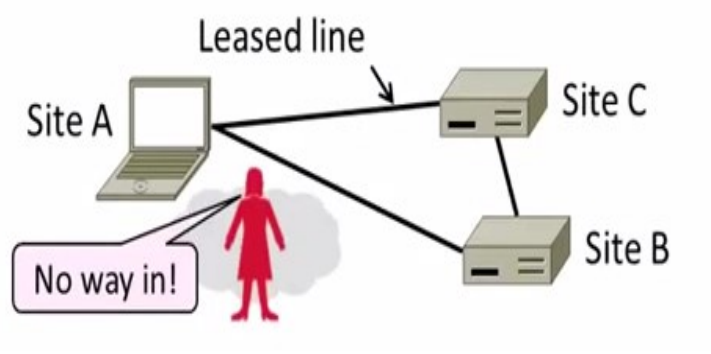
# Számítógépes Hálózatok

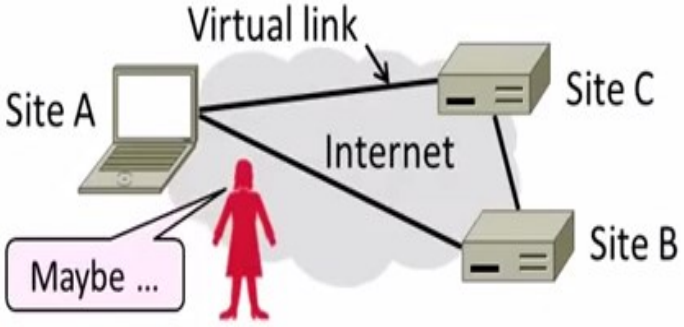
## 9. Előadás: VPN + Szállítói réteg

Based on slides from **Zoltán Ács ELTE** and D. Choffnes Northeastern U., Philippa Gill from StonyBrook University , Revised Spring 2016 by S. Laki

# Virtuális magánhálózatok alapok

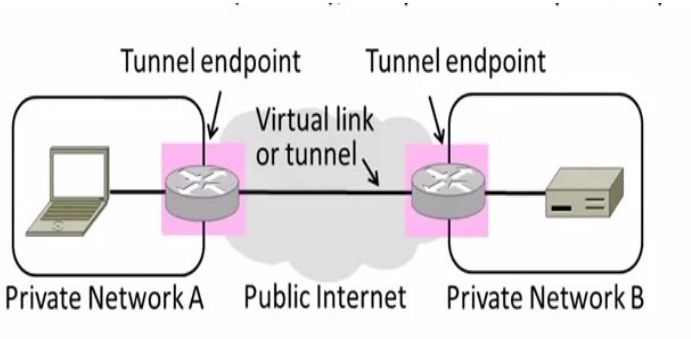
- **Fő jellemzői**
  - Mint közeli hálózat fut az interneten keresztül.
  - IPSEC-et használ az üzenetek titkosítására.
- Azaz informálisan megfogalmazva fizikailag távol lévő hosztok egy közös logikai egységet alkotnak.
  - Például távollévő telephelyek rendszerei.
- **Alapelv**
  - Bérelt vonalak helyett használjuk a publikusan hozzáférhető Internet-et.
  - Így az Internettől **logikailag** elkülöníthető hálózatot kapunk. Ezek a virtuális magánhálózatok avagy VPN-ek.
  - A célok közé kell felvenni a külső támadó kizárását.

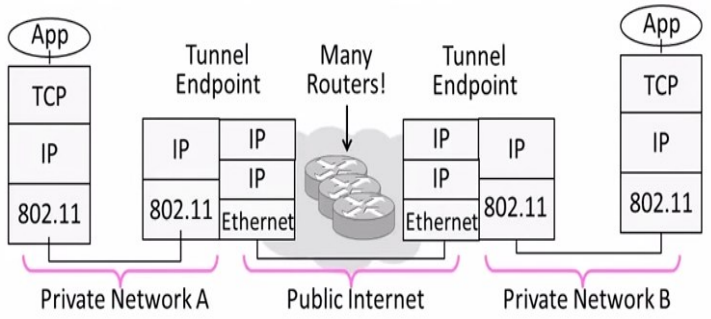


# Virtuális magánhálózatok alapok

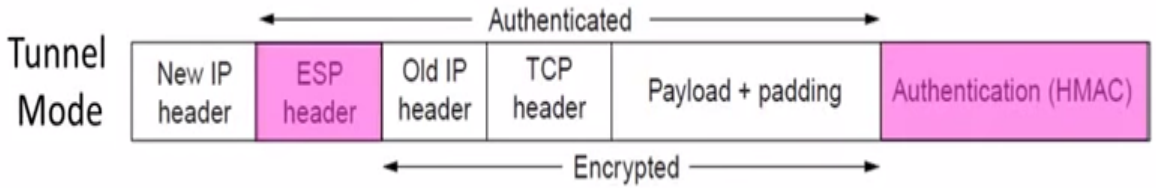
- A virtuális linkeket alagutak képzésével valósítjuk meg.
- **ALAGÚTAK**
  - Egy magánhálózaton belül a hosztok egymásnak normál módon küldhetnek üzenetet.
  - Virtuális linken a végpontok beágyazzák a csomagokat.
    - IP az IP-be mechanizmus.
- Az alagutak képzése önmagában kevés a védelemhez. Mik a hiányosságok?
  - Bizalmasság, authentikáció
  - Egy támadó olvashat, küldhet üzeneteket.
  - *Válasz:* Kriptográfia használata.



# Virtuális magánhálózatok alapok

- **IPSEC**
  - Hosszú távú célja az IP réteg biztonságossá tétele. (bizalmasság, autentikáció)
  - <u>Műveletei:</u>
    - Hoszt párok kommunikációjához kulcsokat állít be.
    - A kommunikáció kapcsolatorientáltabbá tétele.
    - Fejlécek és láblécek hozzáadása az IP csomagok védelme érdekében.
  - Több módot is támogat, amelyek közül az egyik az **alagút mód**.



# Szállítói réteg




- Feladat:
  - Adatfolyamok demultiplexálása
- További lehetséges feladatok:
  - Hosszú élettartamú kapcsolatok
  - Megbízható, sorrendhelyes csomag leszállítás
  - Hiba detektálás
  - Folyam és torlódás vezérlés
- Kihívások:
  - Torlódások detektálása és kezelése
  - Fairség és csatorna kihasználás közötti egyensúly

# Kivonat

- UDP
- TCP
- Torlódás vezérlés
- TCP evolúciója
- A TCP problémái

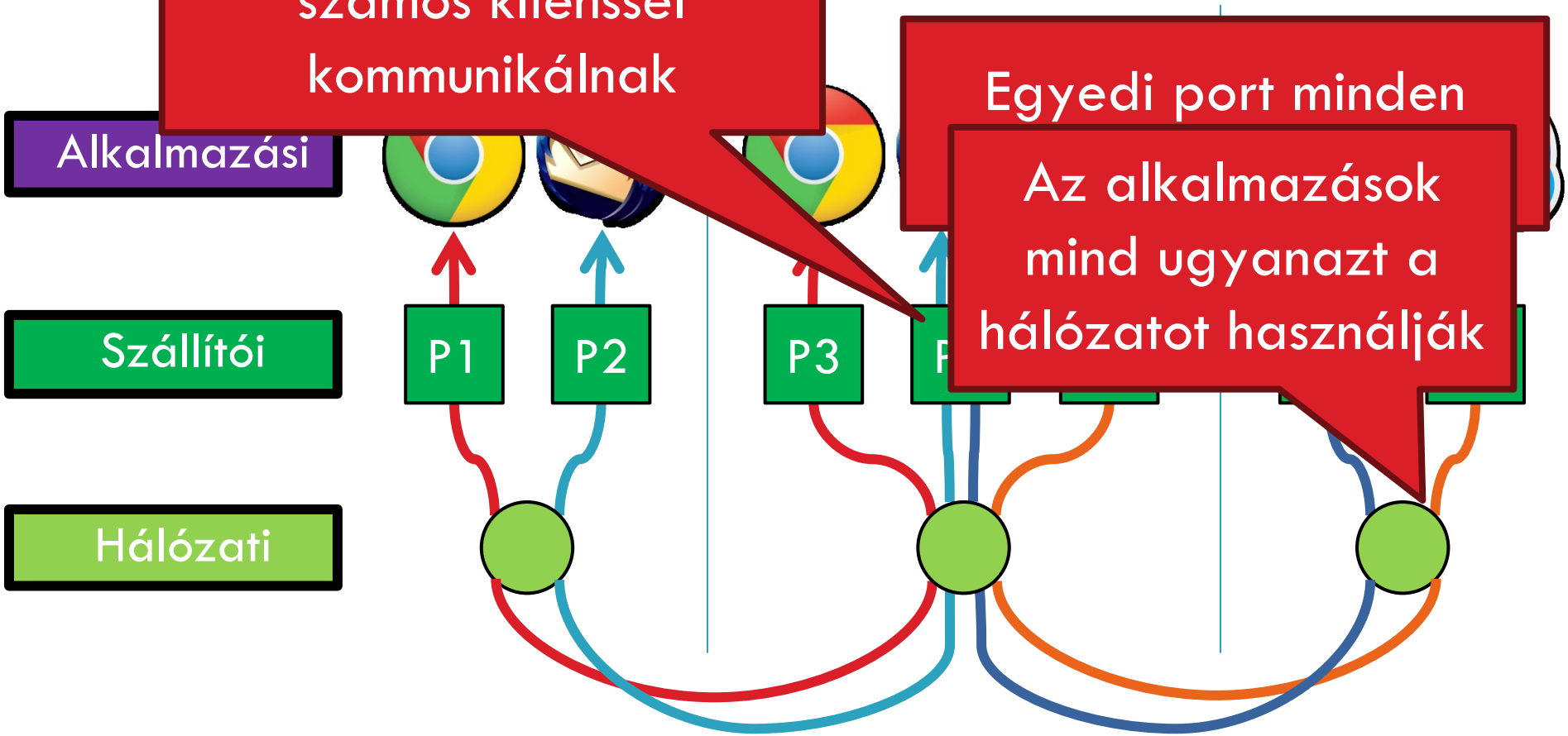
# Multiplexálás



- Datagram hálózat
  - Nincs áramkör kapcsolás
  - Nincs kapcsolat
- A kliensek számos alkalmazást futtathatnak egyidőben
  - Kinek szállítsuk le a csomagot?
- IP fejléc “protokoll” mezője
  - 8 bit = 256 konkurens folyam
  - Ez nem elég…
- Demultiplexálás megoldása a szállítói réteg feladata

# Forgalom demultiplexálása






Végpontok azonosítása: <*src_ip, src_port, dest_ip, dest_port, proto*>

*ahol src_ip, dst_ip a forrás és cél IP cím,*

*src_port, dest_port forrás és cél port, proto pedig UDP vagy TCP.*

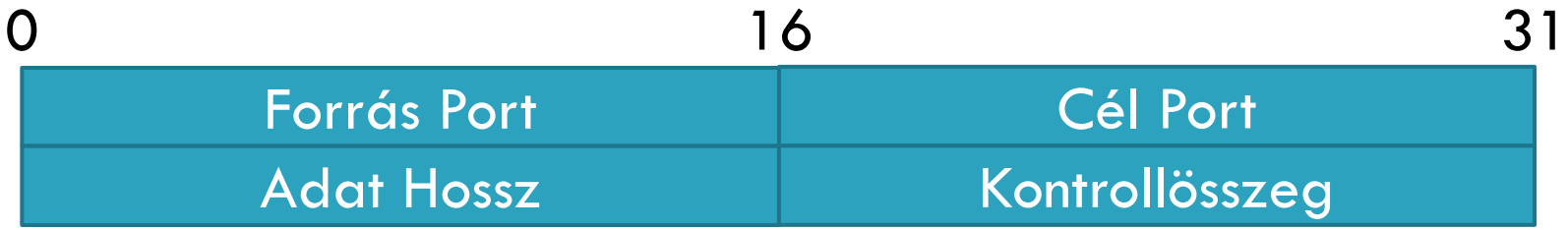
# Réteg modellek




- A legalacsonyabb szintű végpont-végpont protokoll
  - A szállítói réteg fejlécei csak a forrás és cél végpontok olvassák
  - A routerek számára a szállítói réteg fejléce csak szállítandó adat (payload)

# User Datagram Protocol (UDP)






- 8 bájtos UDP fejléc
- Egyszerű, kapcsolatnélküli átvitel
  - C socketek: SOCK_DGRAM
- Port számok teszik lehetővé a demultiplexálást
  - 16 bit = 65535 lehetséges port
  - 0 port nem engedélyezett
- Kontrollösszeg hiba detektáláshoz
  - Hibás csomagok felismerése
  - Nem detektálja az elveszett, duplikátum és helytelen sorrendben beérkező csomagokat (UDP esetén nincs ezekre garancia)

# UDP felhasználások

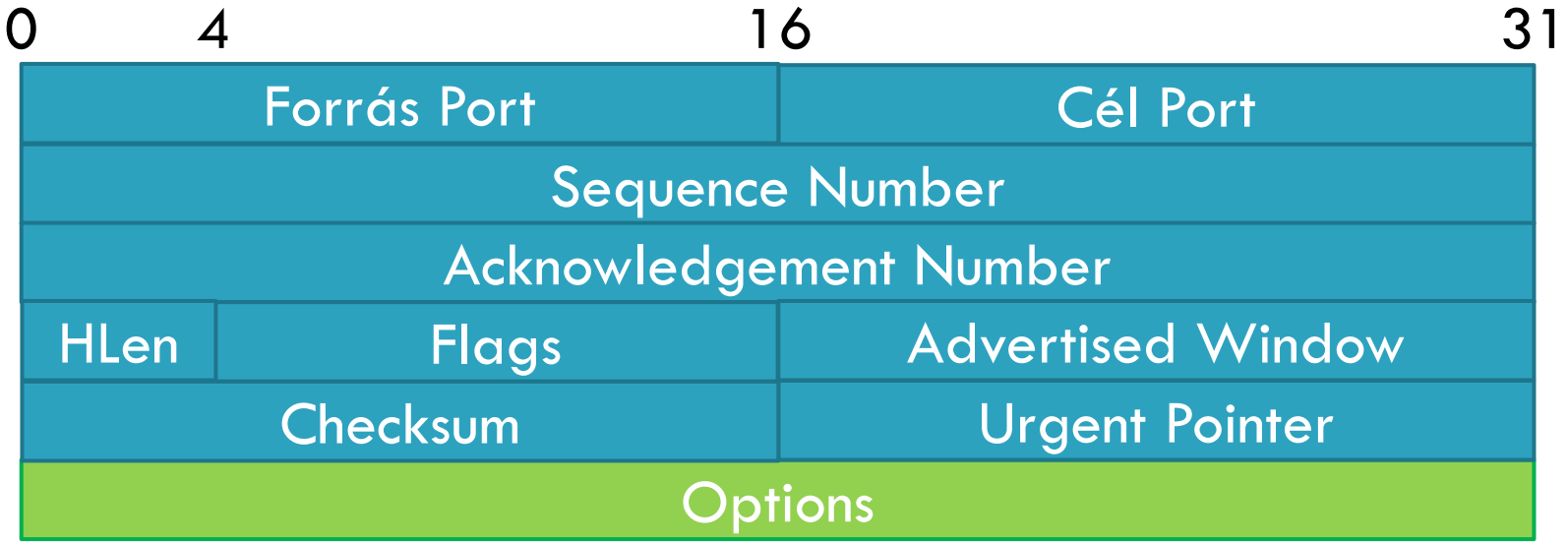


- A TCP után vezették be
  - Miért?
- Nem minden alkalmazásnak megfelelő a TCP
- UDP felett egyedi protokollok valósíthatók meg
  - Megbízhatóság? Helyes sorrend?
  - Folyam vezérlés? Torlódás vezérlés?
- Példák
  - RTMP, real-time média streamelés (pl. hang, video)
  - Facebook datacenter protocol

# Transmission Control Protocol



- Megbízható, sorrend helyes, két irányú bájt folyamok
  - Port számok a demultiplexáláshoz
  - Kapcsolat alapú
  - Folyam vezérlés
  - Torlódás vezérlés, fair viselkedés
- 20 bájtos fejléc + options fejlécek



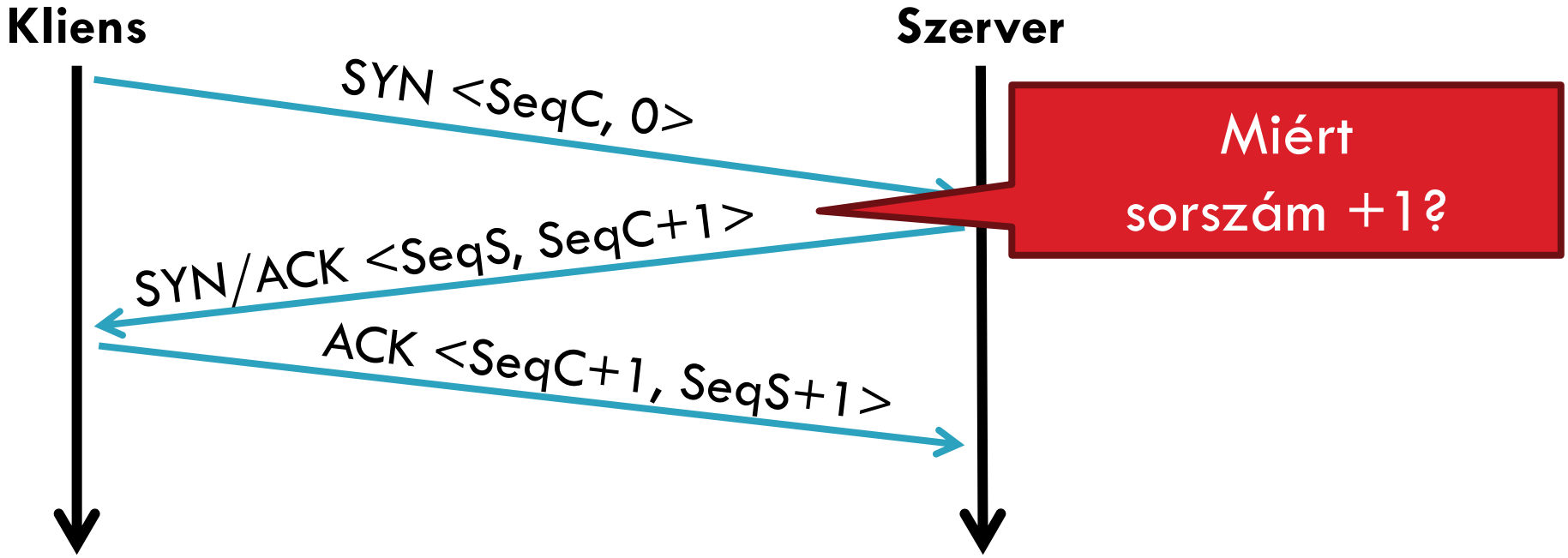
# Kapcsolat felépítés



- Miért van szükség kapcsolat felépítésre?
  - Állapot kialakítása mindkét végponton
  - Legfontosabb állapot: sorszámok/sequence numbers
    - Az elküldött bájtok számának nyilvántartása
    - Véletlenszerű kezdeti érték
- Fontos TCP flag-ek/jelölő bitek (1 bites)
  - SYN – szinkronizációs, kapcsolat felépítéshez
  - ACK – fogadott adat nyugtázása
  - FIN – vége, kapcsolat lezárásához

# Three Way Handshake
# Három-utas kézfogás






- Mindkét oldalon:
  - Másik fél értesítése a kezdő sorszámról
  - A másik fél kezdő sorszámának nyugtázása

# Kapcsolat felépítés problémája

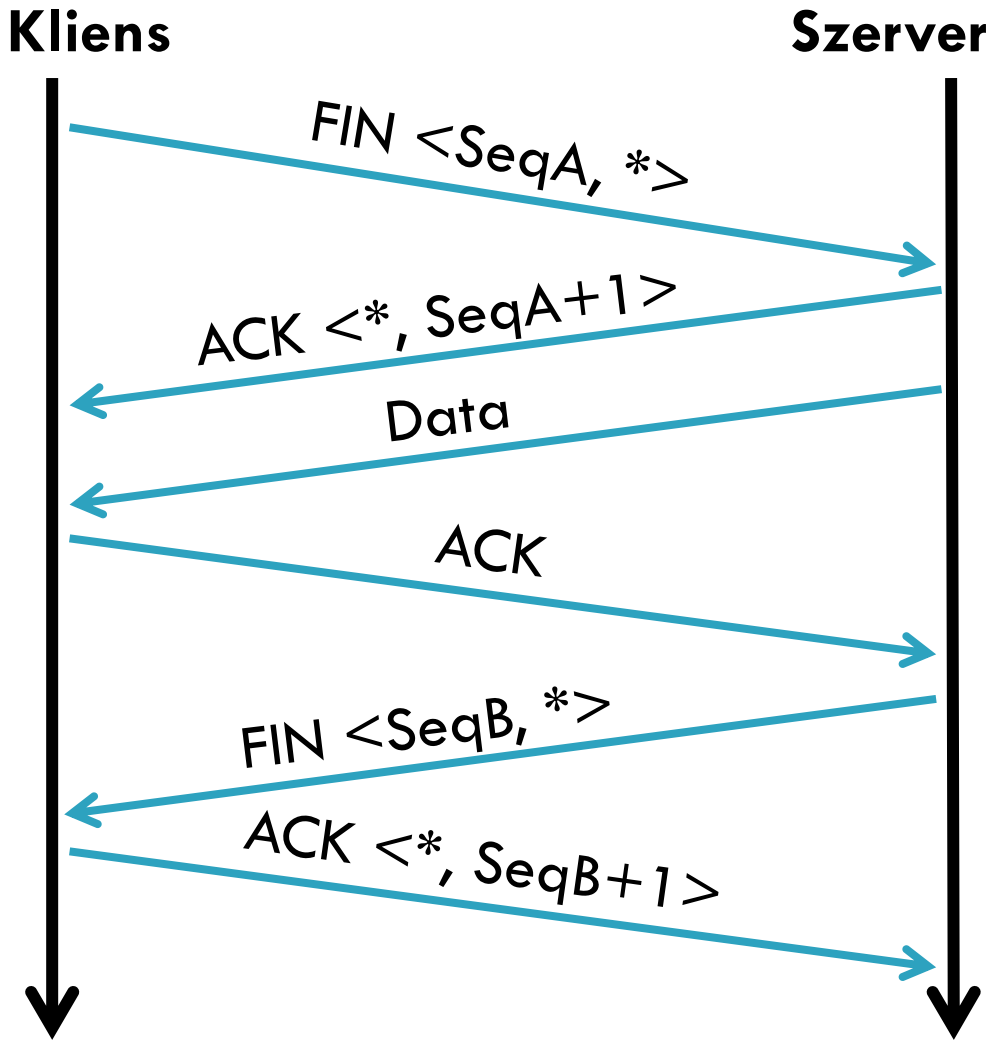


- Kapcsolódási zűrzavar
  - Azonos hoszt kapcsolatainak egyértelműsítése
  - Véletlenszerű sorszámmal - biztonság
- Forrás hamisítás
  - Kevin Mitnick
  - Jó random szám generátor kell hozzá!
- Kapcsolat állapotának kezelése
  - Minden SYN állapotot foglal a szerveren
  - SYN flood = denial of service (DoS) támadás
  - Megoldás: SYN cookies

# Kapcsolat lezárása



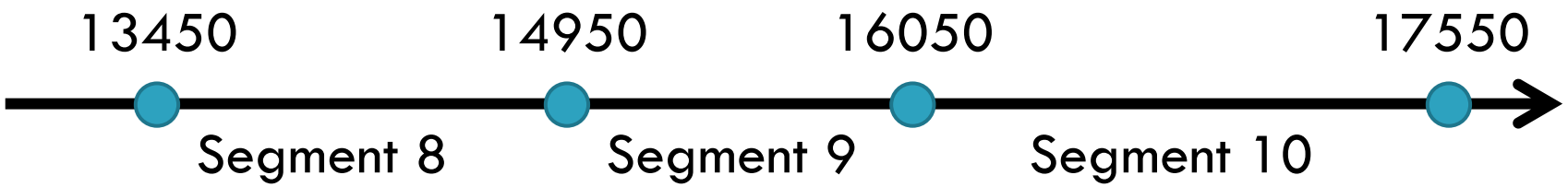
- Mindkét oldal kezdeményezheti a kapcsolat bontását
- A másik oldal még folytathatja a küldést
  - Félig nyitott kapcsolat
  - *shutdown()*
- Az utolsó FIN nyugtázása
  - Sorszám + 1
- Mi történik, ha a 2. FIN elveszik?



# Sorszámok tere

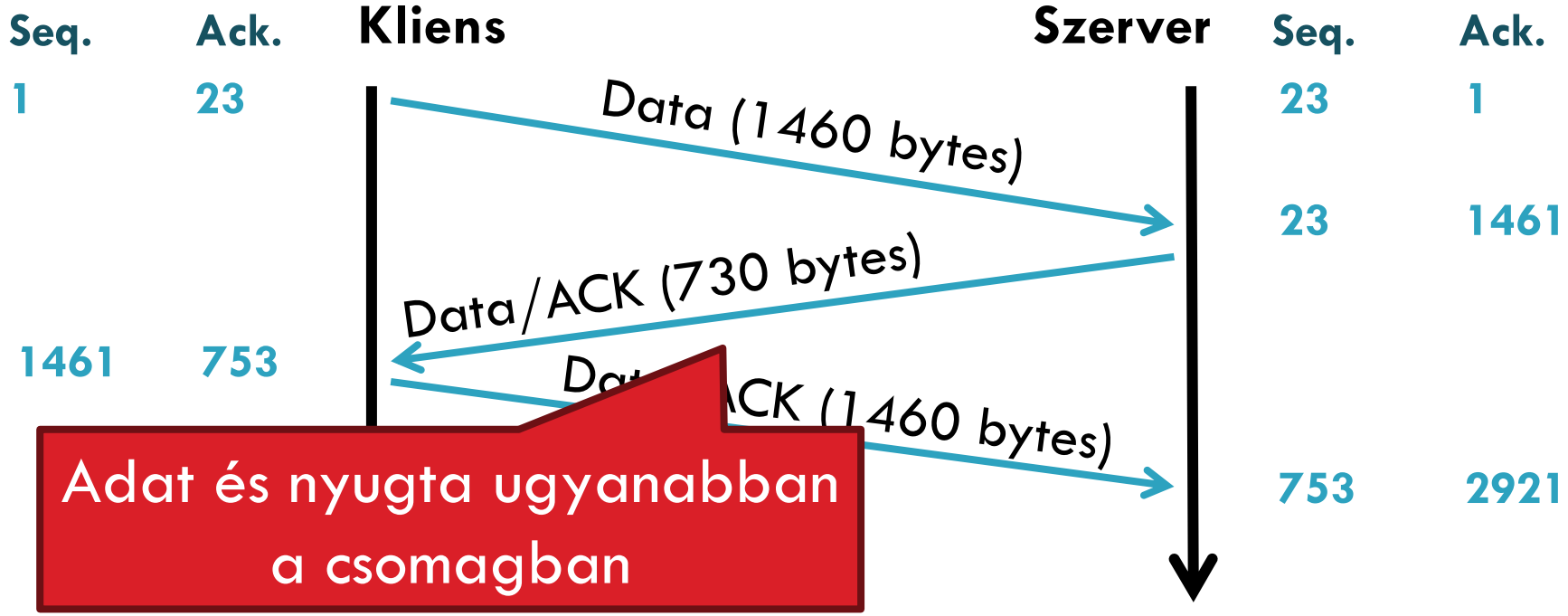


- A TCP egy absztrakt bájt folyamot valósít meg
  - A folyam minden bájtja számozott
  - 32-bites érték, körbefordul egy idő után
  - Kezdetben, véletlen érték a kapcsolat felépítésénél.
- A bájt folyamot szegmensekre bontjuk (TCP csomag)
  - A méretét behatárolja a Maximum Segment Size (MSS)
  - Úgy kell beállítani, hogy elkerüljük a fregmentációt
- Minden szegmens egyedi sorszámmal rendelkezik



# Kétirányú kapcsolat






- Mindkét fél küldhet és fogadhat adatot
  - Különböző sorszámok a két irányba

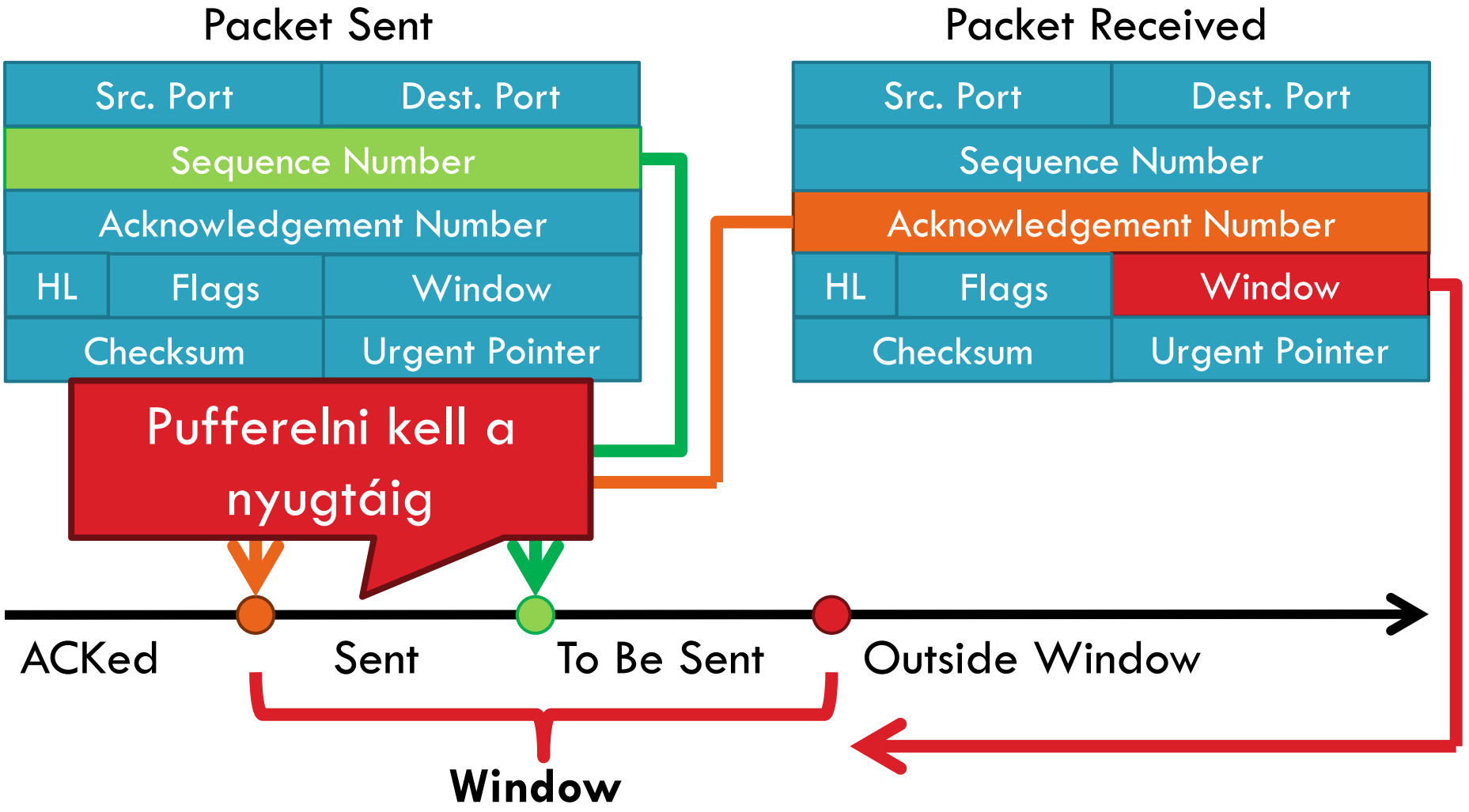
# Folyam vezérlés



- Probléma: Hány csomagot tud a küldő átvinni?
  - Túl sok csomag túlterhelheti a fogadót
  - A fogadó oldali puffer-méret változhat a kapcsolat során
- Megoldás: csúszóablak
  - A fogadó elküldi a küldőnek a pufferének méretét
  - Ezt nevezzük meghirdetett ablaknak: advertised window
  - Egy *n* ablakmérethez, a küldő n bájtot küldhet el ACK fogadása nélkül
  - Minden egyes ACK  után, léptetjük a csúszóablakot
- Az ablak akár nulla is lehet!

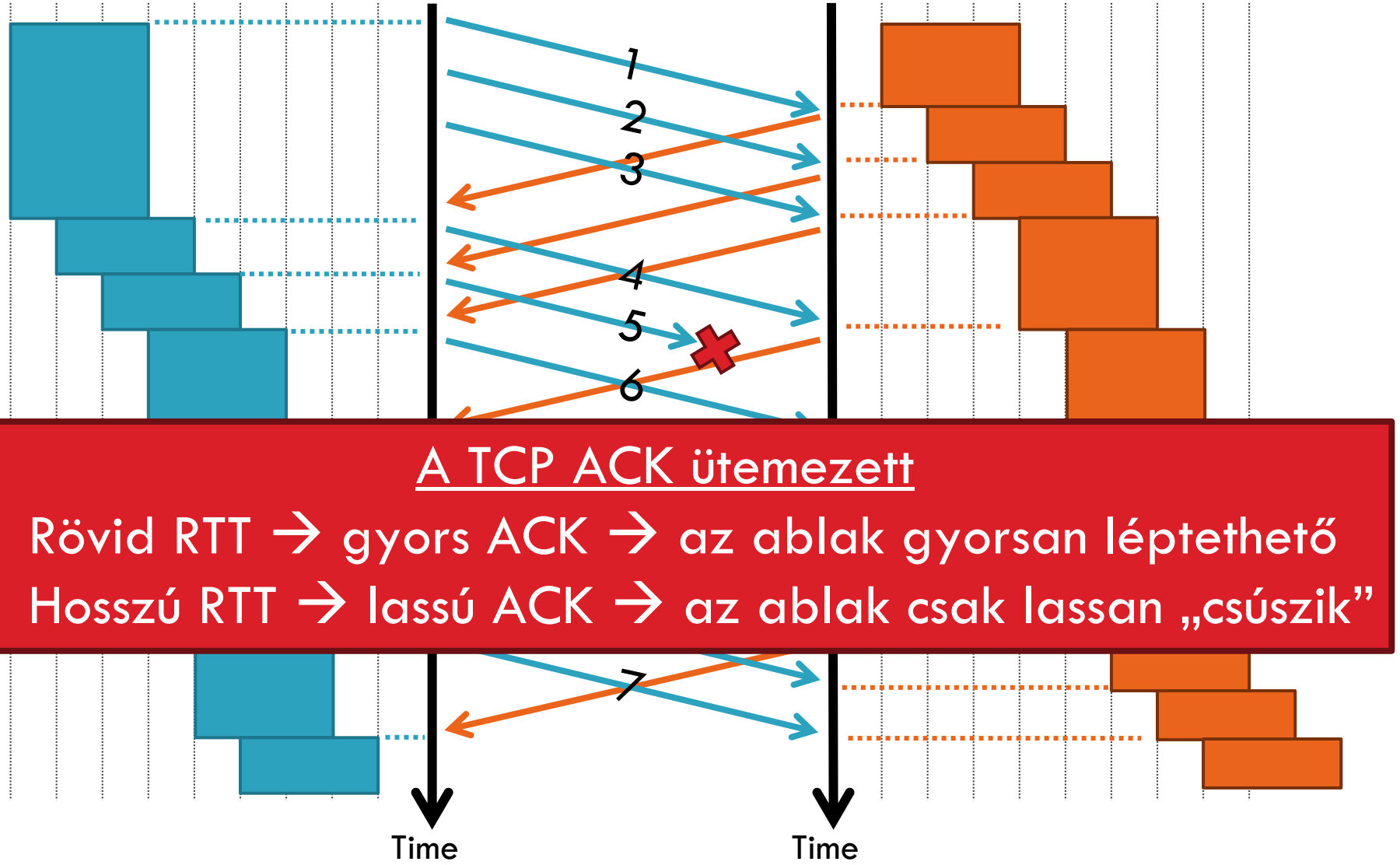
# Folyam vezérlés - csúszóablak





# Csúszóablak példa






**A TCP ACK ütemezett**

- Rövid RTT → gyors ACK → az ablak gyorsan léptethető
- Hosszú RTT → lassú ACK → az ablak csak lassan „csúszik”

# Megfigyelések



- Átvitel arányos ~ w/RTT
  - w: küldési ablakméret
  - RTT: körülfordulási idő
- A küldőnek pufferelni kell a nem nyugtázott csomagokat a lehetséges újraküldések miatt
- A fogadó elfogadhat nem sorrendben érkező csomagokat, de csak amíg az elfér a pufferben

# Mit nyugtázhat a fogadó?



1. Minden egyes csomagot
2. Használhat *kumulált nyugtát*, ahol egy n sorszámú nyugta minden k<n sorszámú csomagot nyugtáz
3. Használhat *negatív nyugtát* (NACK), megjelölve, hogy mely csomag nem érkezett meg
4. Használhat *szelektív nyugtát* (SACK), jelezve, hogy mely csomagok érkeztek meg, akár nem megfelelő sorrendben

- SACK egy TCP kiterjesztés
  - SACK TCP

# Sorszámok



- 32 bites, unsigned
  - Miért ilyen nagy?
- A csúszó-ablakhoz szükséges…
  - |sorszámok tere| > 2 * |Küldő ablak mérete|
  - $2^{32} > 2 * 2^{16}$
- Elkóborolt csomagok kivédése
  - IP csomagok esetén a maximális élettartam (MSL) of 120 mp
    - Azaz egy csomag 2 percig bolyonghat egy hálózatban

# Buta ablak szindróma



- Mi van, ha az ablak mérete nagyon kicsi?
  - Sok, apró csomag. A fejlécek dominálják az átvitelt.


- Lényegében olyan, mintha bájtonként küldenénk az üzenetet…

```
1. for (int x = 0; x < strlen(data); ++x)
2.   write(socket, data + x, 1);
```

# Nagle algoritmusa



1. Ha az ablak >= MSS és az elérhető adat >= MSS:

   Küldjük el az adatot

   Egy teljes csomag küldése

2. Különben ha van nem nyugtázott adat::

   Várakoztassuk az adatot egy pufferben, amíg nyugtát nem kapunk

3. Különben: küldjük az adatot

   Küldjünk egy nem teljes csomagot, ha nincs más

- Probléma: Nagle algoritmusa késlelteti az átvitelt
  - Mi van, ha azonnal el kell küldeni egy csomagot?

```
1. int flag = 1;
2. setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
   (char *) &flag, sizeof(int));
```

# Hiba detektálás

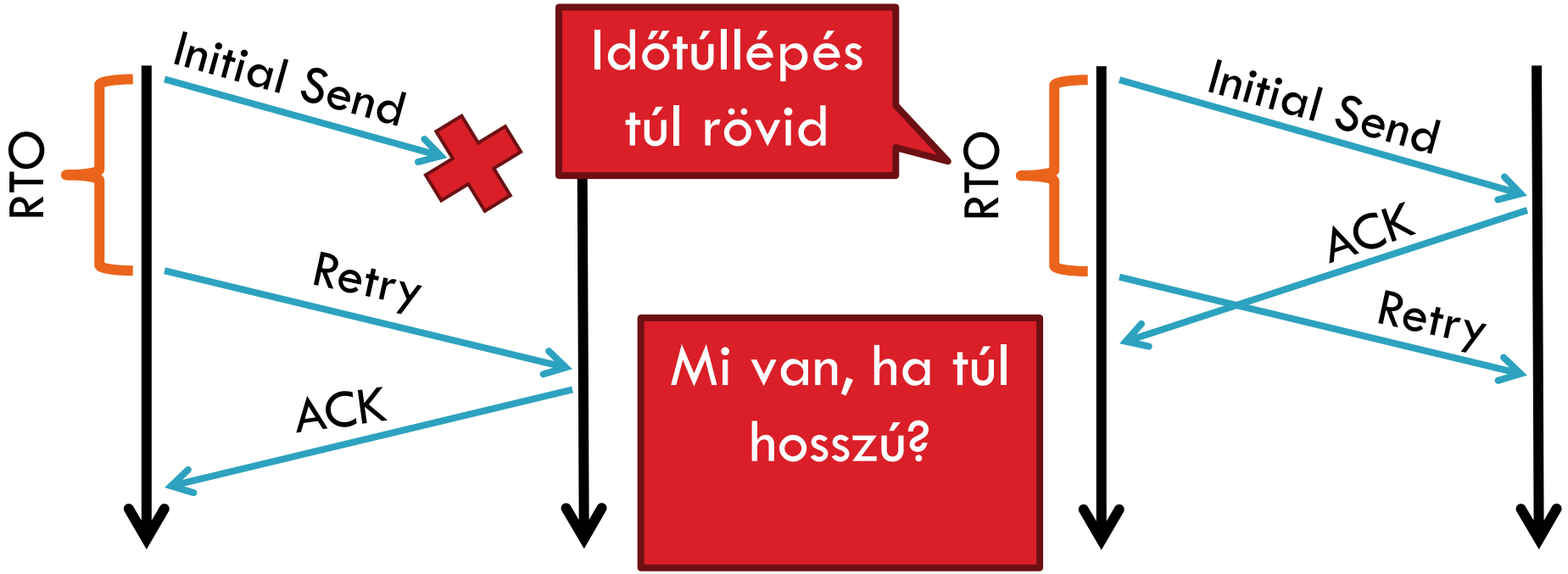


- A kontrollösszeg detektálja a hibás csomagokat
  - Az IP, TCP fejlécből és az adatból számoljuk
- A sorszámok segítenek a sorrendhelyes átvitelben
  - Duplikátumok eldobása
  - Helytelen sorrendben érkező csomagok sorba rendezése vagy eldobása
  - Hiányzó sorszámok elveszett csomagot jeleznek
- A küldő oldalon: elveszett csomagok detektálása
  - Időtúllépés (timeout) használata hiányzó nyugtákhoz
  - Szükséges az RTT becslése a időtúllépés beállításához
  - Minden nem nyugtázott csomagot pufferelni kell a nyugtáig

# Retransmission Time Outs (RTO)
# Időtúllépés az újraküldéshez

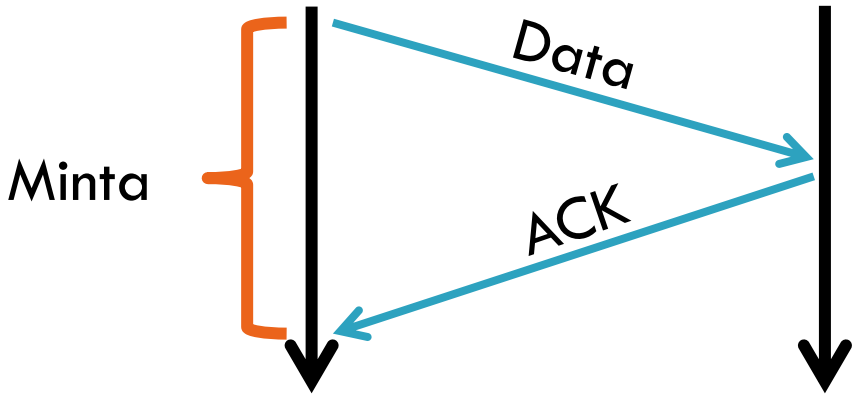


- Probléma: Időtúllépés RTT-hez kapcsolása



# Round Trip Time becslés

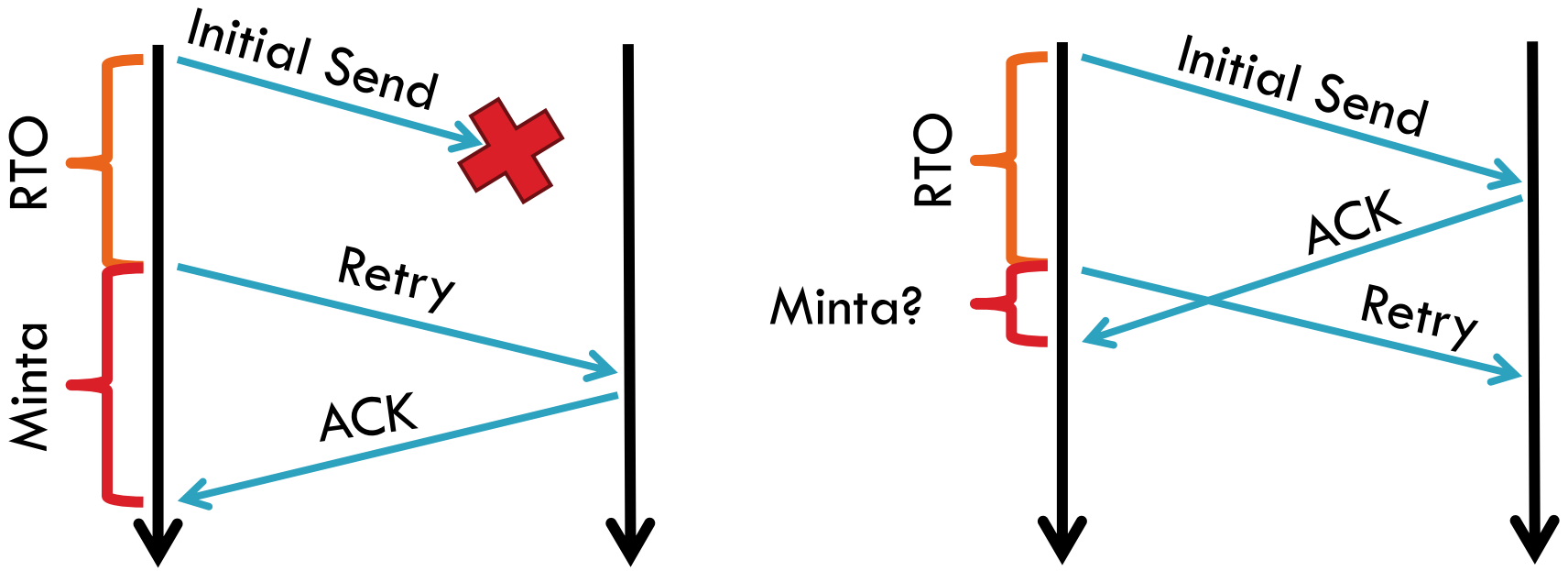





- Az eredeti TCP RTT becslője:
  - RTT becslése mozgó átlaggal
  - new_rtt = α (old_rtt) + (1 – α)(new_sample)
  - Javasolt α: 0.8-0.9 (0.875 a legtöbb TCP esetén)
- RTO = 2 * new_rtt (a TCP konzervatív becslése)

# Az RTT minta félre is értelmezhető





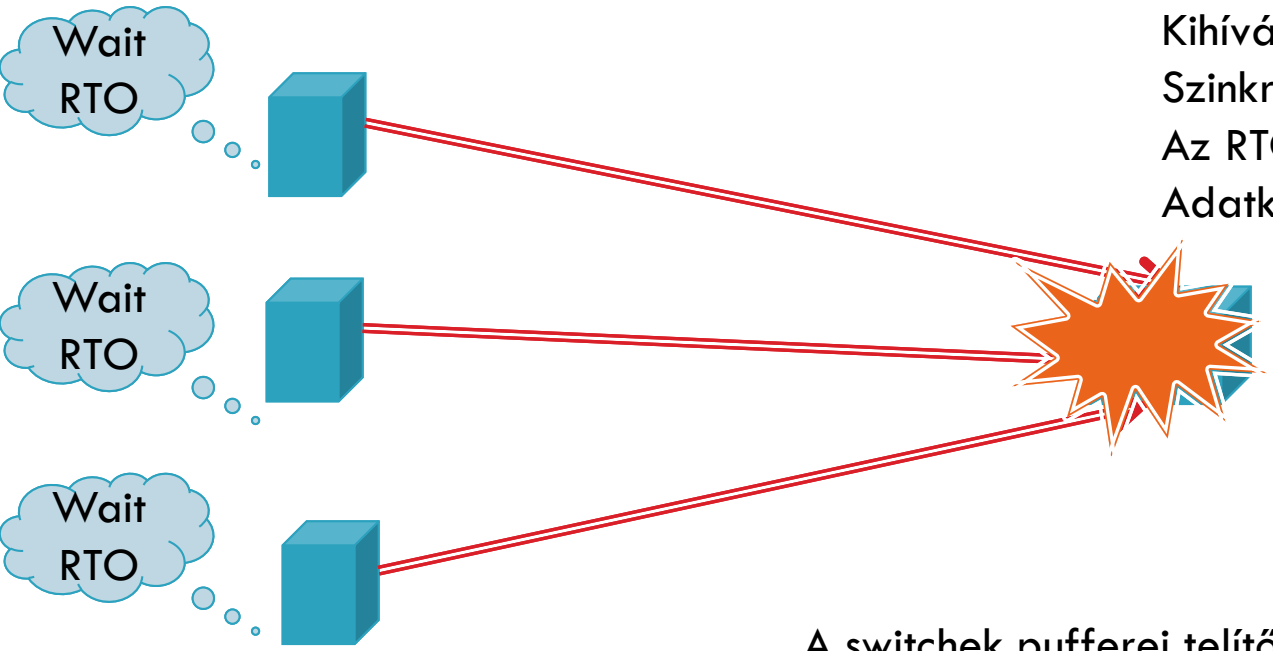

- **Karn algoritmusa**: dobjuk el azokat a mintákat, melyek egy csomag újraküldéséből származnak

# RTO adatközpontokban???



- TCP Incast probléma – pl. Hadoop, Map Reduce, HDFS, GFS

Sok szimultán küldő egy fogadóhoz




Kihívás:
Szinkronizáció megtörése
Az RTO becslést WAN-ra tervezték
Adatközpontban sokkal kisebb RTT van
1-2ms vagy kevesebb

A switchek pufferei telítődnek és csomagok vesznek el!
Nyugta nem megy vissza ☹

# Mi az a torlódás?

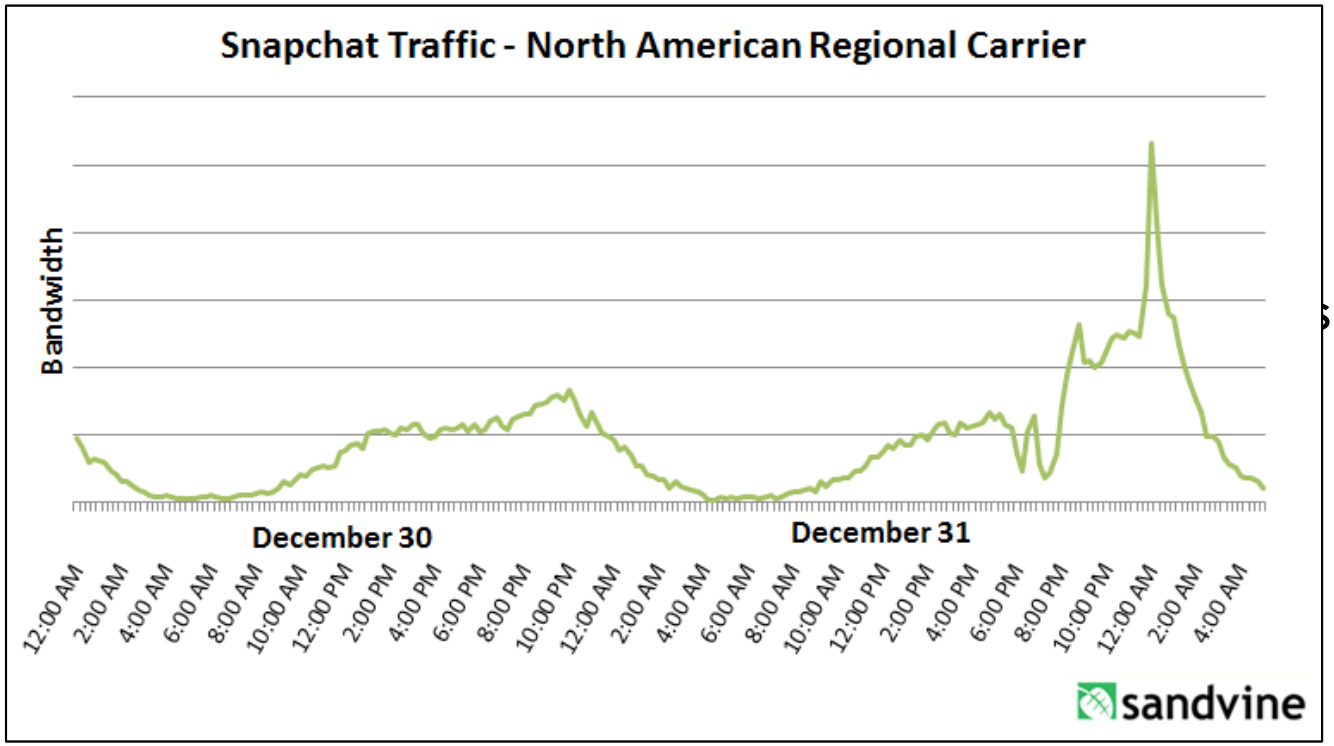


- A hálózat terhelése nagyobb, mint a kapacitása
  - A kapacitás nem egyenletes a hálózatban
    - Modem vs. Cellular vs. Cable vs. Fiber Optics
  - Számos folyam verseng a sávszélességért
    - otthoni kábel modem vs. corporate datacenter
  - A terhelés időben nem egyenletes
    - Vasárnap este 10:00 = Bittorrent Game of Thrones

# Mi az a torlódás?



- A hálózat terhelése nagyobb, mint a kapacitása
  - A kapacitás nem egyenletes a hálózatban
    - Modem vs. Cellular vs. Cable vs. Fiber Optics



# Miért rossz a torlódás?

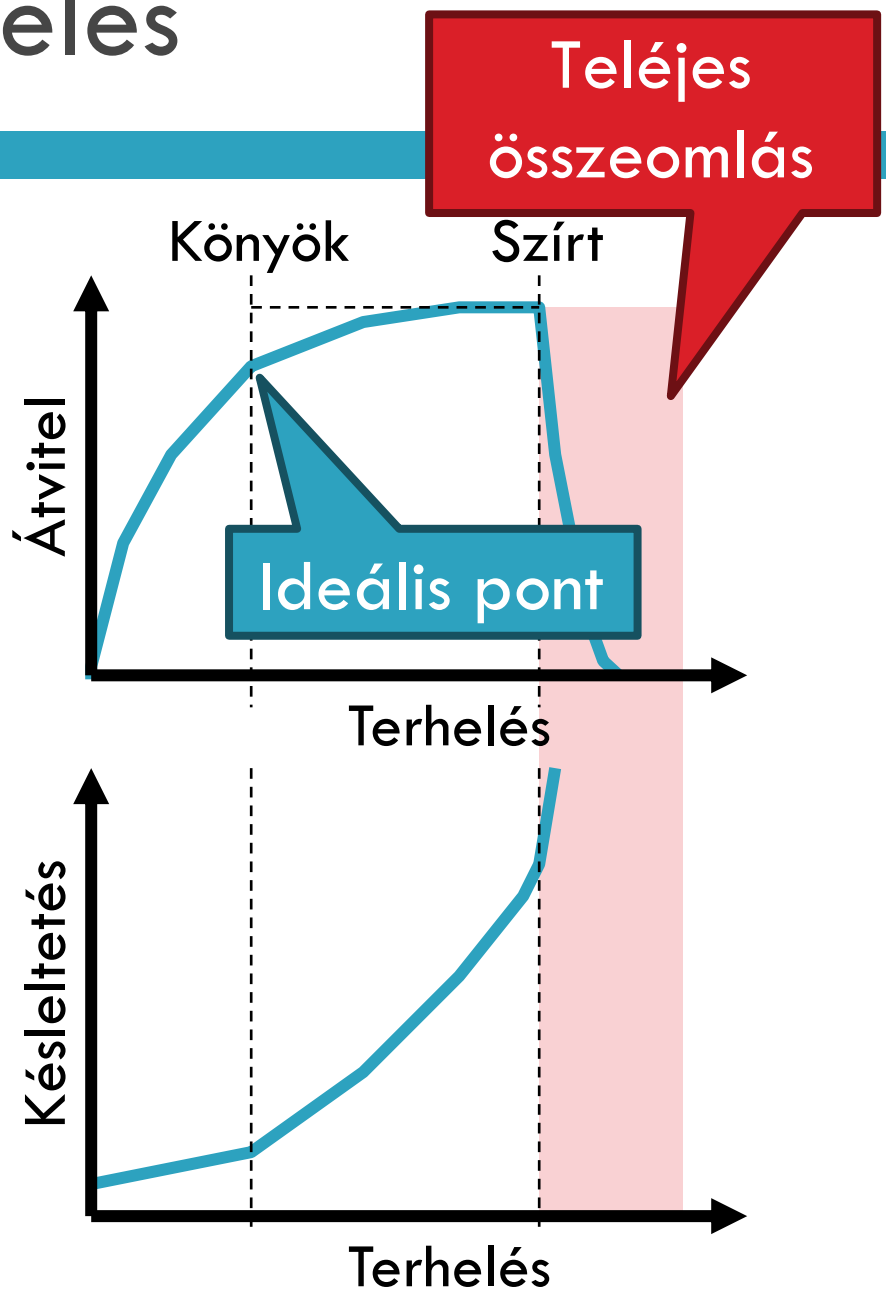


- Csomagvesztést eredményez
  - A routerek véges memóriával (puffer) rendelkeznek
  - Önhasonló Internet forgalom, nincs puffer, amiben ne okozna csomagvesztést
  - Ahogy a routerek puffere elkezd telítődni, csomagokat kezd eldobni… (RED)
- Gyakorlati következmények
  - A routerek sorai telítődnek, megnövekedett késleltetés
  - Sávszélesség pazarlása az újraküldések miatt
  - Alacsony hálózati átvitel (goodput)

# Megnövekedett terhelés



- Könyök („knee”)– a pont, ami után
  - Az átvitel szinte alig nő
  - Késleltetés viszont gyorsan emelkedik
- Egy egyszerű sorban (M/M/1)
  - Késleltetés = 1/(1 – utilization)
- Szírt („cliff”) – a pont, ami után
  - Átvitel lényegében leesik 0-ra
  - A késleltetés pedig → ∞

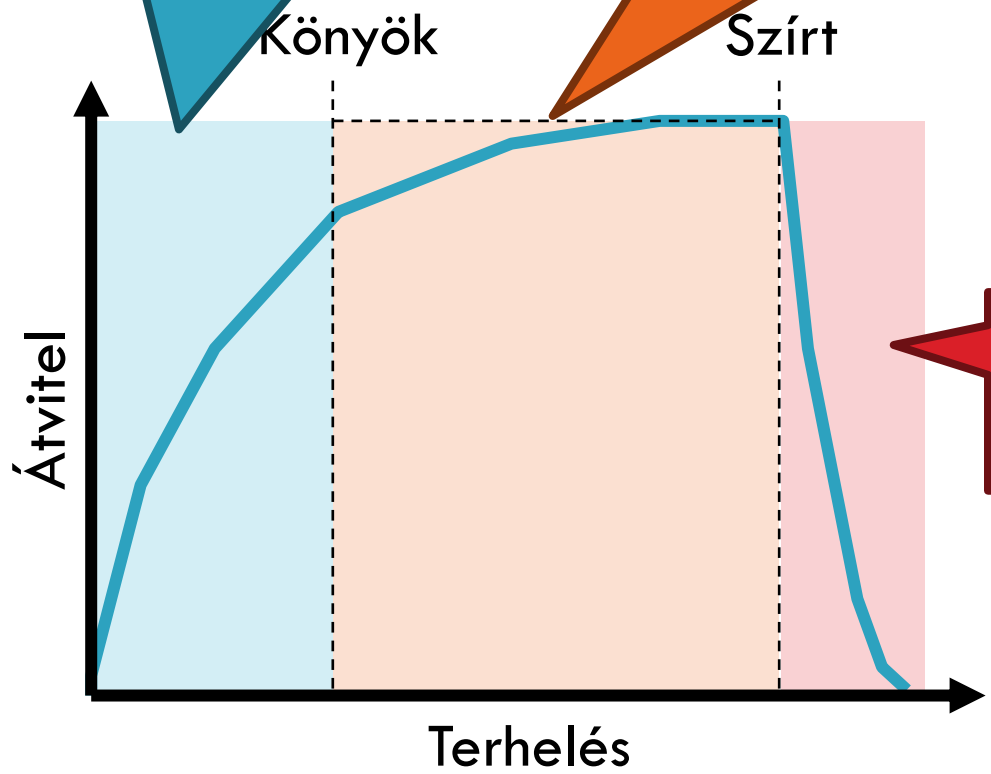


# Torlódás vezérlés vs torlódás elkerülés



Torlódás elkerülés:
Maradj a könyök bal oldalán

Torlódás vezérlés
Maradj a szírt bal oldalán




Teljes
összeomlás