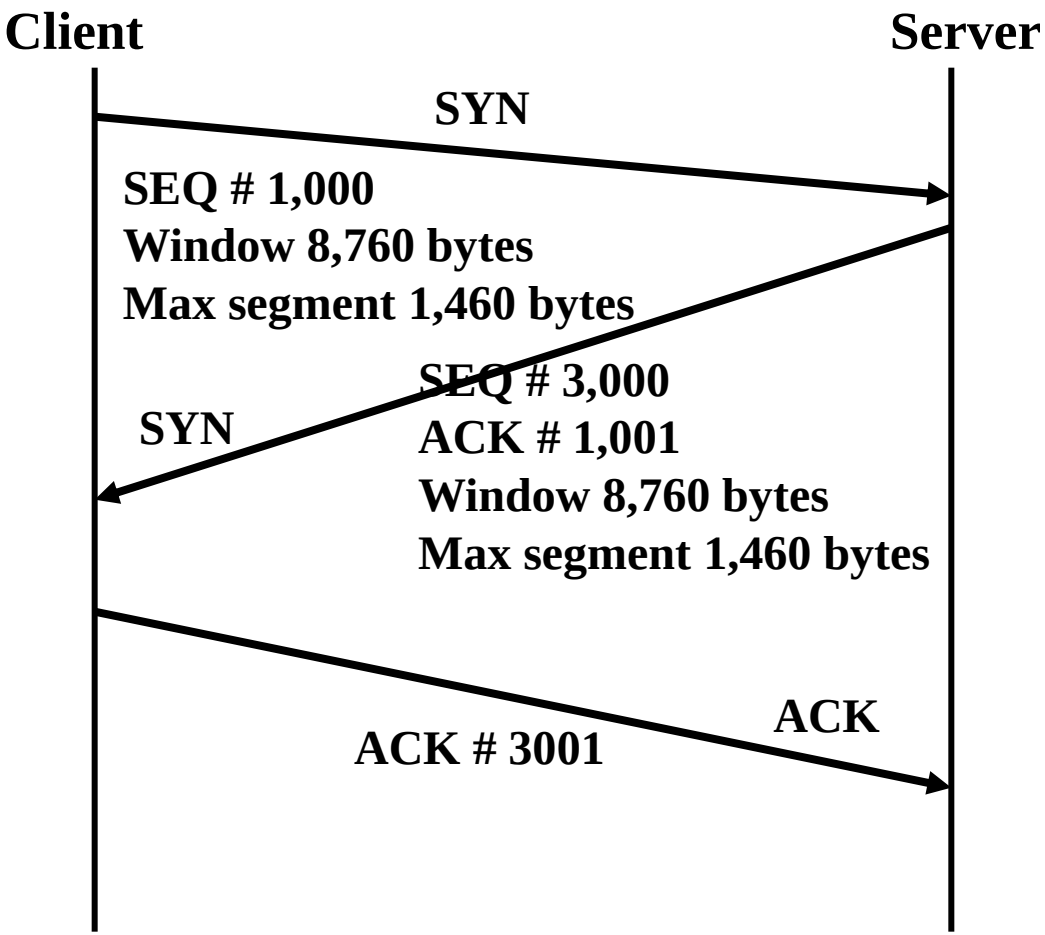
Kapcsolat felépítés 3

3. A kliens nyugtáz
   - Kötelező a kliensnek nyugtát küldeni a szerver SYN szegmensére



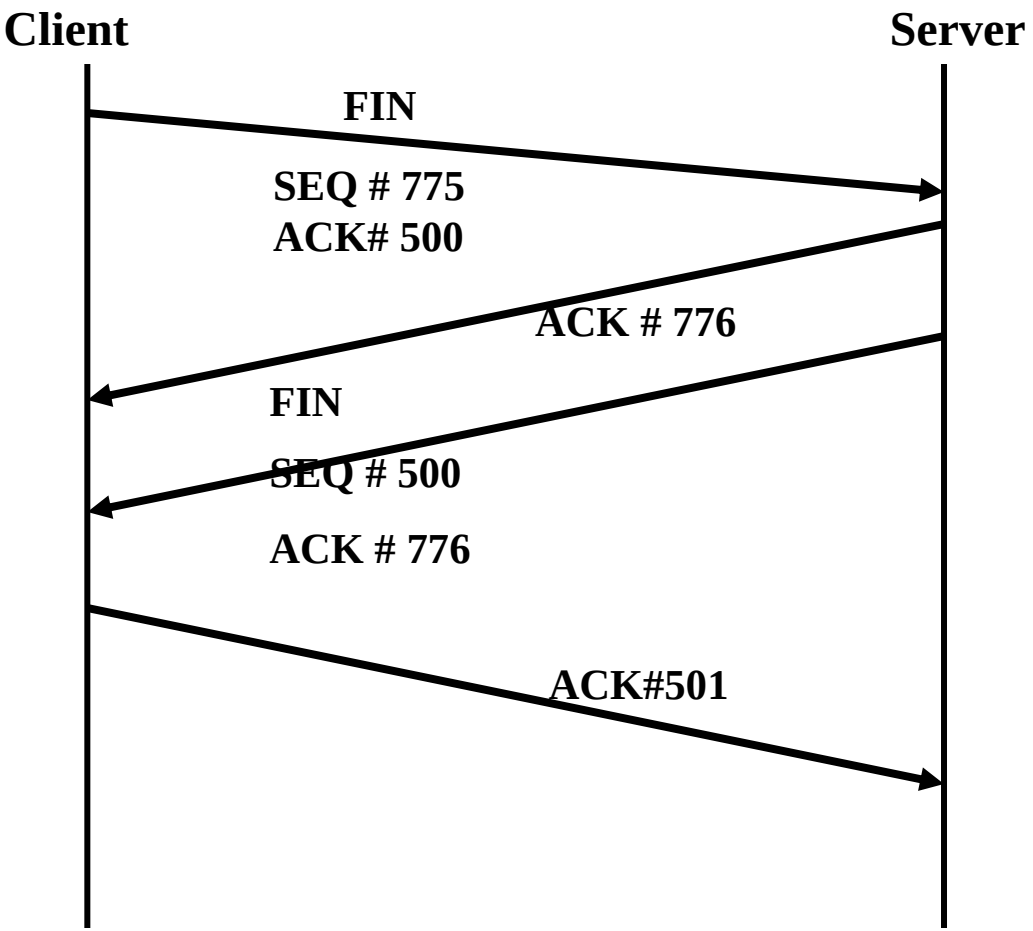
Three-way handshake

- TCP nagyon érzékeny a SYN szegmens elvesztésekre
  - Hosszú időzítések kapcsolat felépítéskor
- SYN szegmens 1 sorszámot foglal
  - Adatoknál bájtokat számlál a seqnum



TCP kapcsolat lezárása

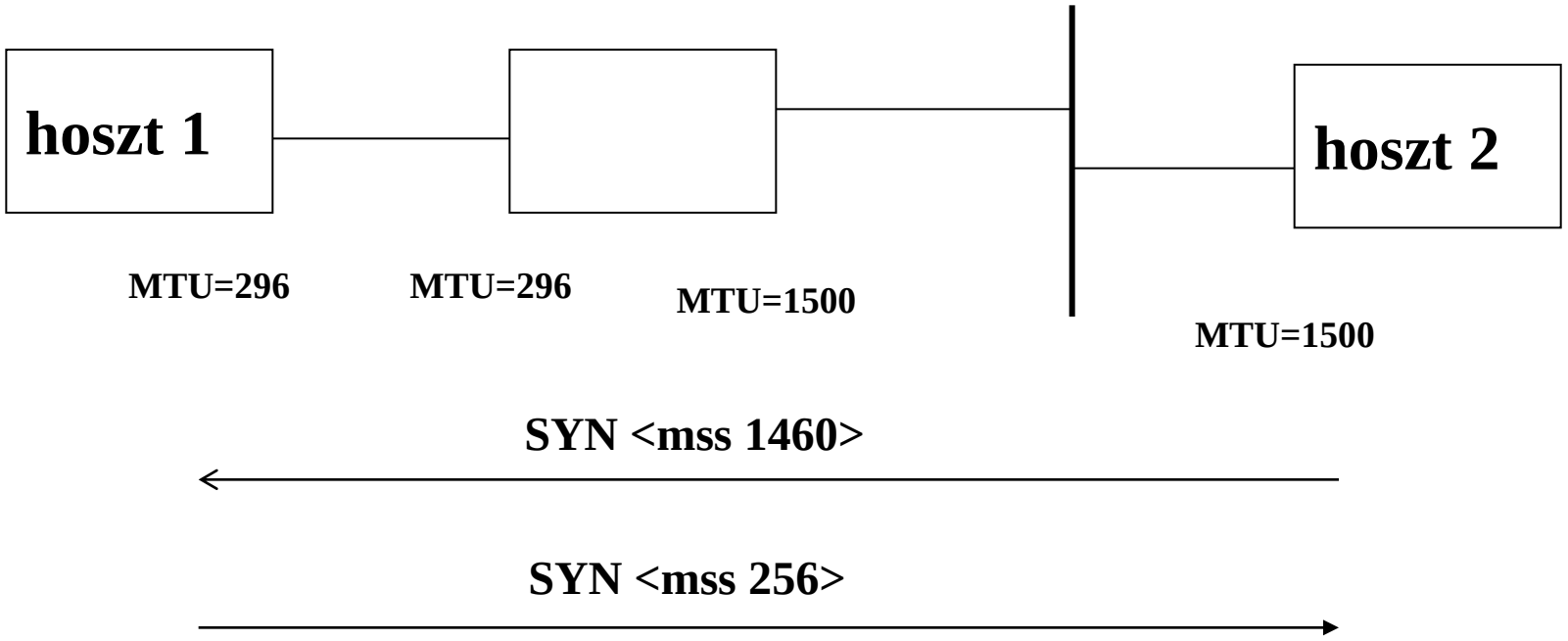
- 4 szegmens a kapcsolat szabályos lezárásakor
- Mindkét végpont egymástól függetlenül lezár
- FIN fogadásakor
  - A TCP-nek értesíteni kell az alkalmazást, hogy a túloldal lezárta a kapcsolatot
  - TCP FIN fogadás után továbbra is tud adatokat küldeni



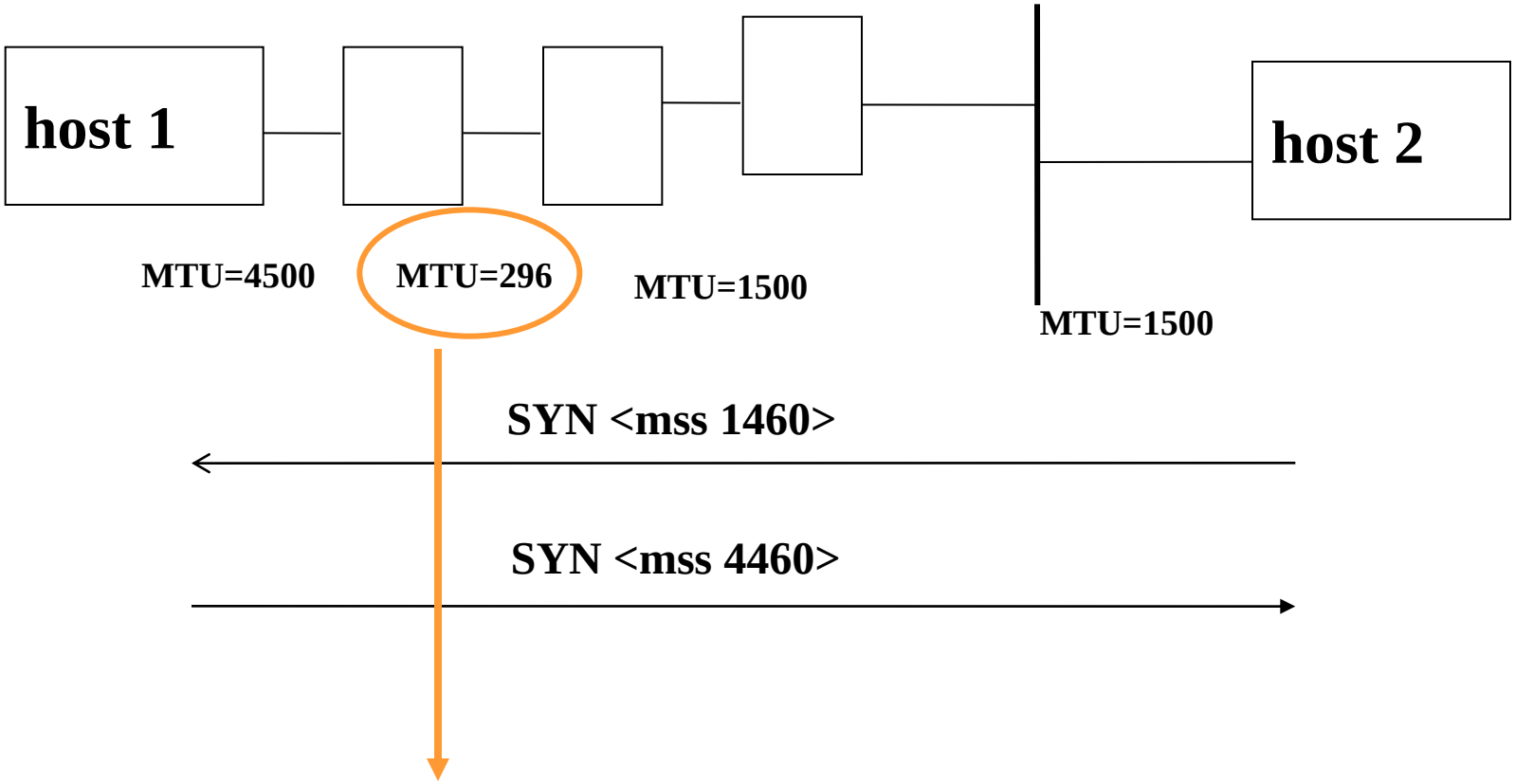
Maximum Segment Size, MSS

- Legnagyobb szegmens méret, melyet a másik oldal küldhet
- Mindkét végpont meghírdeti, hogy milyen méretet vár el
- MSS opció a SYN szegmensben
  - Ha nincs ilyen, akkor alapértelmezett (536 bájt)
- MSS max értéke:
  - A kimenő interfész MTU csökkentve az IP és TCP fejléc méretével
  - Ethernet: 1500-20(IP)-20(TCP)=1460 bájt max

MSS értékek


hoszt 1
hoszt 2
MTU=296
MTU=296
MTU=1500
MTU=1500
SYN <mss 1460>
SYN <mss 256>

MSS értékek 2



MTU Path discovery!

MTU Path felderítés

- Kapcsolat felépítés után a TCP
  - Saját MSS
  - Túloldal által hirdetett MSS
  - Minimumát használja a szegmensek méretéhez
- Előfordulhat az útvonalon kisebb MTU-jú hálózat
  - Kommunikció közben derülhet ki
  - Felderítéséhez a TCP beállított DF bites IP csomagokat küldhet

MTU Path felderítés

- Ha fregmentáció történik ICMP üzenet érkezik vissza:
  - "DF set, can't fragment"
- TCP csökkenti a szegmens méretet és újraküld
  - Az újabb ICMP tartalmazza a következő hop MTU-ját
  - Régebbi ICMP nem tartalmazza, következő kisebb MTU-val próbálkozik a TCP
- Nagyobb MTU érték próbája MSS növeléséhez (~10 percenként)
  - Routing változás esetén lehet
  - Kis szegmens – nagyobb overhead

Csúszó ablak

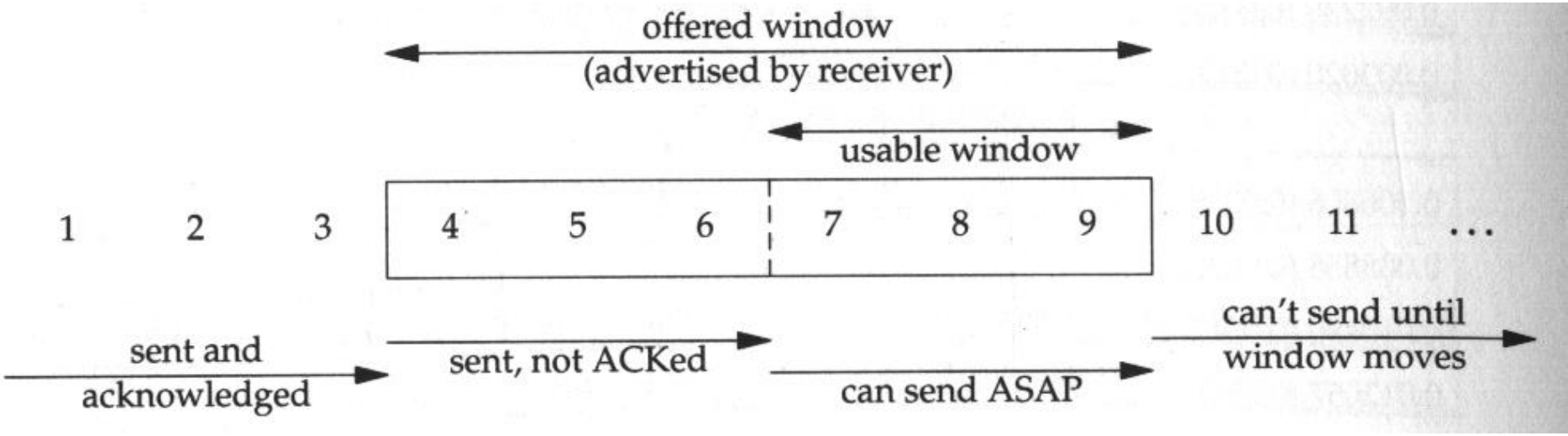
„TCP Sliding window”

MŰEGYETEM 1782

Budapest University of Technology and Economics

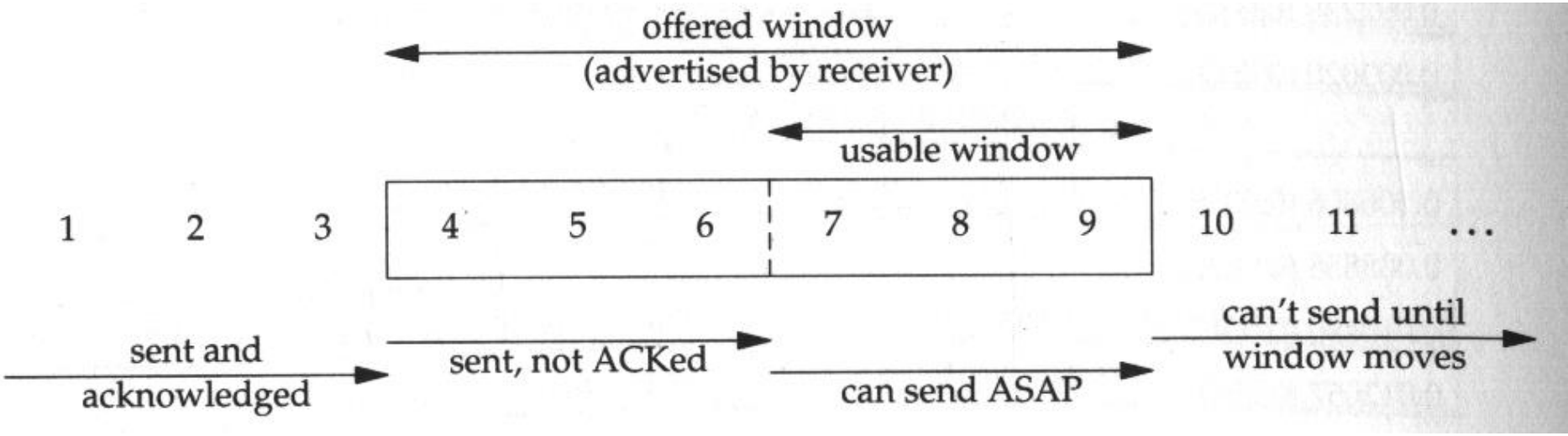
Department of
Telecommunications and Media Informatics

Csúszó ablak – Sliding Window



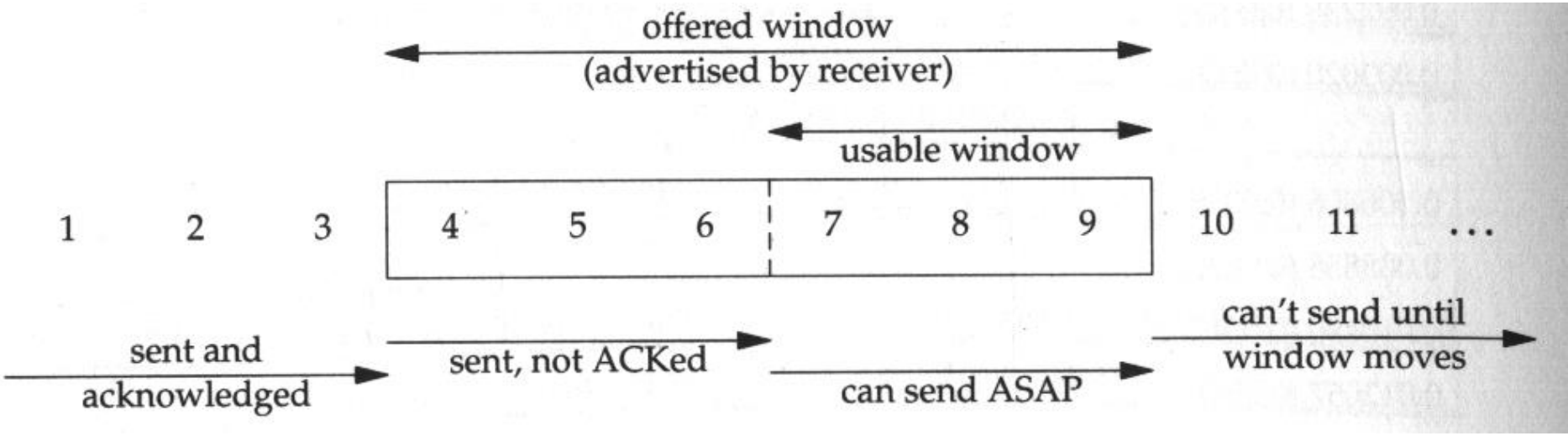
- Csúszóablak jellemzői:
  - Mérete **bájtban** adott
  - Ábrán szegmens számok!!!
- Előző példa:
  - Window méret 4096
  - Ebbe 4 db szegmens fér bele, ha egy szegmens 1024
  - Mert pl. az MSS = 1460

Csúszó ablak – Sliding Window



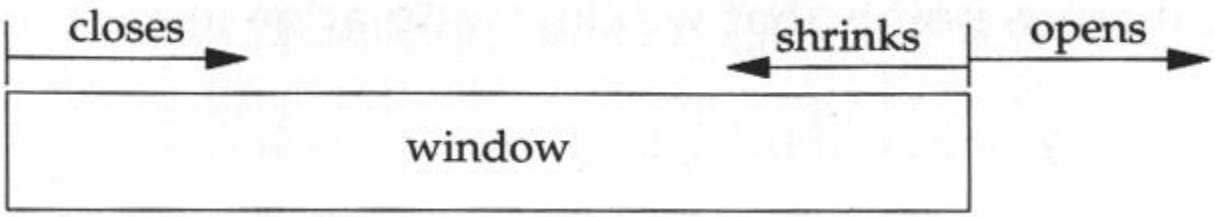
- 1-3 szegmensek már nyugtázottak
- **offered window**:
  - A fogadó által meghirdetett ablakméret
  - 4-9 szegmensek:
    - 4-6 elküldött, de még nem jött nyugta
    - 7-9 azonnal küldhető szegmensek
- **usable window**:
  - az azonnal küldhető szegmensek

Csúszó ablak – Sliding Window



- 10 ... szegmensek
  - Elküldésre váró szegmensek
  - Csak ha érkezik nyugta és az ablak tovább csúszik

A csúszóablak végeinek mozgása



- **Zár - Closes**: ha nyugta érkezik
- **Nyit - Opens**: ha a fogadó oldalon az alkalmazás fogadja az adatokat – ürül a buffer
- **Összehúzódik - Shrinks**:
  - Normál esetben nincs
    - De a TCP-nek kell tudni kezelni!

Csúszóablak működése

http://www2.rad.com/networks/2004/sliding_window/

Az ablak mérete

- Az ablak méretét a fogadó oldal kezeli
  - TCP teljesítményére hatással van
- Általában (window size)
  - Az általános **alapértelmezett érték 8 kbájt**
  - **Nem mindig optimális**
  - **Korlátozó tényező lehet!**
- Az **optimális ablak mérete** függ
  - A kommunikációs média **sávszélességétől**
  - A két hoszt közötti **körülfordulási időtől**
    - Round trip time

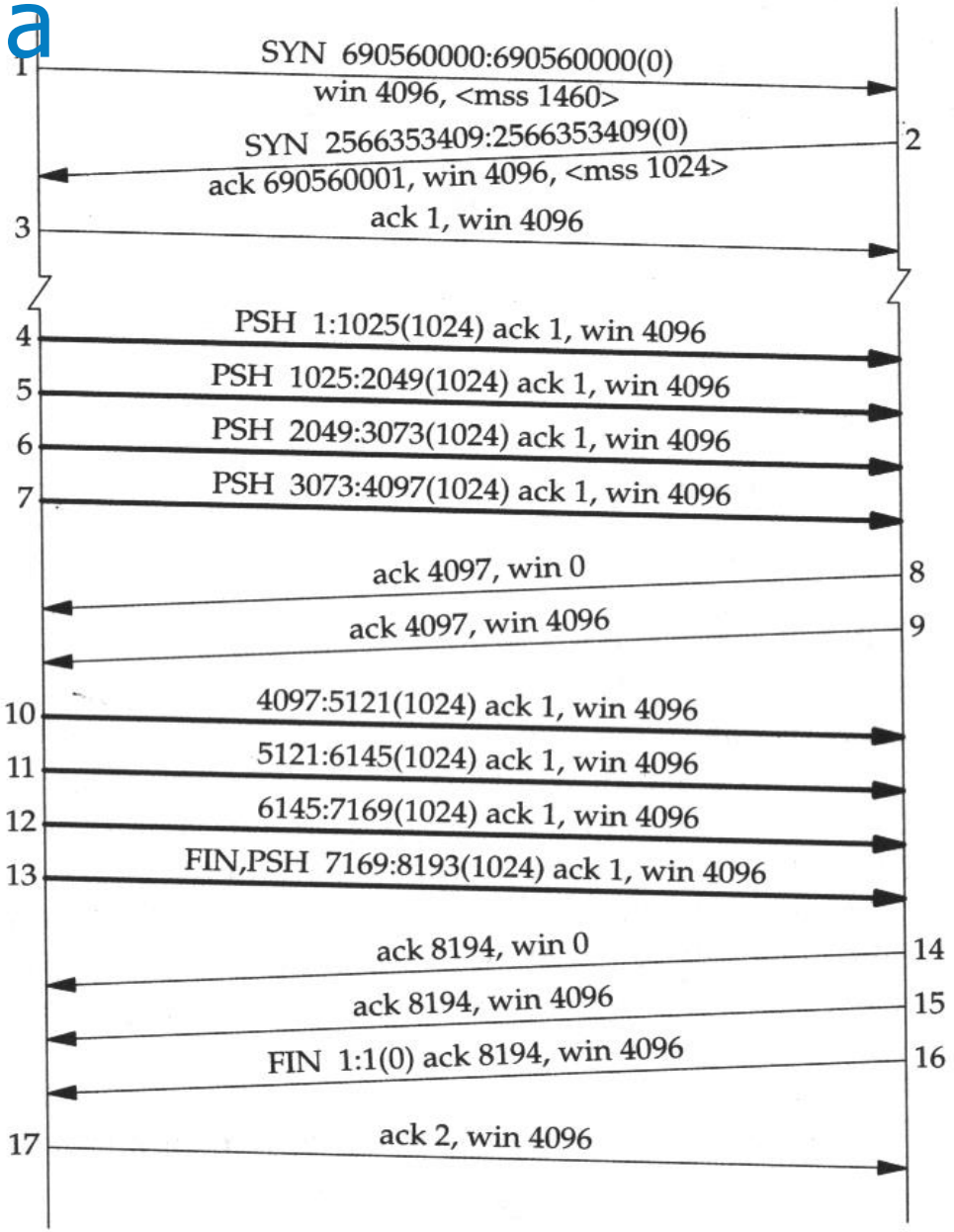
Slow Start

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

Többszörös csomagok küldésének problémája

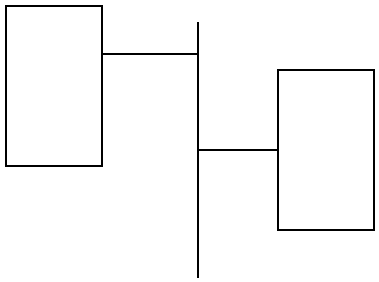
- Korábbi példa:
  - A küldő a kapcsolat elején rögtön elküldi a meghirdetett ablakméret szerinti szegmensmennyiséget
  - Milyen problémákat okozhat ez?



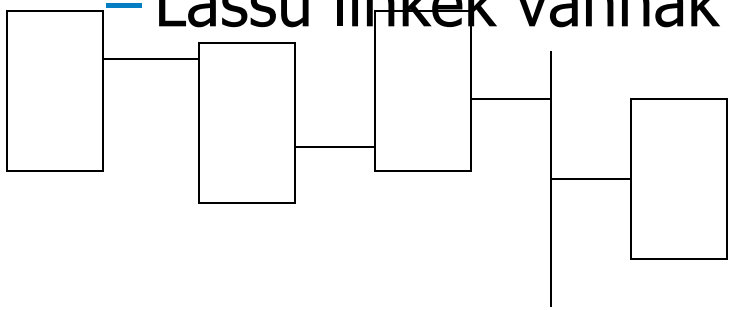
Többszörös csomagok küldésének problémája

Többszörös csomag küldés

- **Alkalmazható**
- **ha a két fél egy LAN-on van**

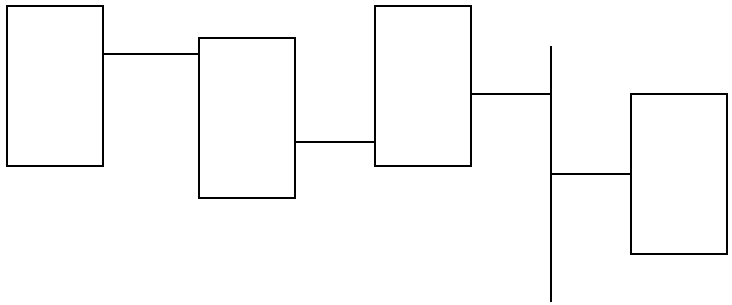


- **Nem alkalmazható**
- ha a két fél nem egy alhálózaton van,
- köztük
  - Routerek
  - Lassú linkek vannak



Többszörös csomag küldés nem alkalmazható

- A közbenső routerekben felsorakozhatnak a csomagok
  - Csomageldobás következhet be



- Jelentős teljesítménycsökkenést okoz
  - **Megoldás: TCP Slow start algoritmusa**

Slow Start – lassú indítás

- Mire való?
  - Elkerülhető vele a meghirdetett ablakméret szerinti többszörös csomagküldésből származó problémák
- Slow Start algoritmus alapja
  - Meghirdetett ablakméreten felül
  - Definiál egy ***congestion window (cwnd)*** változót a kapcsolathoz
  - Megadja az adott pillanatban elküldhető maximális szegmensszámot
  - A kapcsolat állapotától függ
  - Számlálása bájtban (példákban szegmensszámmal!)
- A cwnd - küldő oldal forgalomszabályzója
- Az adv. window - fogadó oldal forgalomszabályzója

Slow Start tulajdonságai

- Új kapcsolat létrehozatalakor
  - Congestion window; **cwnd = 1 szegmens**
  - Jelentés: a küldő 1 szegmenst küldhet
- Minden alkalomkor, ha ACK érkezik
  - cwnd növelhető egy szegmensnyivel
  - (cwnd bájtban számol, de a növekedések mindig szegmens egységnyik).
- A küldő egyszerre mindig ***min(cwnd, advertised window)*** mennyiségű adatot küldhet
- A cwnd maximuma az advertised window
- **Exponenciális növekedés** az átviteli sebességben
- Egy bizonyos határ felett a közbenső hálózat elérhető sávszélességét eléri a kapcsolat
  - A közbenső routerek elkezdik eldobni a csomagokat
  - Vissza szabályzás!

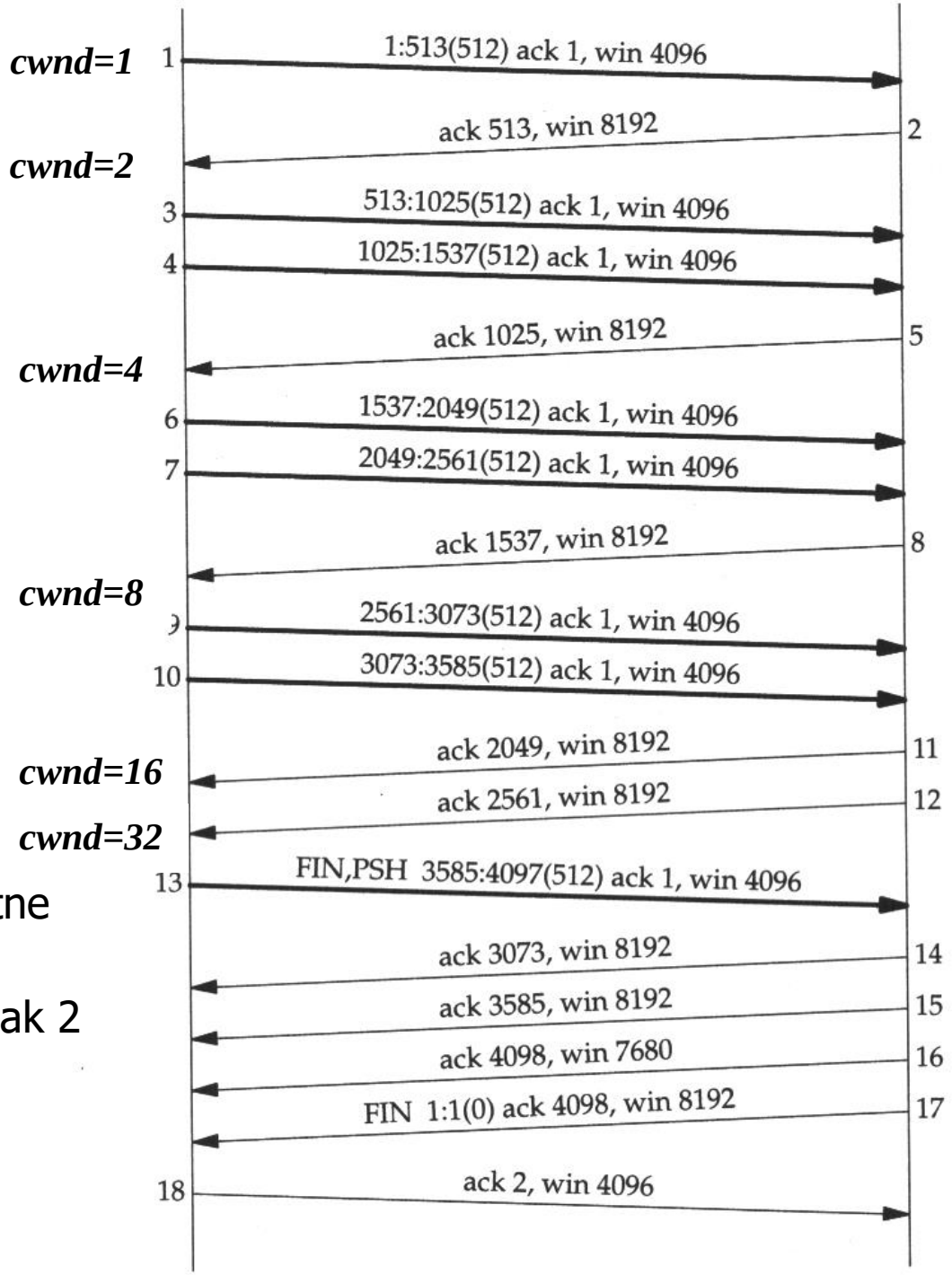
Az exponenciális növekedés

- Küldő kiküld 1 szegmenst (cwnd=1)
  - Vár az ACK-ra
- ACK megérkezik
  - **cwnd** megnövelhető 2-re
  - 2 szegmens küldhető
- Megérkezik a nyugta
  - **cwnd** 4-re növekszik
- **…**

- ≈ minden körülfordulási időnként (RoundTripTime)
  - A **cwnd** megduplázódik

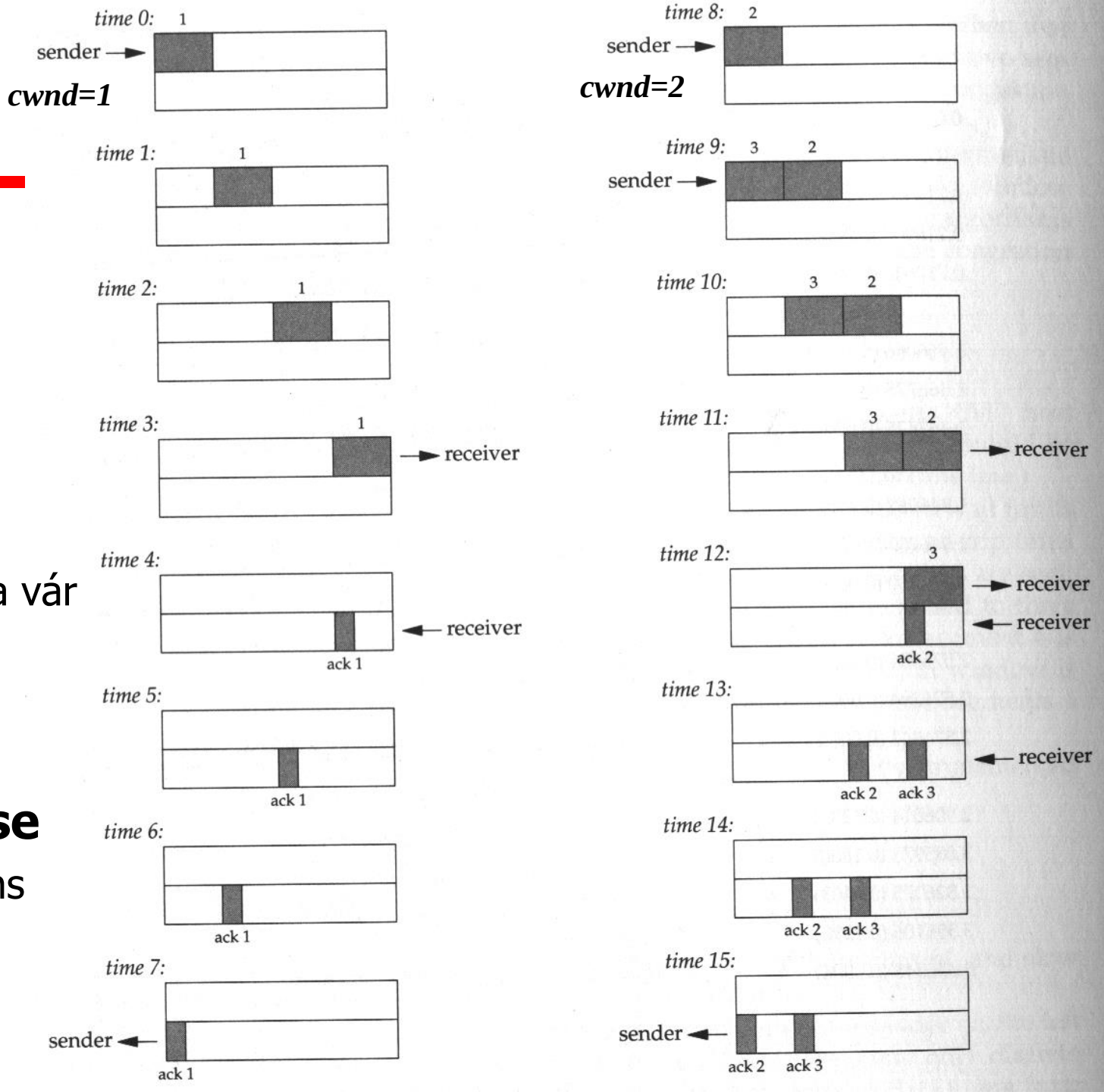
Példa a Slow Startra

- Cwnd=1
  - 1 szegmens küldése
  - Bár a window=8192 a fogadónál
- Nyugta
- Cwnd=2
  - 2 szegmens küldése
- Nyugta
- Cwnd=4
  - 4 szegmens küldése lehetne
  - DE a nyugta csak 1-et nyugtázott, így most is csak 2 szegmens
- ...



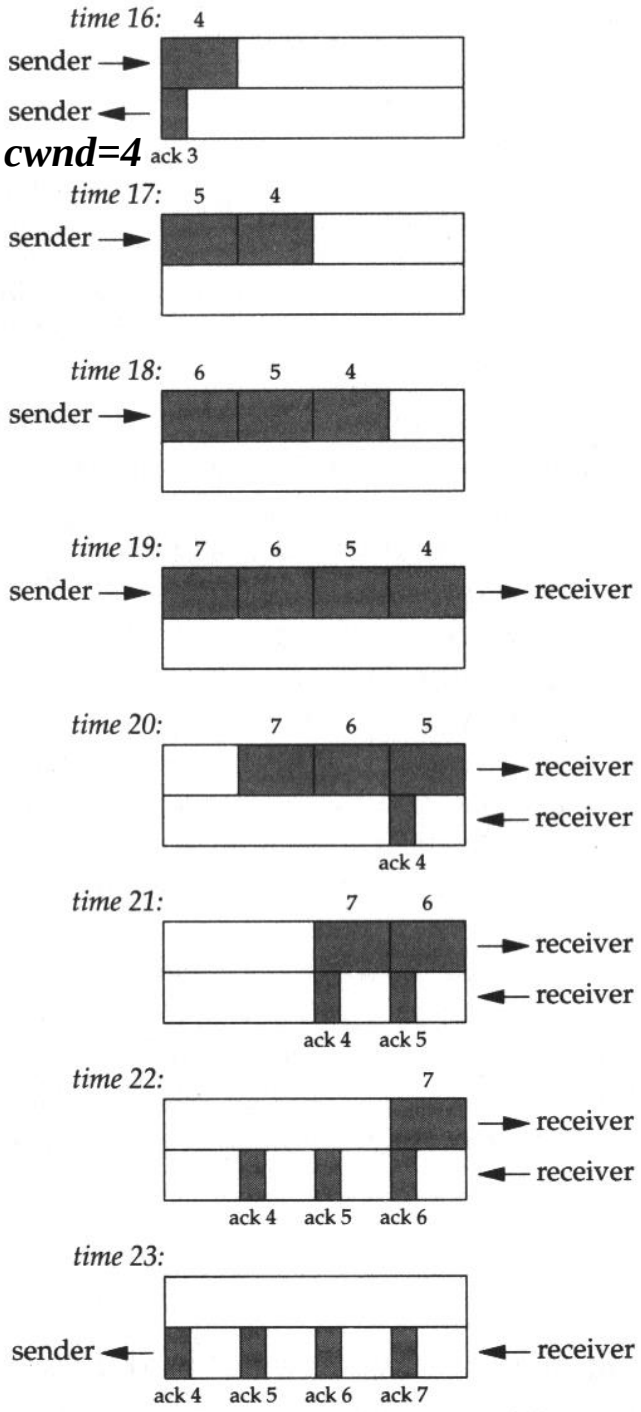
Ideális álapot elérése

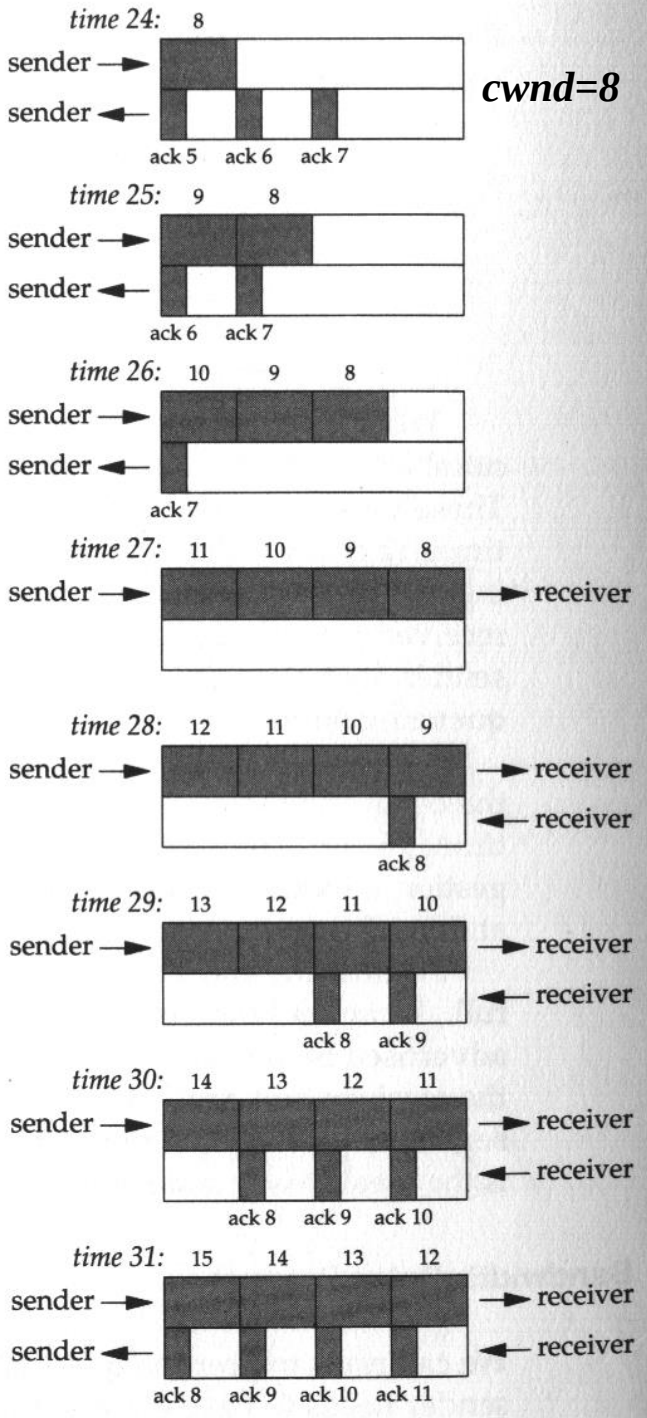
- **time 0**:
  - küldő egy szegmenst továbbít
  - A slow start miatt ACK-ra vár
  - RTT: 8 időegység
- **ACK megérkezése**
  - két szegmens küldhető (cwnd=2)



Ideális állapot elérése

- 31-től a csatorna a küldő és a fogadó között **telített**
  - Ideális állapota a kapcsolatr





TCP interaktív adatfolyam

TCP interactive data flow

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

TCP forgalom típusai, jellemzők

- Bulk – tömeges
  - ftp, e-mail, ... (általában maximális méretű szegmensek)
- Interaktív
  - telnet (minimális, kb. 10 bájtot szállítanak)

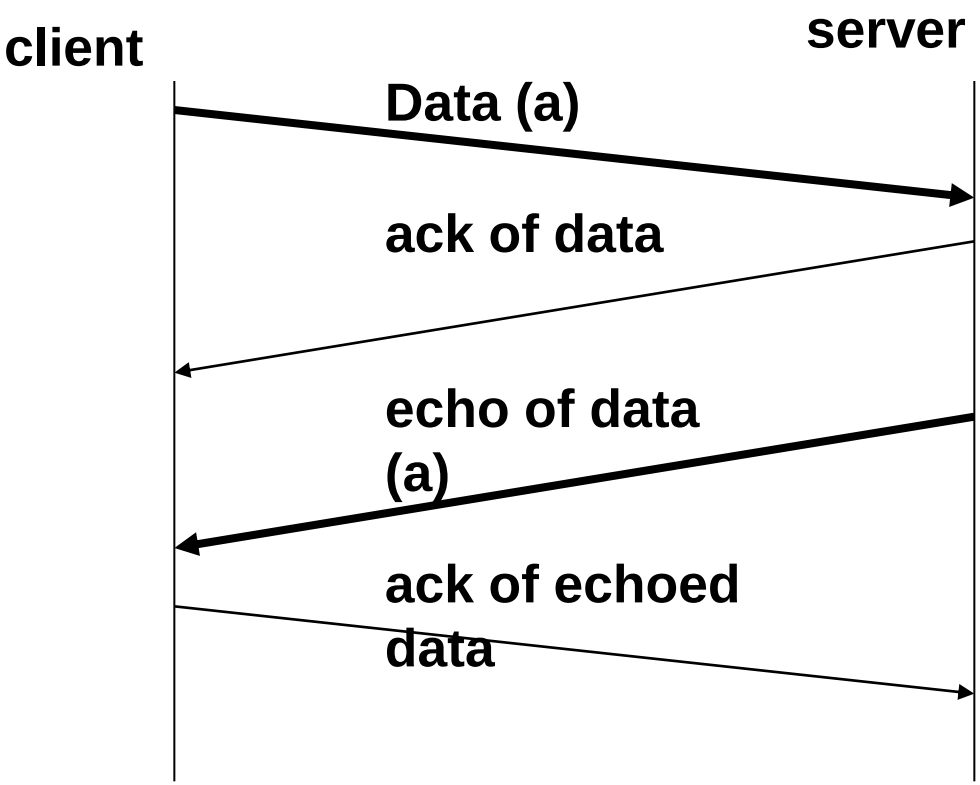
- Csomagok száma alapján
  - Összes TCP szegmens 50%-a bulk adat
- Szállított bájtok alapján
  - 90% bulk adat
  - 10% interaktív adat

Bulk – Interactive

- Adatfolyamvezérléshez különböző típusú algoritmusok
  - Kis csomagok (kevés hasznos adat)
  - Küldés: ritkán

Interaktív adatátvitel (telnet)

- „a” karakter átvitele és megjelenése a képernyőn
  - általában a 2. és a 3. szegmensek összevonhatók

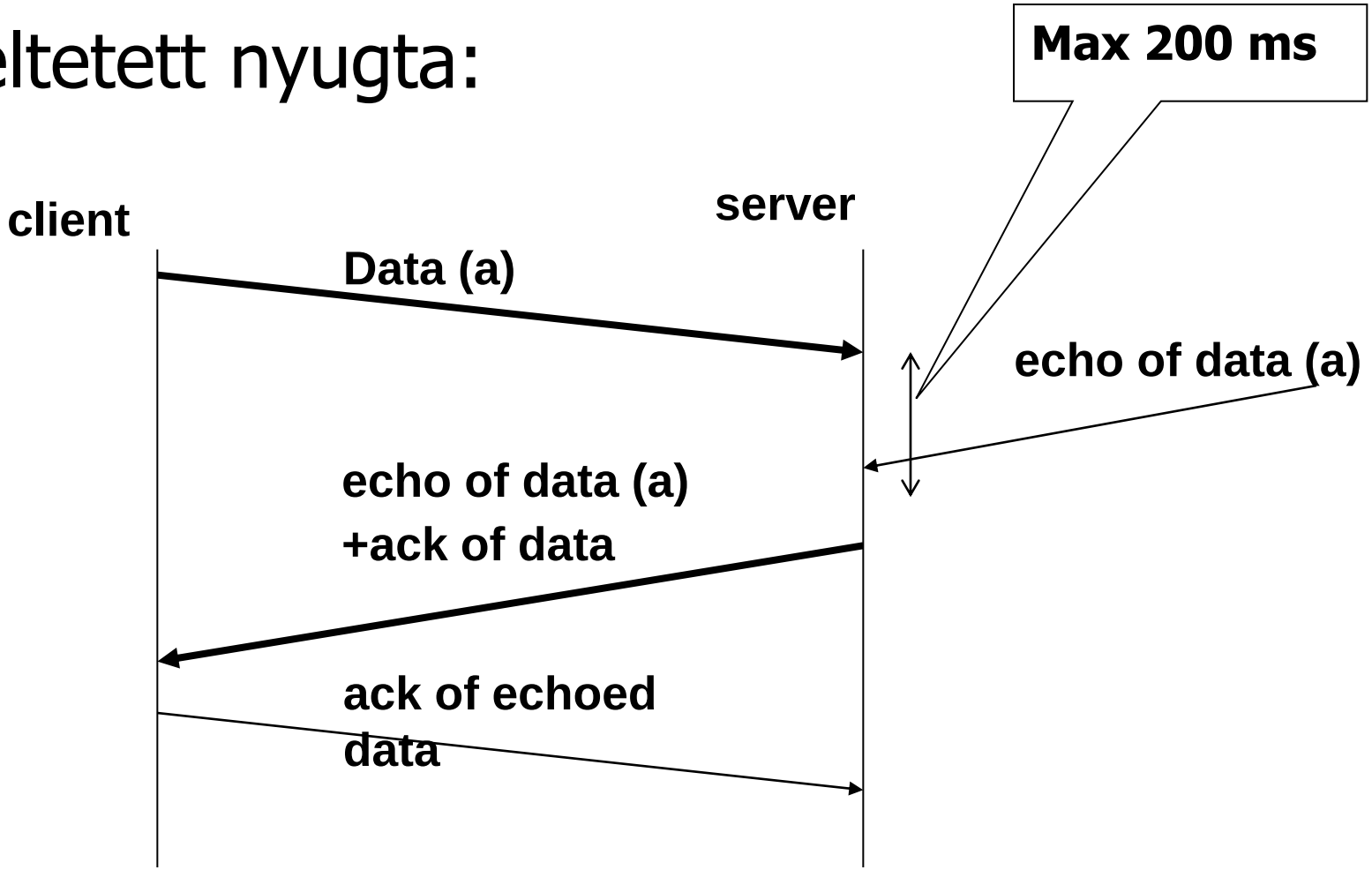


Késleltetett nyugta – Delayed ACK

- TCP az adat fogadásakor nem küld rögtön nyugtát
  - Késlelteti az ACK kiküldését:
  - Várja, hogy jön-e hasznos adat – majd annak a fejlécében nyugtáz
  - (ACK ***piggyback*** with the data)
- Pl. 200 ms-os időzítő az ACK-ra:
  - TCP az adat fogadása után maximum 200 ms-ig késlelteti az ACK küldést
  - Ha nem jön adat ACK magában megy

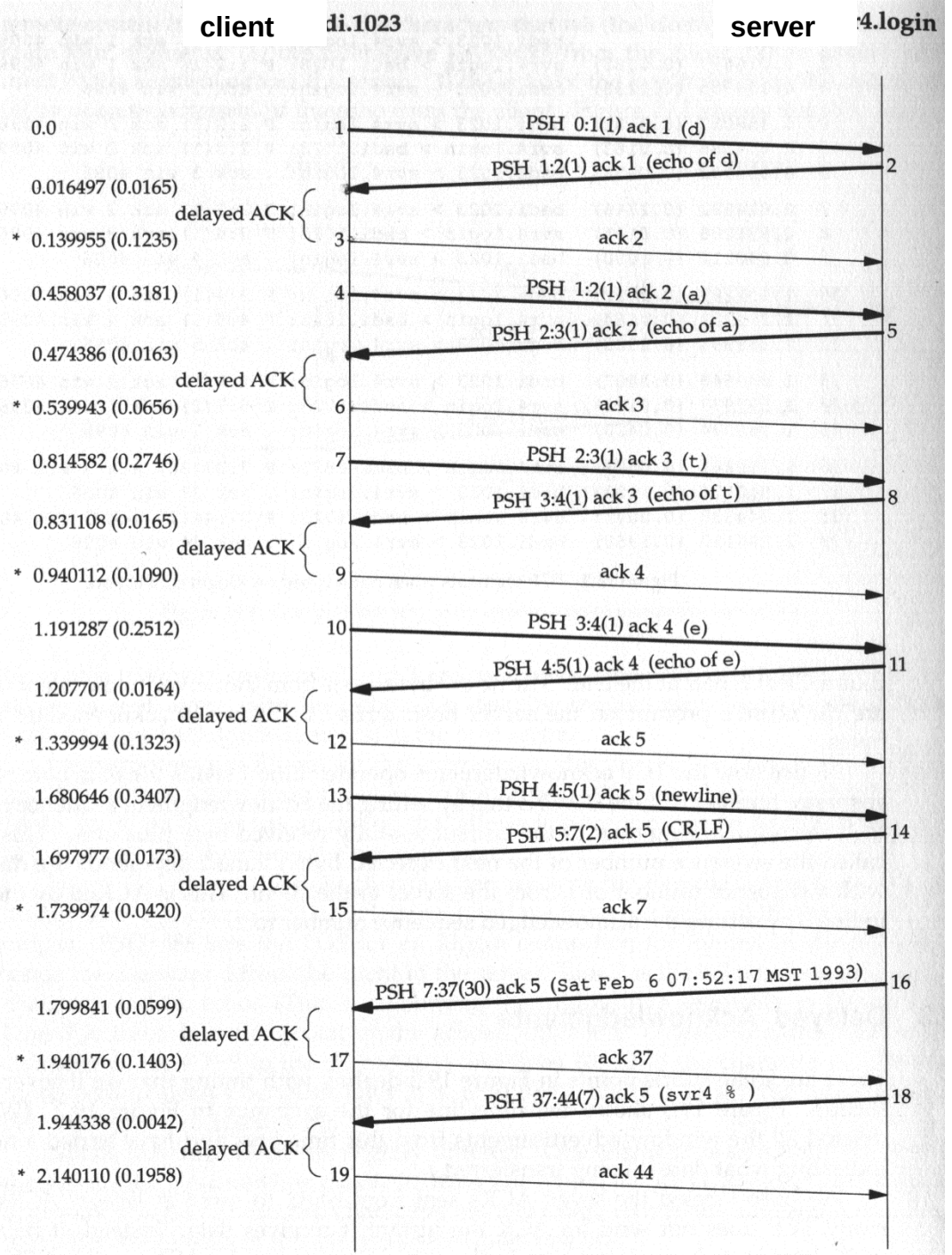
Interaktív adatátvitel (telnet)

- Késleltetett nyugta:



Delayed ACK példa

- Date parancs...
- ...és a válasz



Nagle algoritmus

- Gyakran mennek 1 bájtos csomagok interaktív forgalom esetén
  - 1 bájt adat - 41 bájtos IP datagrammba kerül
  - tinygram
- Nagy overhead: hasznos adat - összes adat arány alacsony
  - Torlódás okozó
    - LAN-on nem probléma
    - WAN-on sok ilyen forgalom probléma lehet
  - Rossz sávszélesség kihasználás

Nagle algoritmus

- Nagle algorithm
  - Ha a TCP kapcsolatnak van kintlévő adata, amelyet még nem nyugtáztak, kis szegmensek nem küldhetők, amíg a nyugta meg nem érkezik
  - Helyette, a kismennyiségű adatokat összegyűjti és nyugta vételekor egy szegmensben küldi el őket
  - Minél gyorsabban jön a nyugta, annál gyorsabban küldi az adatokat

Nagle algoritmus kikapcsolása

- Előfordul, hogy szükséges kis szegmensek átvitele:
  - X-window egér pozíciók
  - Speciális terminálfunkciók
    - Több bájtos adatok (escape karakterrel kezdve)
    - Előfordulhat, hogy az első bájtot elküldi a kliens, majd vár a nyugtára
      - De a szerver is vár a nyugtával
      - Észrevehető késleltetés!

TCP tömeges adatfolyam

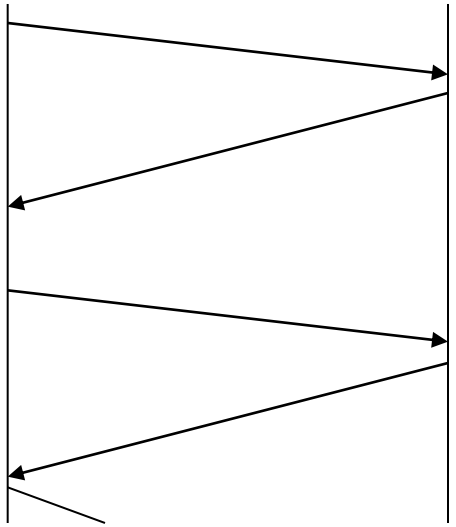
TCP bulk data flow

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

Egyszerű folyamvezérlés

- Stop-and-wait protocol
  - Adatszeletet elküldése
  - Nyugtára vár
    - Nyugta megérkezik – első lépés (következő adatszelettel)
    - Nyugta nem érkezik (időzítés) – előző adatszelet újraküldése, majd nyugtára vár



- Jellemzők
  - Minden adatszeletre nyugta
  - Nagy távolság esetén nagyon lassú

TCP folyamvezérlés

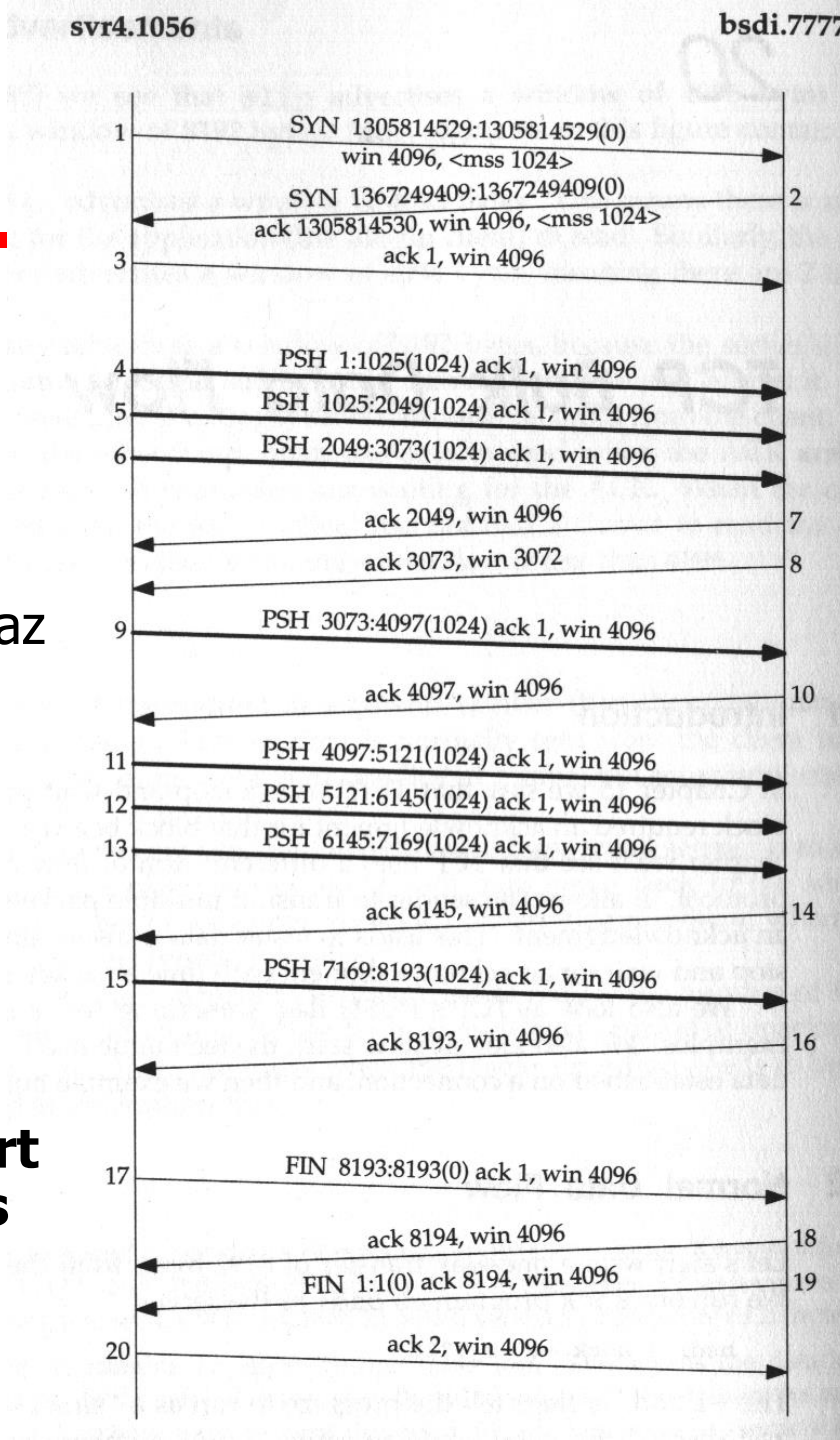
- Sliding Window – csúszóablak

- Nem vár minden szegmensre külön nyugtát
  - Több nyugtázatlan csomag is lehet a hálózaton
  - Többszörös szegmensküldés
  - Nyugták csoportosan is érkezhetnek

- Gyorsabb átvitel
  - Ha megfelelően szabályozott a kint lévő szegmensek száma

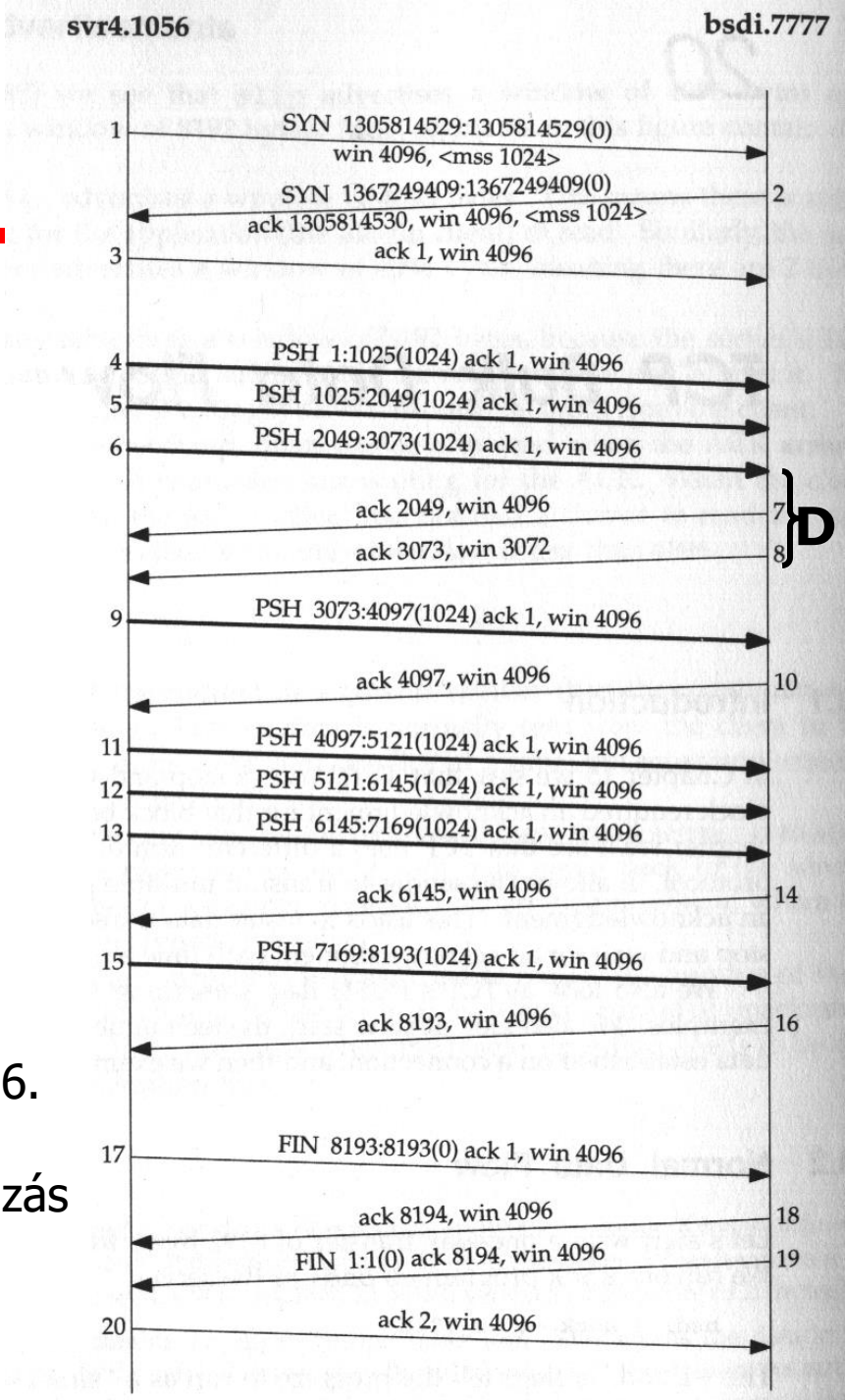
Példa - Normál adatfolyam

- A küldő 3 szegmenst küld (4-6)
- A 7. szegmens nyugtázza az első kettőt – magyarázat:
  - **4,5,6 megérkezik a szerverhez** és az IP bemeneti sorába kerülnek
    - IP input queue
  - TCP feldolgozza a **4.** szegmenst
  - A kapcsolat **késleltetett nyugta küldésre megjelölés**t kap
    - Nyugtaküldő időzítő elindul
  - TCP feldolgozza az **5.** szegmenst
  - 2049-es bájtra **nyugtát generál, mert** már összesen **két kimenő szegmens várakozik**



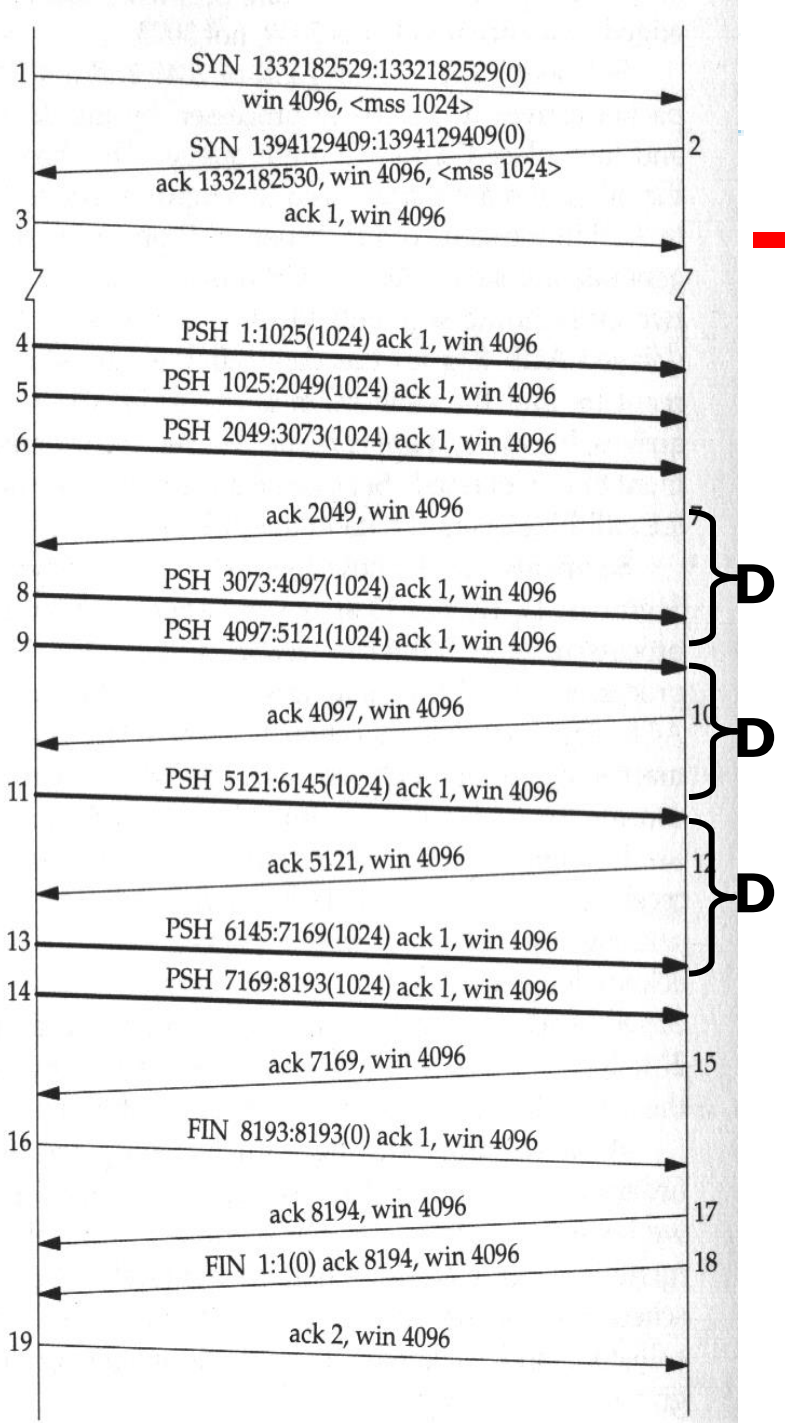
Példa - Normál adatfolyam

- TCP feldolgozza a 6. szegmenst
- A kapcsolat **késleltetett nyugta küldésre** ismét **megjelölés**t kap
  - Nyugtaküldő időzítő elindul
- Nyugtaküldő **időzítő lejár (D)**
  - 9. szegmens még nem jött meg
  - **Nyugtát** (8) a 3073 bájtra **kiküldi**
- Szegmens 8, **win 3072**
  - A Windows méret kevesebb:
    - A nyugta küldésekor még 1024 adat (6. szegmensé) még benne van a TCP fogadó bufferében, melyet az alkalmazás még nem vett át



Példa2 - Normál adatfolyam

- Itt a 8. szegmens a nyugta késleltetés időzítése előtt megérkezik
  - Nyugta 4097-ig kiküldésre kerül
  - Bár még a kiküldés előtt megérkezik a 9. szegmens
- 12. nyugta
  - Csak a 9. Szegmens – időzítő!

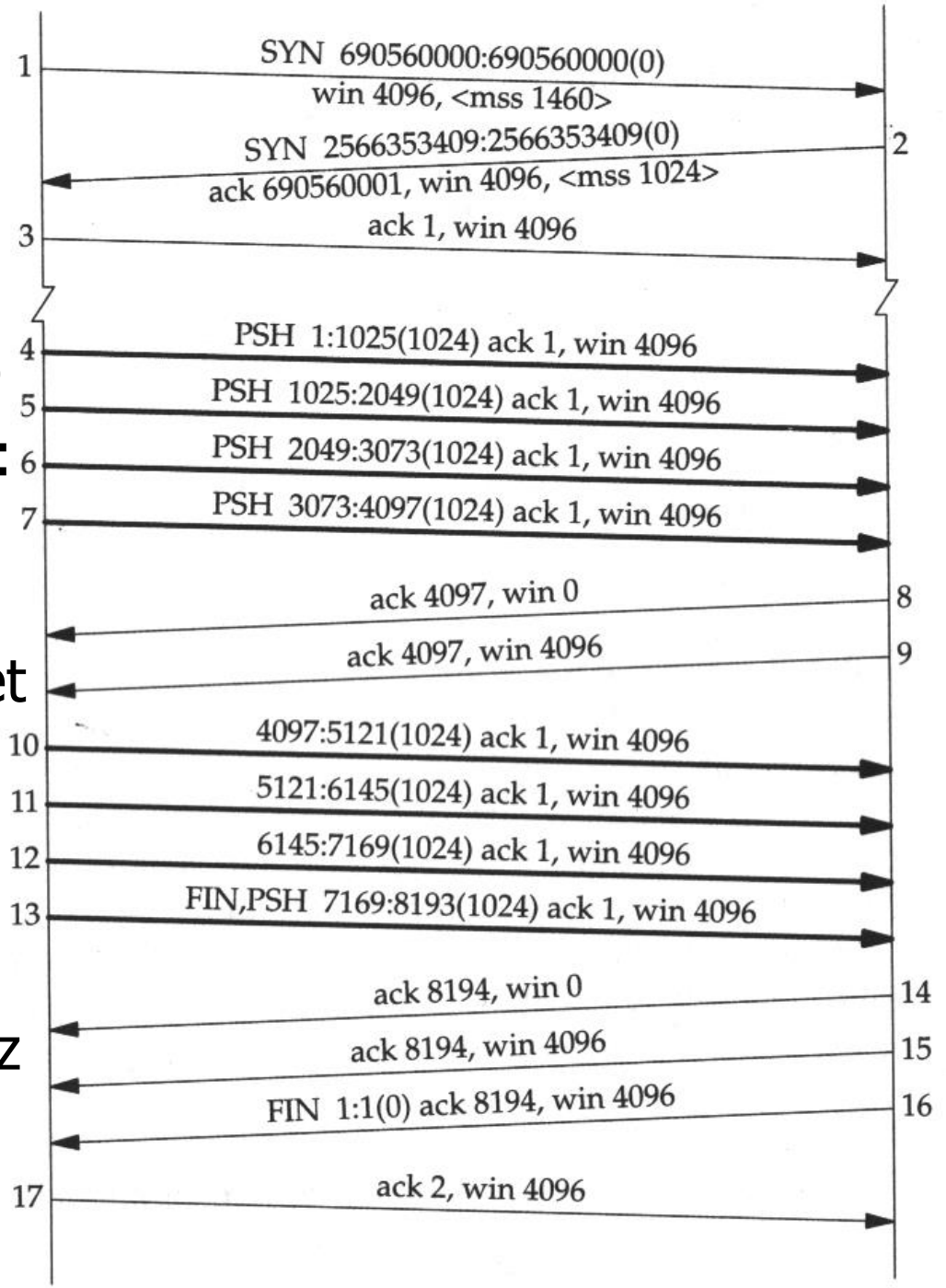


Gyors küldő, lassú fogadó

- Küldő a meghirdetett ablakméret szerint küld:
  - Megtölti a fogadó bufferét
  - Küldő vár a nyugtára
  - A fogadó oldal lassú, nem tudja az alkalmazásnak tovább adni az adatokat – buffere még mindig tele
- A fogadó nyugtájában az „advertised window size” = 0
  - A küldő nem küld több szegmenst
- Ebből az állapotból ki kell billenteni a küldőt:
  - Ha már ürül a buffere
  - Window Update üzenet:
    - Újabb nyugta ugyanarra a szegmensre
    - De új window mérettel

Példa2
window update

- Egymás utáni 4 szegmens küldése (**4-7**)
- Fogadó buffere tele lesz:
  - Ack: 4097 (**8**)
  - Window: **0** !!!
- ***Window update*** üzenet
  - mégegy ACK: 4097 (**9**),
  - Window: 4096 !!!
- A TCP fogadó bufferét az alkalmazás kiürítette



TCP időzítés és újraküldés

- A TCP szegmensek és nyugták elveszhetnek a hálózatban
  - Torlódások miatt a közbeeső routerekben csomageldobás lehet
- **retransmission timer** – újraküldési időzítés
  - Szegmens megküldésekor időzítő indul
  - Lejáratáig az ACK-nak meg kell érkezni
  - Ha nem érkezik meg az időzítés lejáratáig (timeout)
    - A szegmenst újraküldi
    - Cwnd=1 lesz ismét

Sávszélesség – késleltetés szorzat

- Milyen nagy legyen az ablakméret, hogy ideális állapot legyen?
- Előző példa:
  - A küldőnek 8 szegmenst kell a csatornára engedni nyugták nélkül, a maximális teljesítményhez

- **Csatorna kapacitása = sávszélesség** × **körülfordulási idő**
- (bandwidth-delay product)

- A fogadó meghirdetett ablakméretének egyenlőnek kell lenni a csatorna kapacitásával

Round Trip Time számítás

- RTT – becsléssel számítja a TCP
  - Az előző érték (RTT)
  - És a mért (M) értékből származik

$$RTT \leftarrow \alpha RTT + (1-\alpha)M$$

  - α - csillapító faktor ($\approx$0.9)
  - Minden méréskor frissítés

Újraküldési időzítés számítása

Retransmission TimeOut

RTO = RTT $\times\beta$

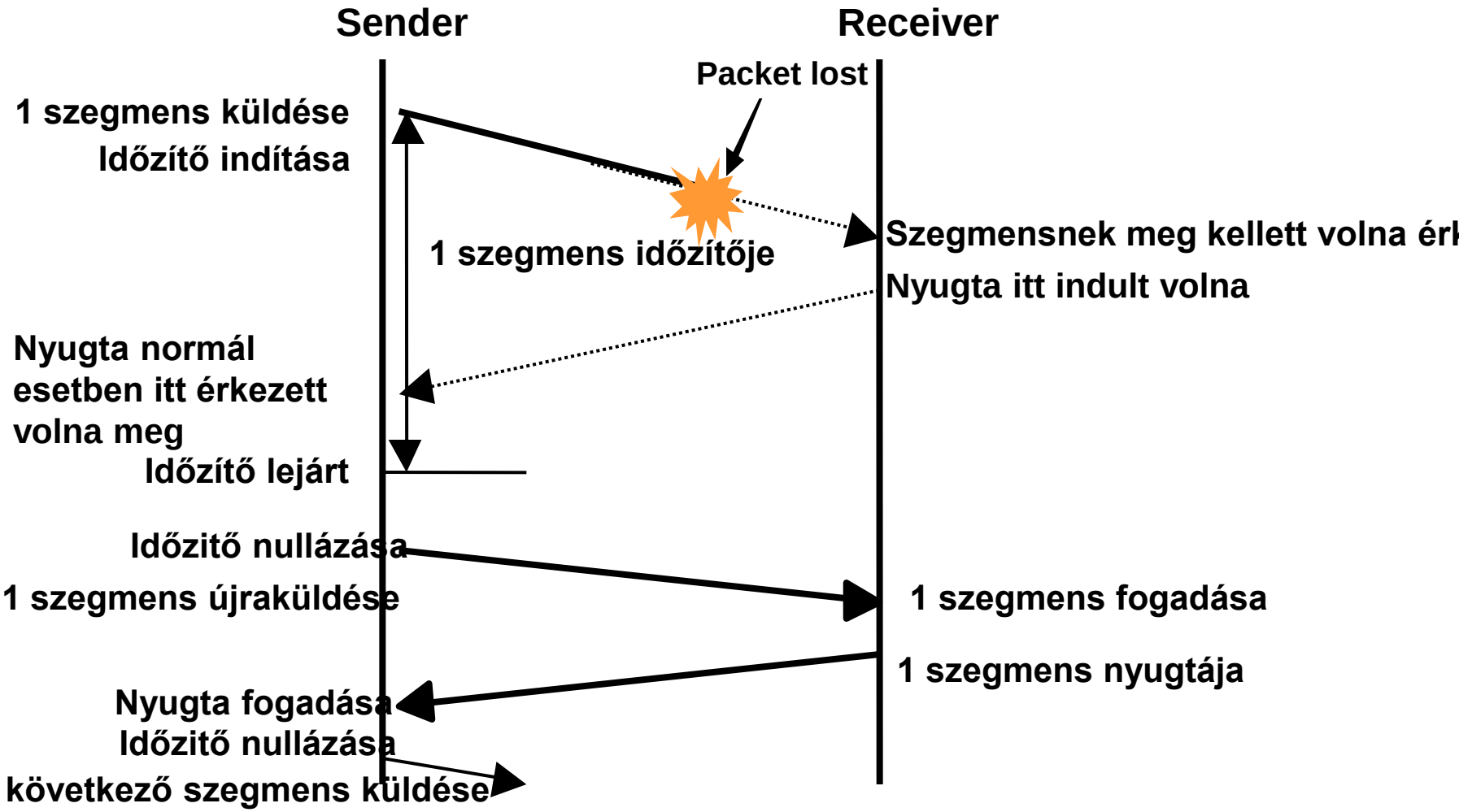
- β : késleltetési variancia faktor
  - Ajánlott érték 2

- **probléma**: a késleltetés nagy ingadozásait nem tudja követni
- Jacobson féle számítás
  - Késleltetés ingadozás jobb követésére más RTT számítás

Jacobson féle számítás

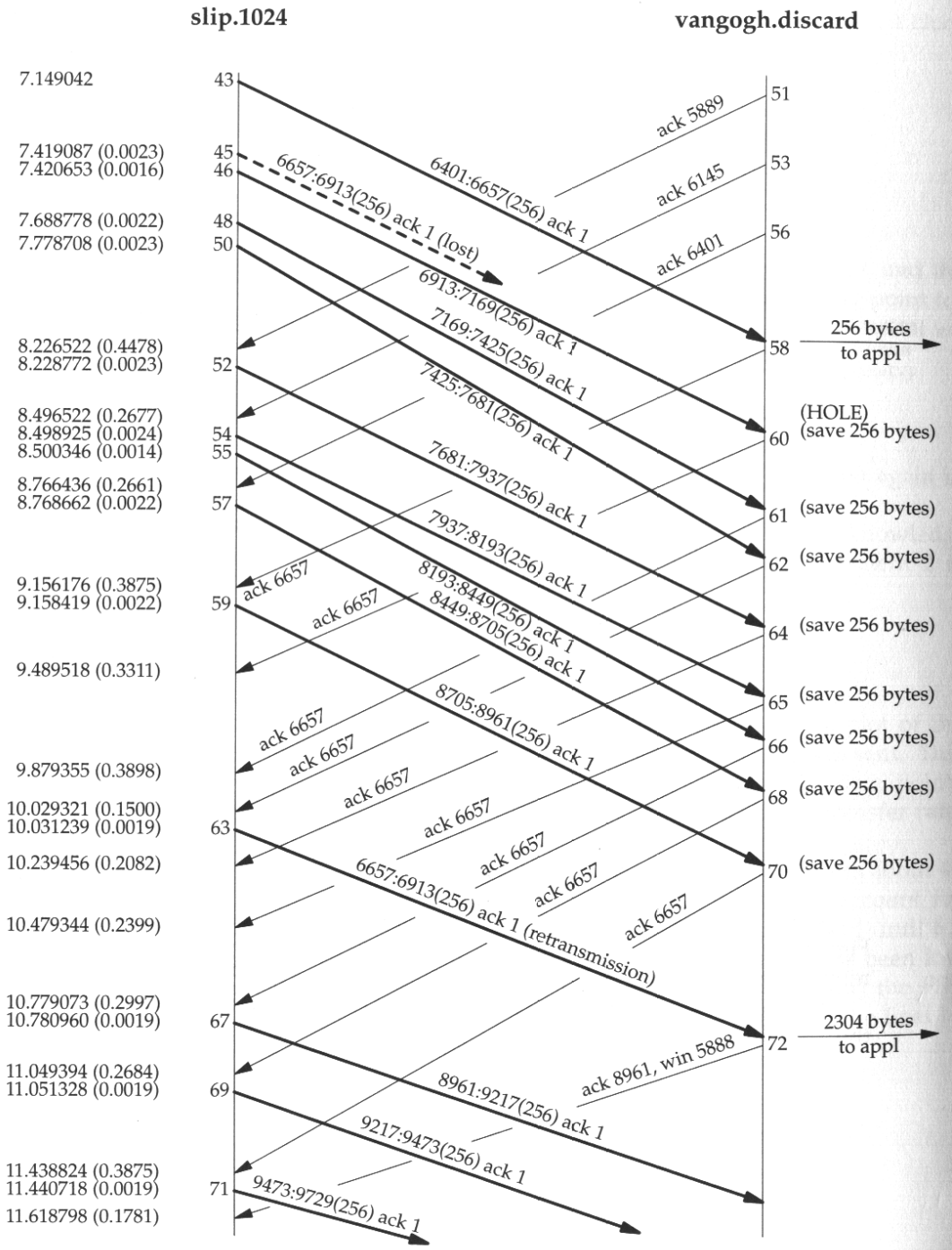
- *Err = M – A*
  - *M Mért érték*
  - *A csillapított RTT (átlag becslése)*
- *A ← A + g Err*
  - *g = 0.125*
- *D ← D + h (Err - D)*
  - *D csillapított átlagos eltérés*
  - *h = 0.25*
- ***RTO = A + 4D***

Példa: újraküldés időzítés lejártakor



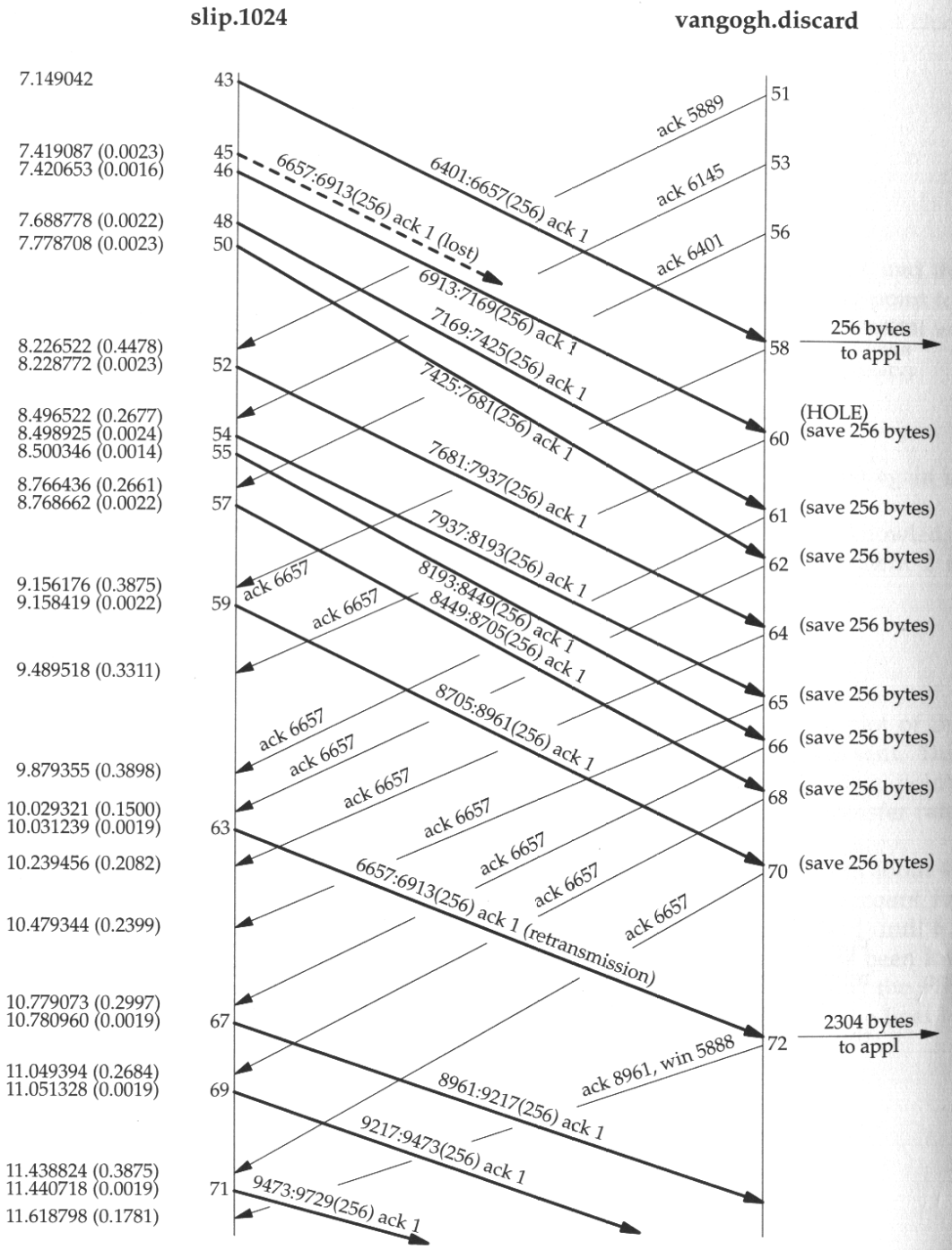
Újraküldés ACK-k száma miatt

- 45. szegmens elveszett
- A 62. szegmens (nyugta) hatására újraküldés
  - Ez volt a 3. egyforma ACK 6657-ből
- Fogadó folyamatosan menti a további szegmenseket
  - A hiányzó utániak így nem vesznek el



Újraküldés ACK-k száma miatt

- A TCP implementáció számolja az egyforma ACK-k számát
- Ha a harmadik egyforma is megérkezik
  - Következtet: a hivatkozott szegmens elveszett
  - Reméli, hogy csak az az egy
  - Csak azt az egy szegmenst küldi újra
- Jacobson féle ***fast retransmit algorithm***
  - ***Gyors újraküldés algoritmus***



Újraküldés ACK-k száma miatt 2

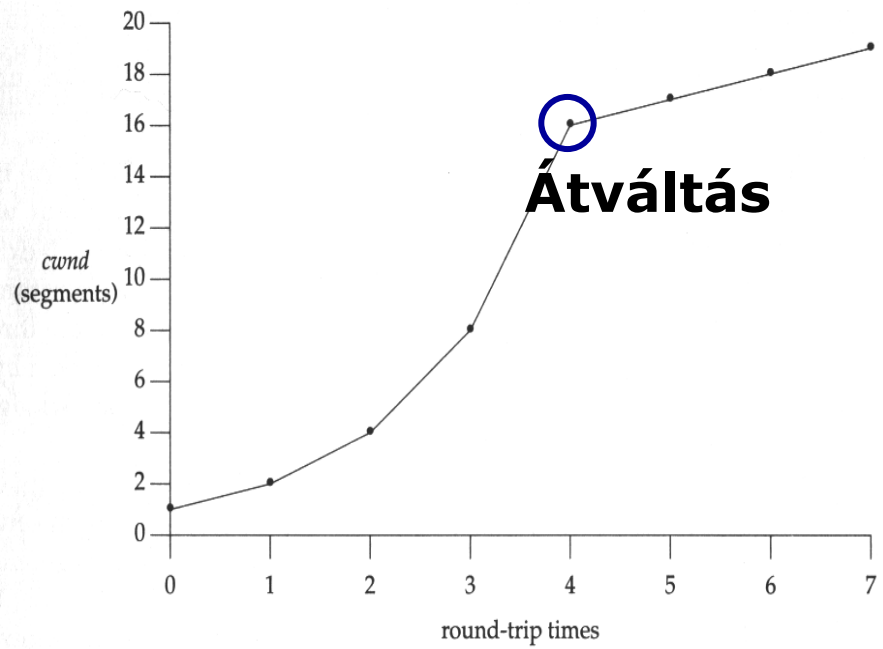
- gyorsítás:
  - A fogadó egy szegmens hiányakor rögtön duplikált ACK küld vissza
  - A küldő gyorsabban újraküldi a hiányzó szegmenst
  - Gyorsabban visszaáll a rendes adatátvitel
- Jelenleg a TCP nem tud
  - Egy szegmens hiányát jelezni
  - Sorrend eltéréseket jelezni

Gyors újraküldés, gyors visszaállítás algoritmus

- Fast retransmit – fast recovery
- 3 vagy több duplikált ACK
  - Gyors jelzése egy szegmens elvesztésének
- Újraküldés (már az RTO lejárata előtt)
  - ***fast retransmit – gyors újraküldés***
- De utána:
  - **congestion avoidance** – torlódás elkerülés
  - NEM slow start – lassú indítás
- Ez a ***fast recovery*** algoritmus

Congestion avoidance alg.

- A Slow Starttal a ***cwnd*** exponenciálisan nő
- Az exponenciális növekedés vége: ha csomagvesztés fordul elő
  - Ezután újraküldés
  - ***cwnd*** lecsökken 1-re, és indul újra a slow-start
- ***Congestion avoidance*** algoritmus lehetővé teszi, hogy az exponenciális növekedés additív növekedéssé váljon
- Ezzel a TCP a cwnd növekedését 1-re tudja maximalizálni egy RTT alatt
- A csomagvesztések kevésbé gyakoriak lesznek



CWND növekedése a congestion avoidance alatt

- Minden ACK fogadásakor

  $cwnd = cwnd + 1/cwnd$

  - Additív növekedés
    - A slow start exponenciális növekedésével szemben
  - cwnd gyakorlatilag 1 szegmens értékkel nő RTT-ként

$cwnd = cwnd + szegmensméret \times szegmensméret/cwnd$

- cwnd valódi értéke szintén bájtban!

Congestion avoidance alg.

- *cwnd: Congestion window*
- *ssthresh:* slow start threshold size

1. Kapcsolat kezdetén a kiindulási értékek
   cwnd =1 szegmens,
   ssthresh=65536 bájt

2. TCP küldő maximum
   min(cwnd; advertised window)
   szegmenst küldhet

Congestion avoidance alg.

3. cwnd értéke slow start szerint (exponenciálisan) nő, míg csomagvesztés nem történik

   ssthresh új értéke: **min(cwnd , advertised window)/2** de legalább 2

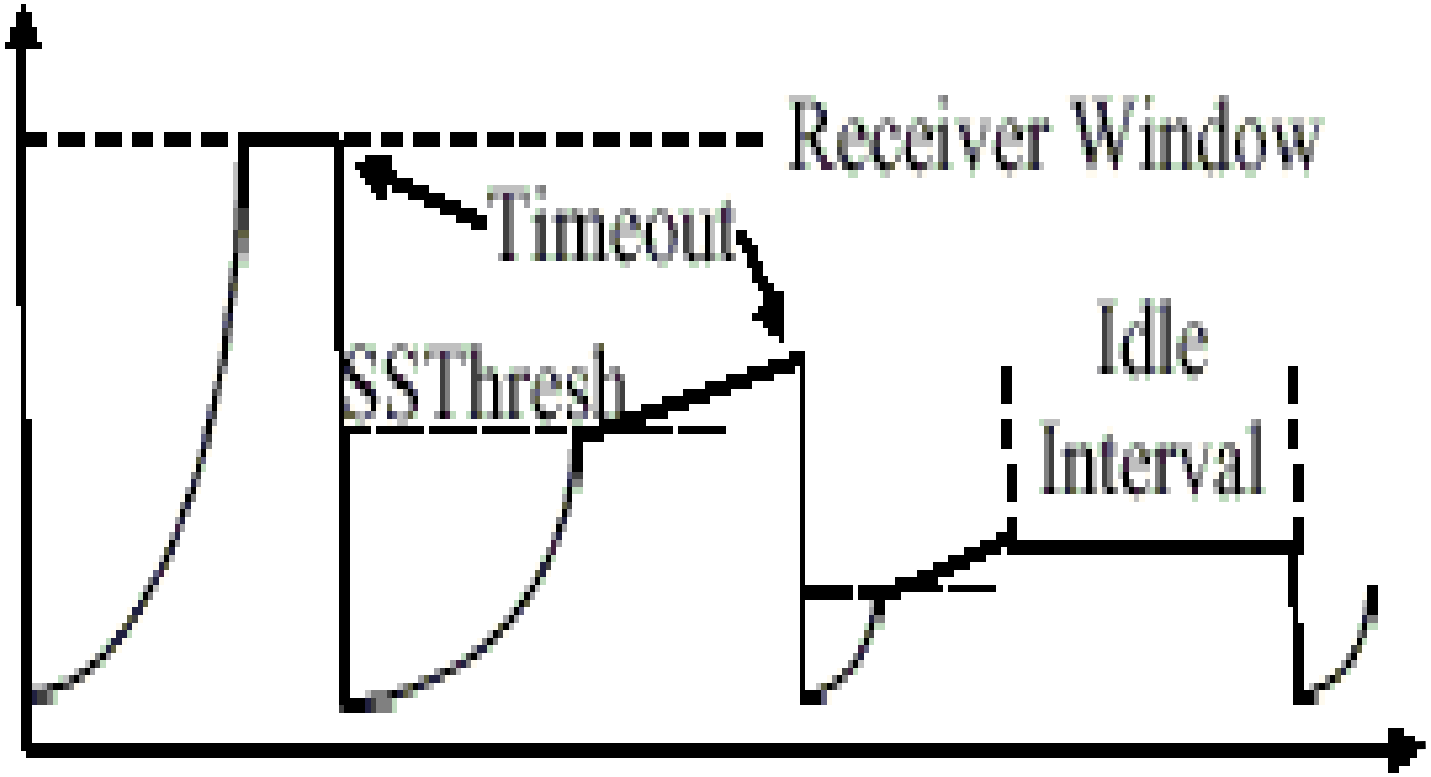
   Ha a torlódás időzítő lejárata miatt következett be, a cwnd értéke 1 lesz és újra slow start

4.

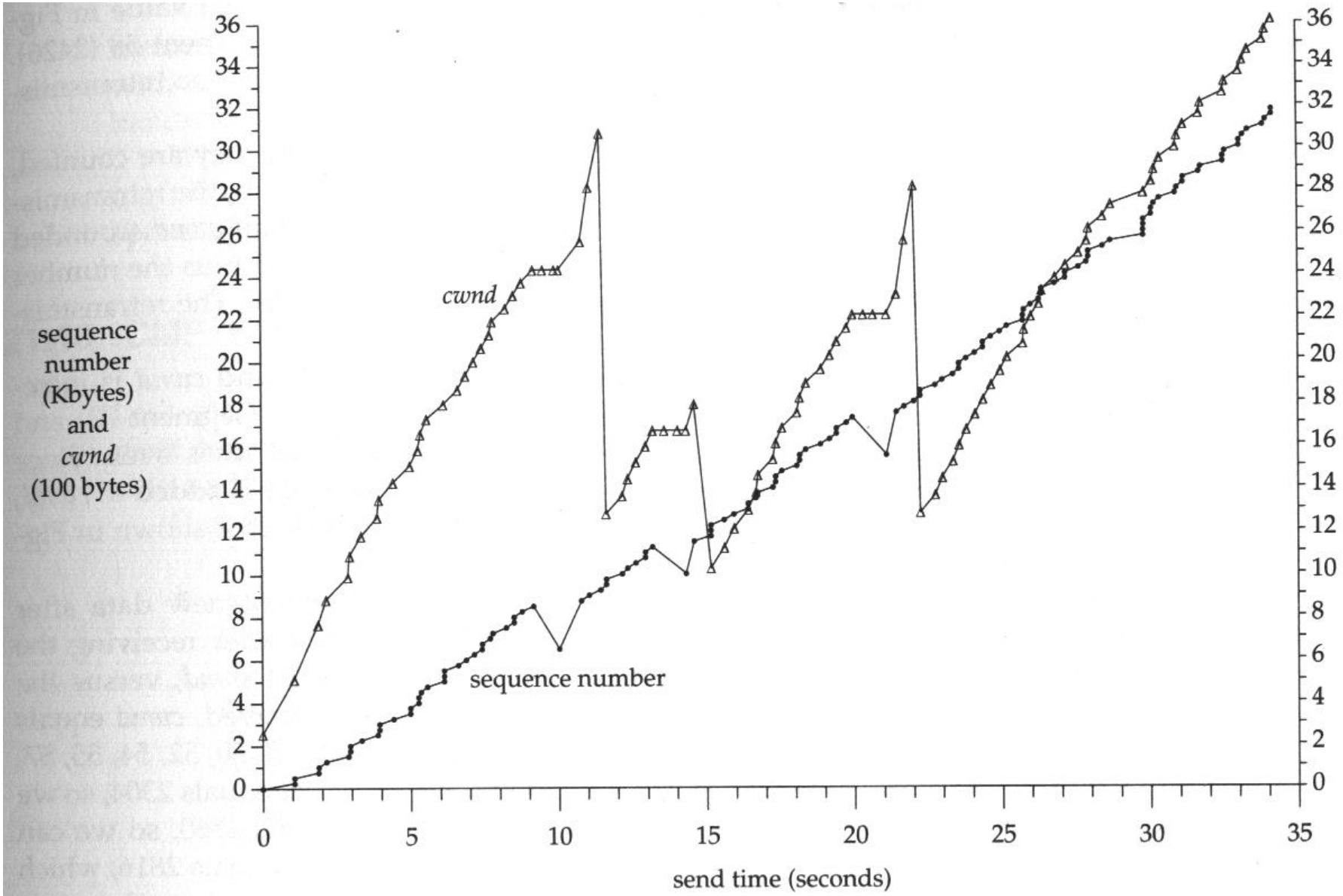
Ha a cwnd kevesebb vagy egyenlő ssthresh-sel, slow start szerinti növekedés

Ha a cwnd nagyobb, mint az ssthresh, congestion avoidance lép működésbe, cwnd legfeljebb 1 szegmenssel növekszik RTT-ként

Congestion avoidance



Valós példa


sequence number (Kbytes) and cwnd (100 bytes)
cwnd
sequence number
send time (seconds)
0
5
10
15
20
25
30
35
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36

Köszönöm a figyelmet

- Vége -

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics