Számítógépes Hálózatok

9. Előadás: ++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

További protokollok

Internet Control Message Protocol

FELADATA

- Váratlan események jelentése

HASZNÁLAT

- Többféle *ICMP*-üzenetet definiáltak:
  - Elérhetetlen cél;
  - Időtúllépés;
  - Paraméter probléma;
  - Forráslefojtás;
  - Visszhang kérés;
  - Visszhang válasz;
  - ...

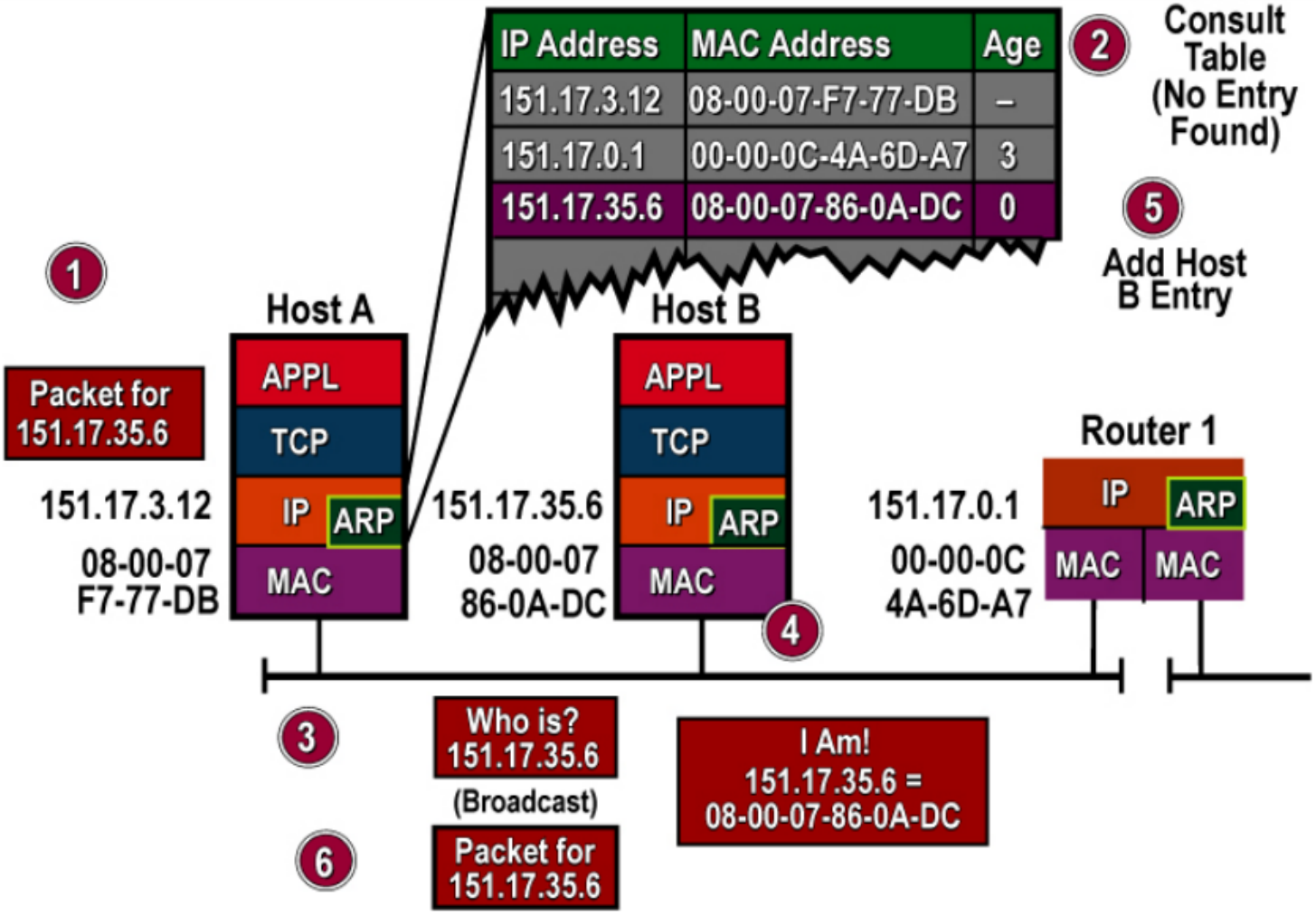
Internet Control Message Protocol

- *Elérhetetlen cél* esetén a csomag kézbesítése sikertelen volt.
  - **Esemény lehetséges oka:** Egy nem darabolható csomag továbbításának útvonalán egy „kis csomagos hálózat” van.
- *Időtúllépés* esetén az IP csomag élettartam mezője elérte a 0-át.
  - **Esemény lehetséges oka:** Torlódás miatt hurok alakult ki vagy a számláló értéke túl alacsony volt.
- *Paraméter probléma* esetén a fejrészben érvénytelen mezőt észleltünk.
  - **Esemény lehetséges oka:** Egy az útvonalon szereplő router vagy a hoszt IP szoftverének hibáját jelezheti.

Internet Control Message Protocol

- Forráslefojtás esetén lefojtó csomagot küldünk.
  - **Esemény hatása:** A fogadó állomásnak a forgalmazását lassítania kellett.
- Visszhang kérés esetén egy hálózati állomás jelenlétét lehet ellenőrizni.
  - **Esemény hatása:** A fogadónak vissza kell küldeni egy visszhang választ.
- Átirányítás esetén a csomag rosszul irányítottságát jelzik.
  - **Esemény kiváltó oka:** Router észleli, hogy a csomag nem az optimális útvonall.

Address Resolution Protocol



Address Resolution Protocol

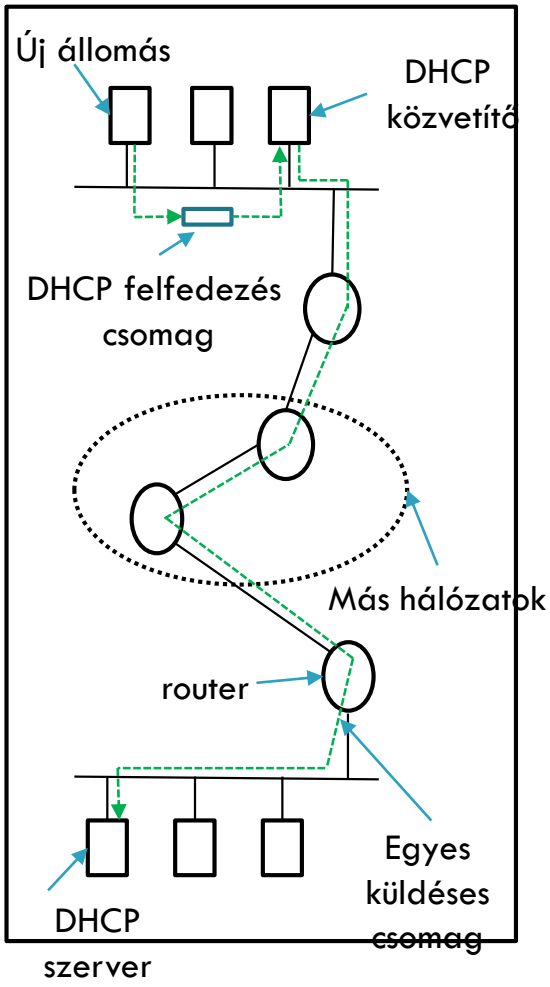
FELADATA

- Az IP cím megfeleltetése egy fizikai címnek.

HOZZÁRENDELÉS

- Adatszóró csomag kiküldése az *Ethernet*re „Ki-é a 192.60.34.12-es IP-cím?” kérdéssel az alhálózaton, és mindenegyes hoszt ellenőrzi, hogy övé-e a kérdéses IP-cím. Ha egyezik az IP a hoszt saját IP-jével, akkor a saját *Ethernet* címével válaszol. Erre szolgál az ARP.
- Opcionális javítási lehetőségek:
  - a fizikai cím IP hozzárendelések tárolása (*cache használata*);
  - Leképezések megváltoztathatósága (*időhatály bevezetése*);
- Mi történik távoli hálózaton lévő hoszt esetén?
  - A router is válaszoljon az ARP-re a hoszt alhálózatán. (*proxy ARP*)
  - Alapértelmezett Ethernet-cím használata az összes távoli forgalomhoz

Reverse Address Resolution Protocol



Reverse Address Resolution Protocol

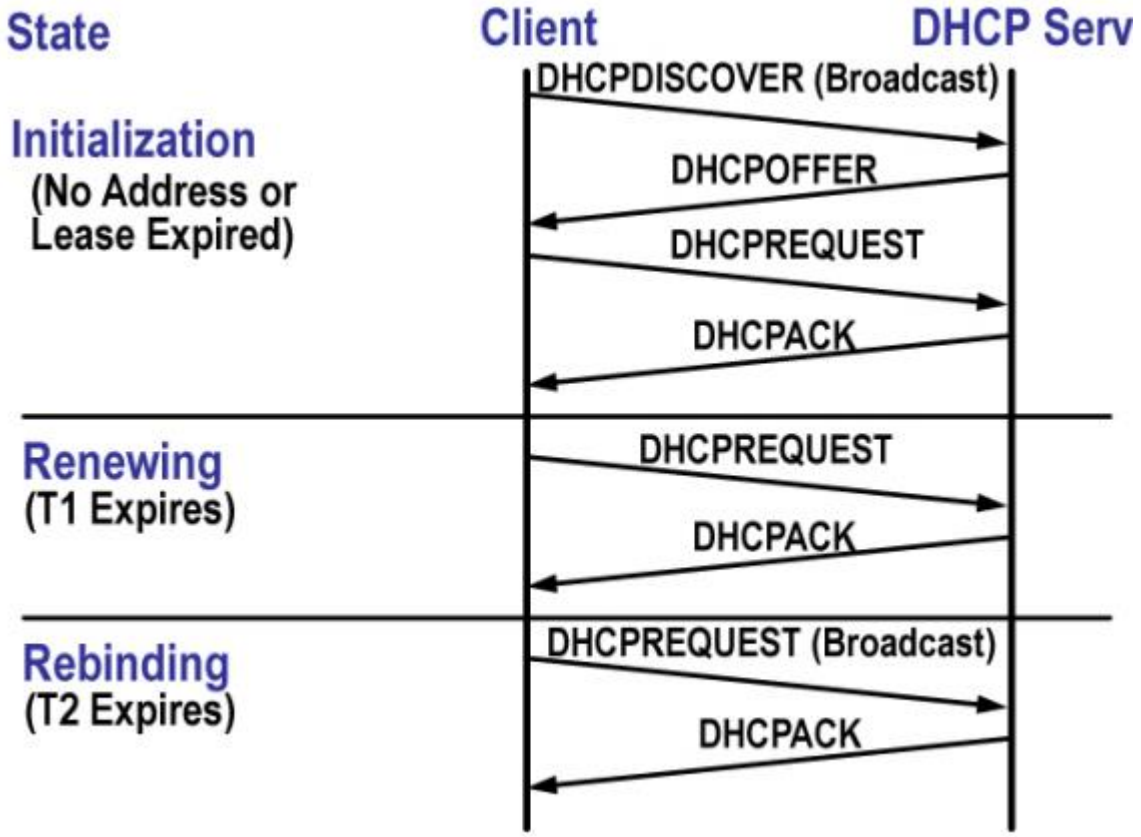
FELADATA

- A fizikai cím megfeleltetése egy IP címnek

HOZZÁRENDELÉS

- Az újonnan indított állomás adatszórással csomagot küld ki az *Ethernet*re „A 48-bites Ethernet-címem 14.04.05.18.01.25. Tudja valaki az IP címemet?” kérdéssel az alhálózaton. Az *RARP*-szerver pedig válaszol a megfelelő IP címmel, mikor meglátja a kérést
- Opcionális javítási lehetőségek:
  - *BOOTP* protokoll használata. UDP csomagok használata. Manuálisan kell a hozzárendelési táblázatot karbantartani. (statikus címkiosztás)
  - *DHCP* protokoll használata. Itt is külön kiszolgáló osztja ki a címeket a kérések alapján. A kiszolgáló és a kérő állomások nem kell hogy ugyanazon a *LAN*-on legyenek, ezért *LAN*-onként kell egy *DHCP relay agent*. (statikus és dinamikus címkiosztás)

DHCP: DYNAMIC HOST CONFIGURATION PROTOCOL



DHCP

- Lényegében ez már az **Alkalmazási réteg**
  - de logikailag ide tartozik

- Segítségével a hosztok automatikusan juthatnak hozzá a kommunikációjukhoz szükséges hálózati azonosítókhoz:
  - IP cím, hálózati maszk, alapértelmezett átjáró, stb.

- Eredetileg az RFC 1531 a BOOTP kiterjesztéseként definiálta. Újabb RFC-k: 1541, 2131 (aktuális)

DHCP lehetőségei

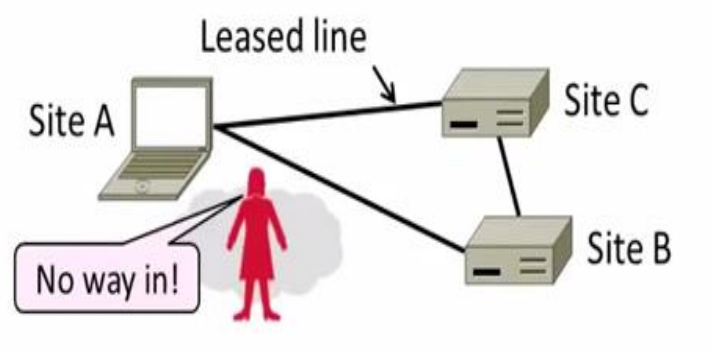
- IP címek osztása MAC cím alapján DHCP szerverrel
  - Szükség esetén (a DHCP szerveren előre beállított módon) egyes kliensek számára azok MAC céméhez fix IP cím rendelhető
- IP címek osztása dinamikusan
  - A DHCP szerveren beállított tartományból „érkezési sorrendben” kapják a kliensek az IP címeket
  - Elegendő annyi IP cím, ahány gép egyidejűleg működik
- Az IP címeken kívül további szükséges hálózati paraméterek is kioszthatók
  - Hálózati maszk
  - Alapértelmezett átjáró
  - Névkiszolgáló
  - Domain név
  - Hálózati rendszerbetöltéshez szerver és fájlnév

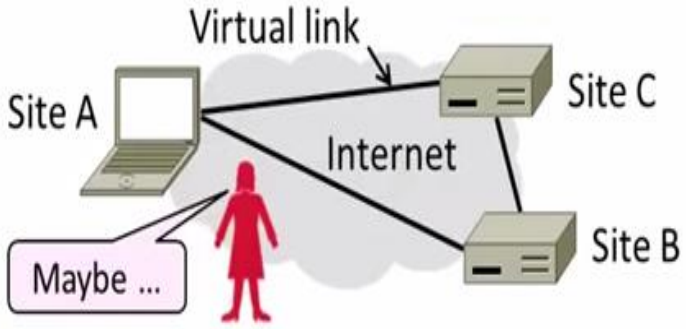
DHCP – Címek bérlése

- A DHCP szerver a klienseknek az IP-címeket bizonyos bérleti időtartamra (lease time) adja „bérbe”
  - Az időtartam hosszánál a szerver figyelembe veszi a kliens esetleges ilyen irányú kérését
  - Az időtartam hosszát a szerver beállításai korlátozzák
- A bérleti időtartam lejárta előtt a bérlet meghosszabbítható
- Az IP-cím explicit módon vissza is adható

Virtuális magánhálózatok alapok

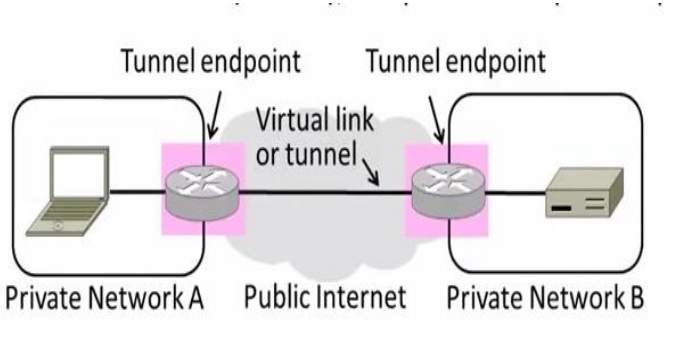
- **Fő jellemzői**
  - Mint közeli hálózat fut az interneten keresztül.
  - IPSEC-et használ az üzenetek titkosítására.
- Azaz informálisan megfogalmazva fizikailag távol lévő hosztok egy közös logikai egységet alkotnak.
  - Például távollévő telephelyek rendszerei.
- **Alapelv**
  - Bérelt vonalak helyett 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.

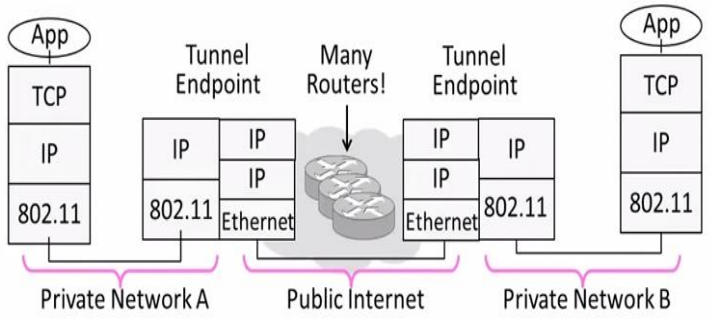




Virtuális magánhálózatok alapok

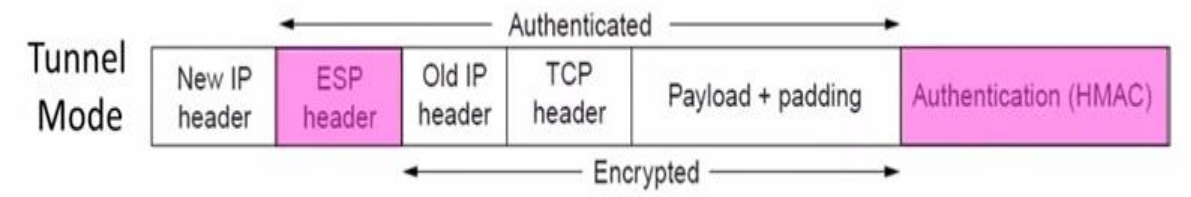
- A virtuális linkeket alagutak képzésével valósítjuk meg.
- **ALAGÚTAK**
  - Egy magánhálózaton belül a hosztok egymásnak normál módon küldhetnek üzenetet.
  - Virtuális linken a végpontok beágyazzák a csomagokat.
    - IP az IP-be mechanizmus.
- Az alagutak képzése önmagában kevés a védelemhez. Mik 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

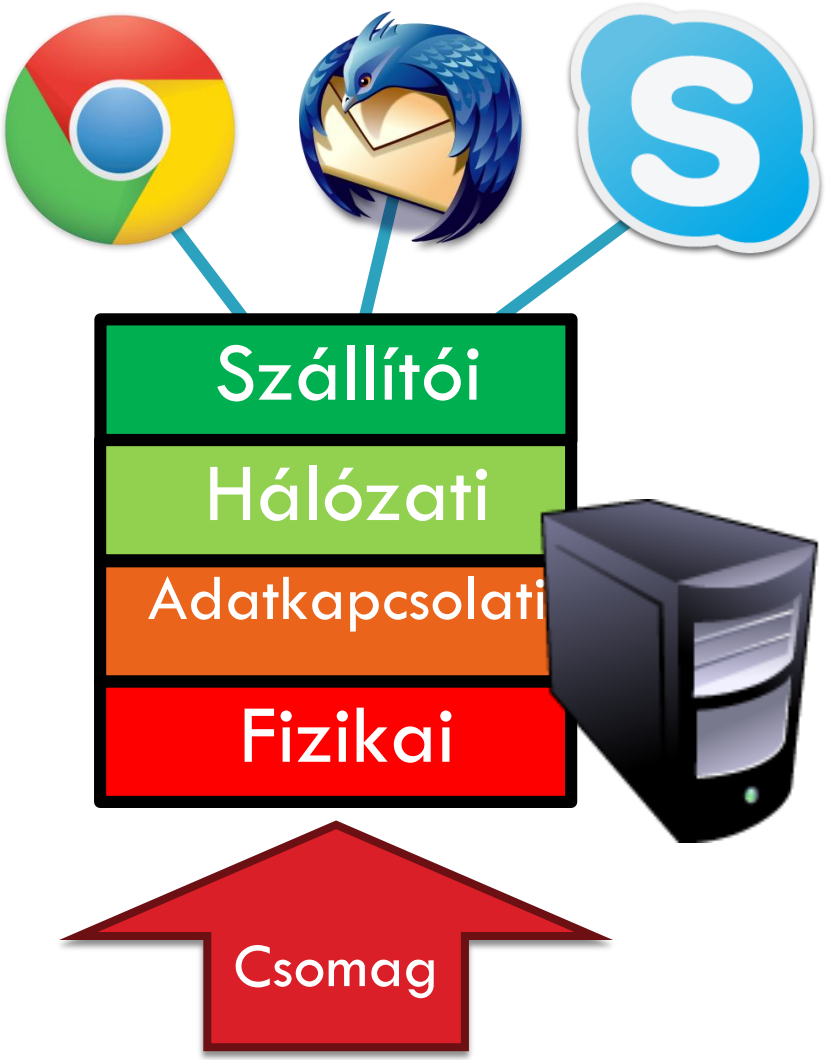
- Feladat:
  - Adatfolyamok demultiplexálása
- További lehetséges feladatok:
  - Hosszú élettartamú kapcsolatok
  - Megbízható, sorrendhelyes csomag leszállítás
  - Hiba detektálás
  - Folyam és torlódás vezérlés
- Kihívások:
  - Torlódások detektálása és kezelése
  - Fairség és csatorna kihasználás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

Kivonat

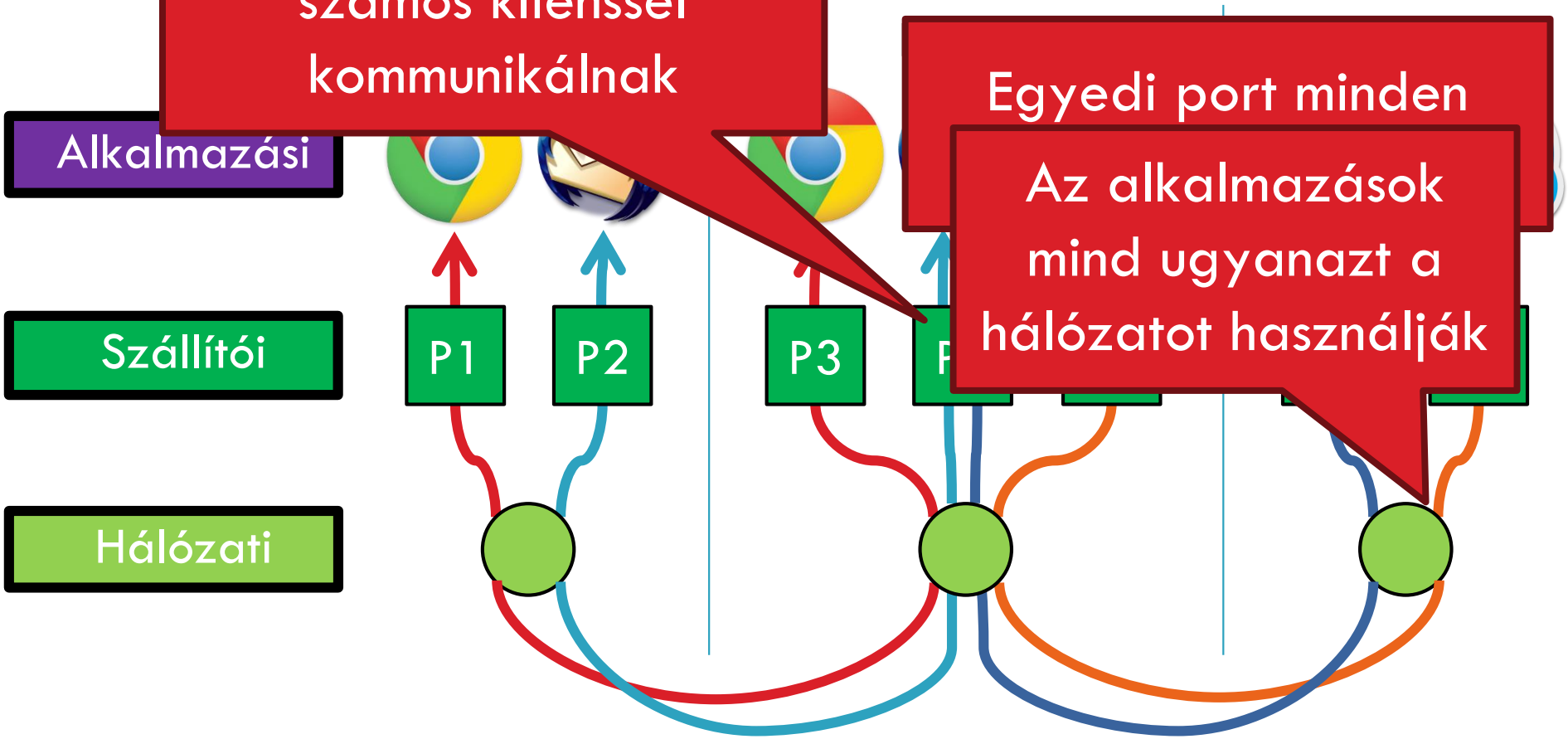
- UDP
- TCP
- Torlódás vezérlés
- TCP evolúciója
- A TCP problémái

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*>

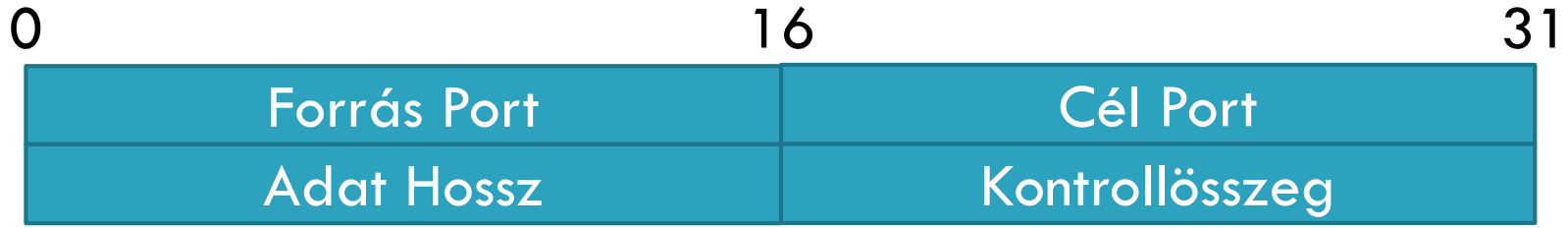
ahol src_ip, dst_ip a forrás és cél IP cím,

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)



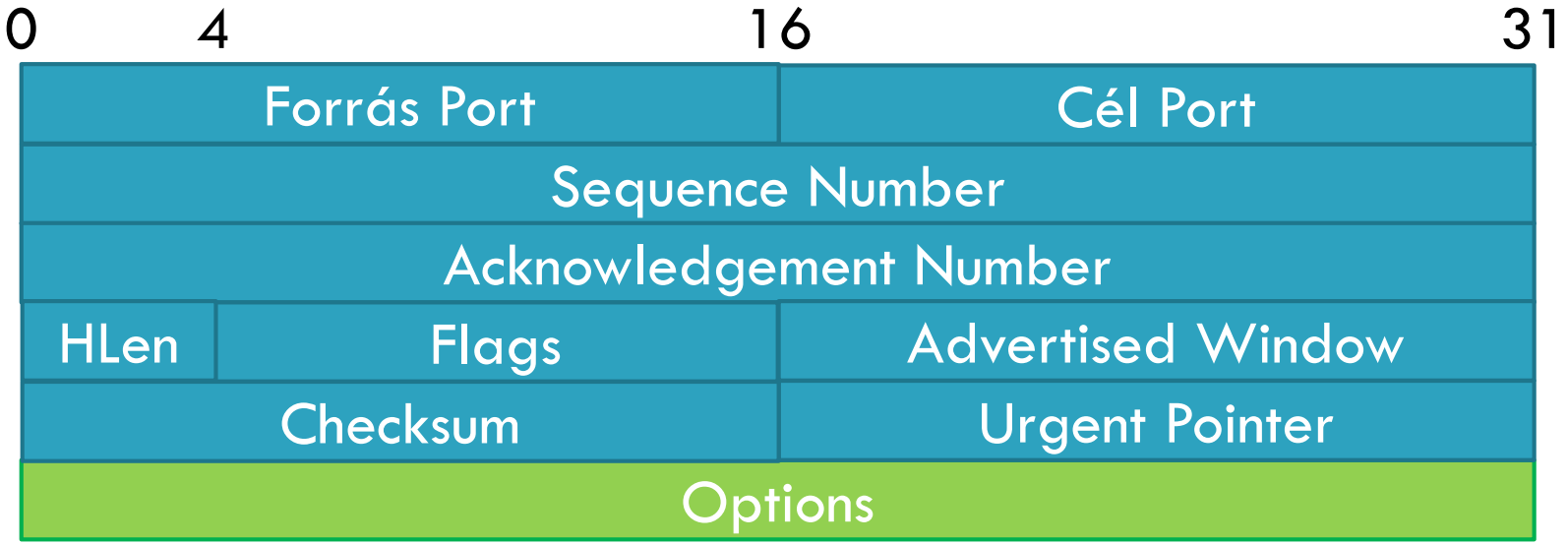
- 8 bájtos UDP fejléc
- Egyszerű, kapcsolatnélküli átvitel
  - C socketek: SOCK_DGRAM
- Port számok teszik lehetővé a demultiplexálást
  - 16 bit = 65535 lehetséges port
  - 0 port nem engedélyezett
- Kontrollösszeg hiba detektáláshoz
  - Hibás csomagok felismerése
  - Nem detektálja az elveszett, duplikátum 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

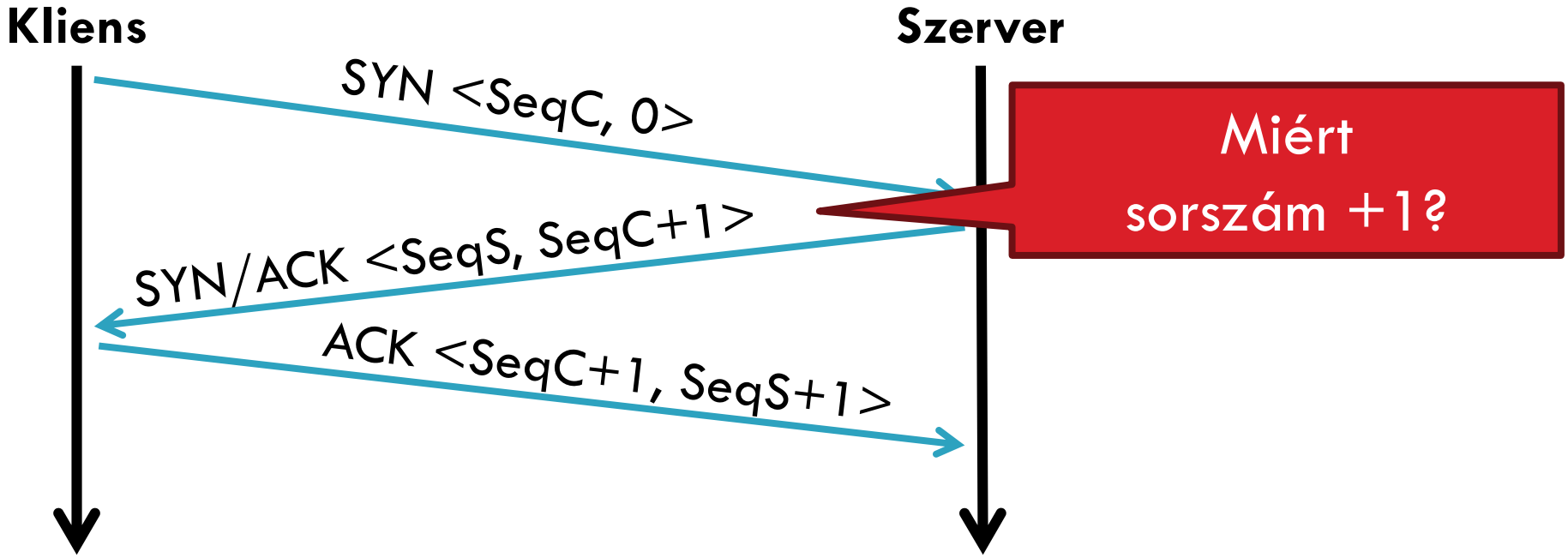
- Megbízható, sorrend helyes, két irányú bájt folyamok
  - Port számok a demultiplexáláshoz
  - Kapcsolat alapú
  - Folyam vezérlés
  - Torlódás vezérlés, fair viselkedés
- 20 bájtos fejléc + options fejlécek



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



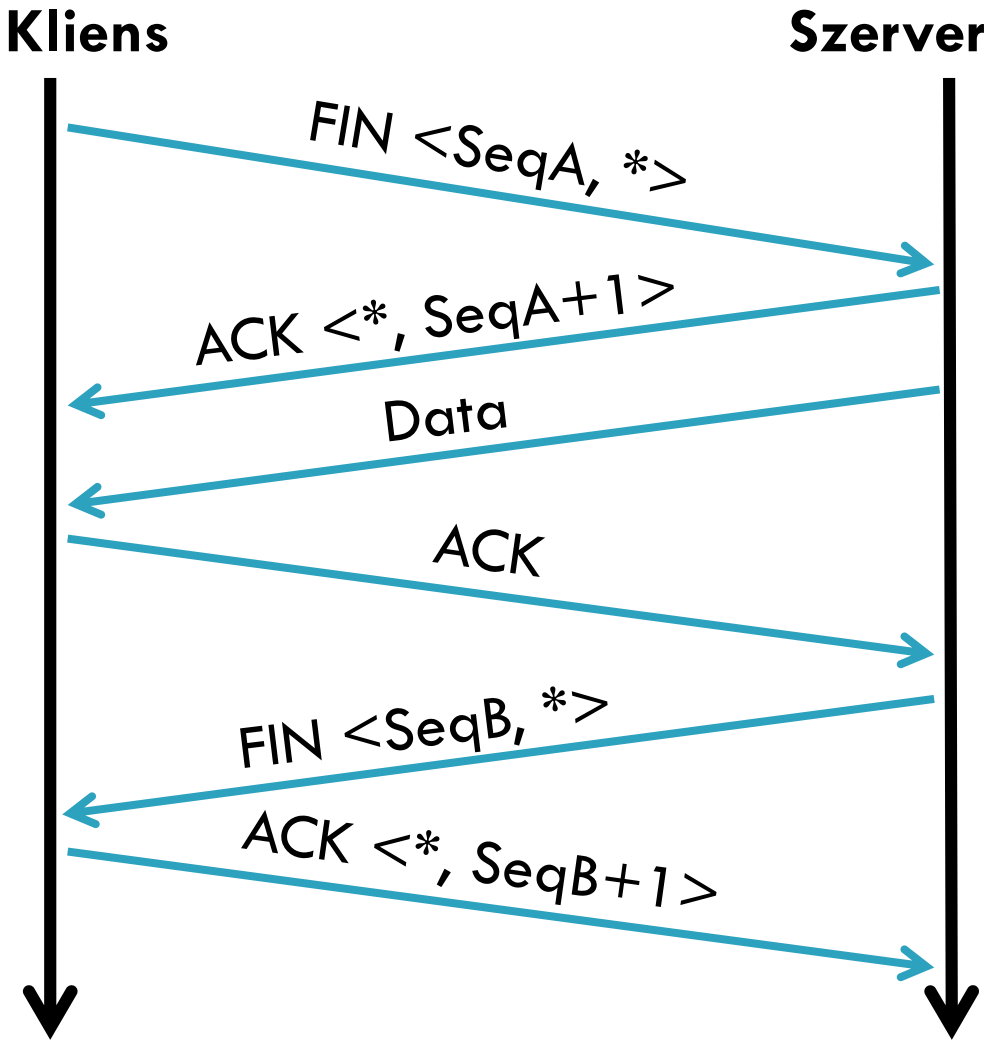
- Mindkét oldalon:
  - Másik fél értesítése a kezdő sorszámról
  - A másik fél kezdő sorszámának nyugtázása

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

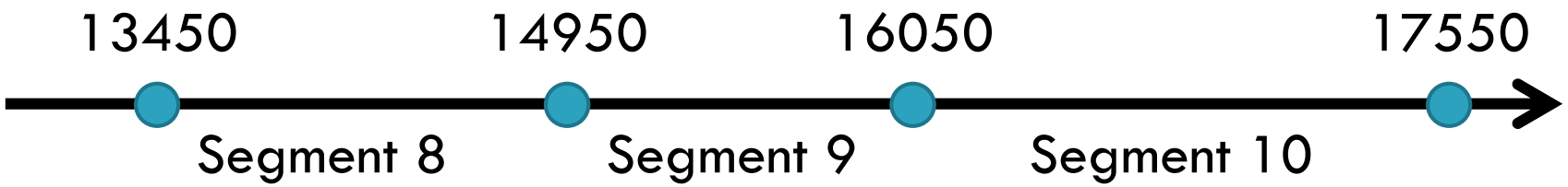
Kapcsolat lezárása

- 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?

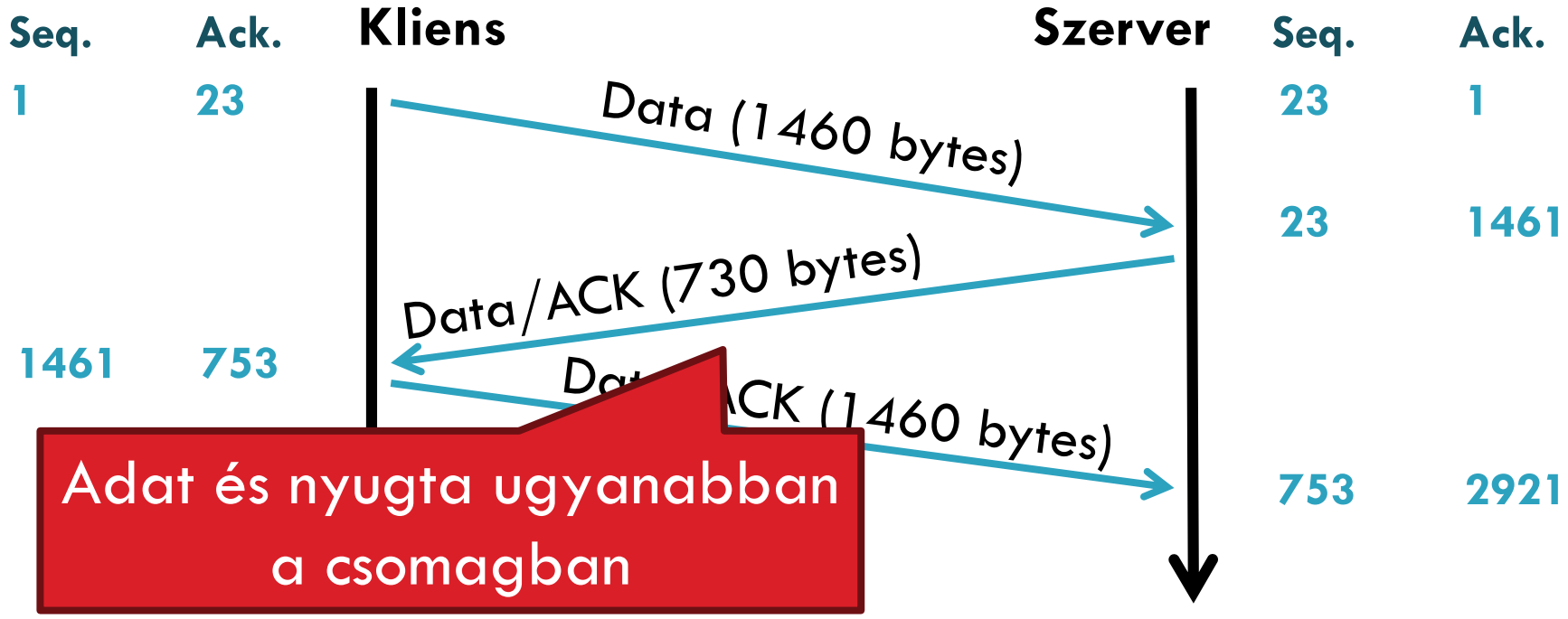


Sorszámok tere

- 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

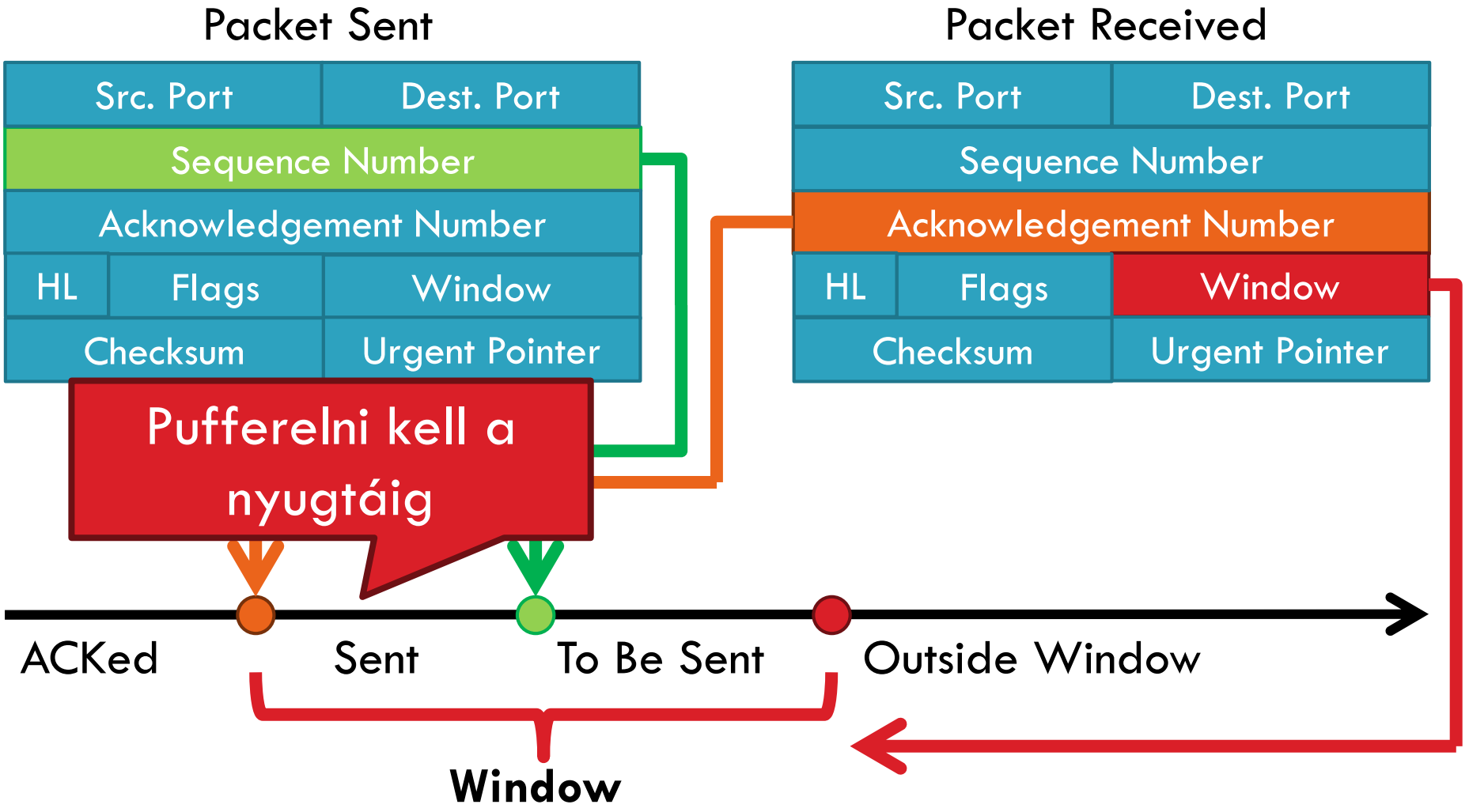


- Mindkét fél küldhet és fogadhat adatot
  - Különböző sorszámok a két irányba

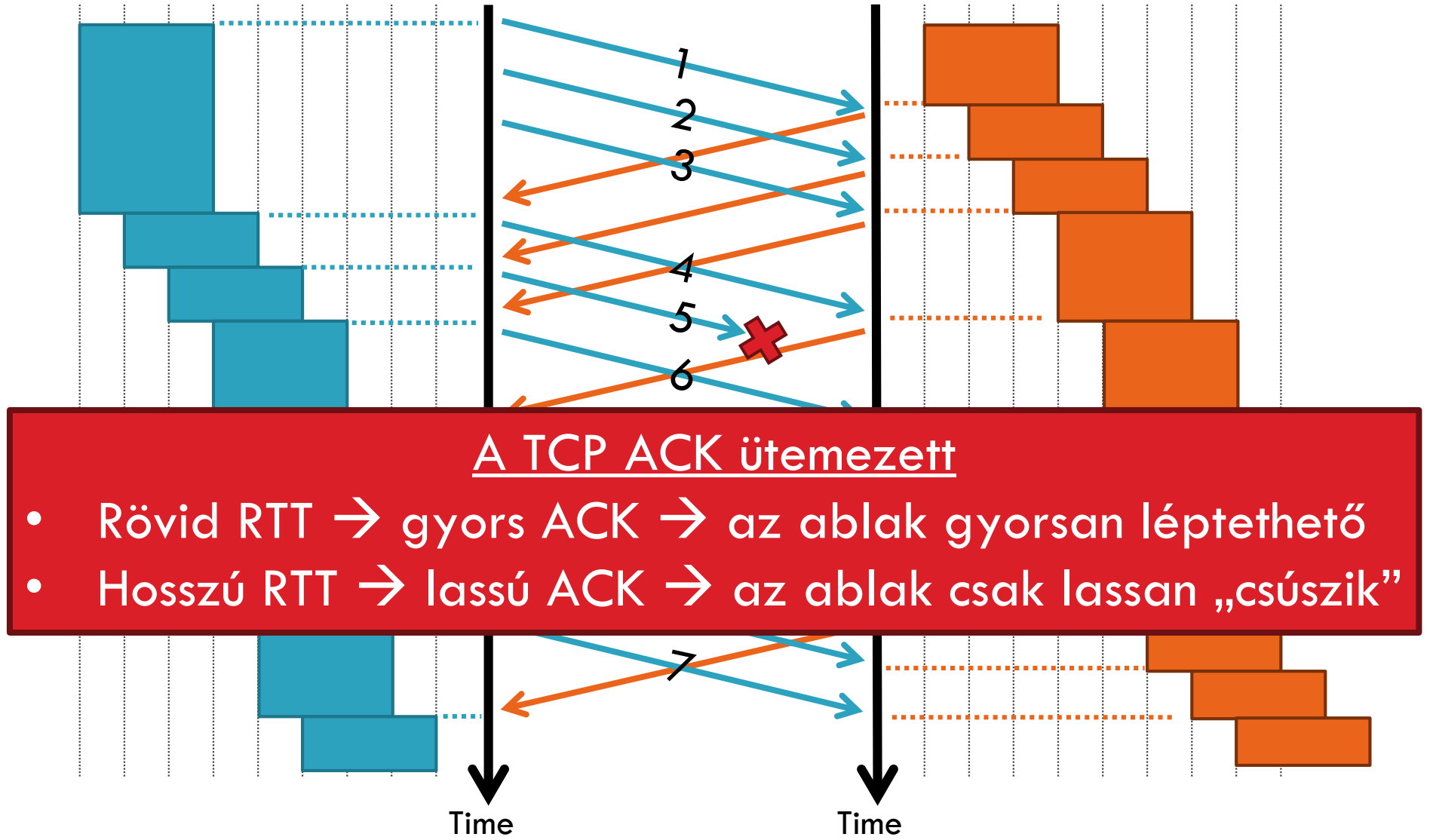
Folyam vezérlés

- 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?

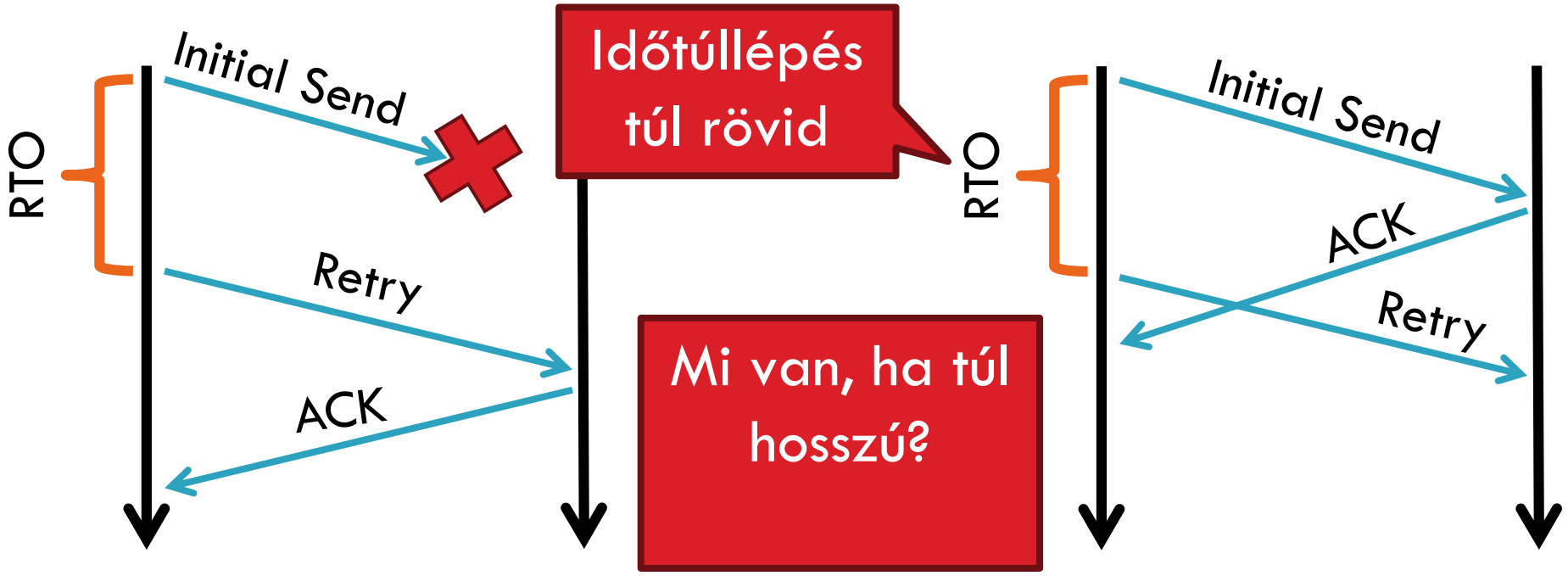
```
1. int flag = 1;
2. setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
   (char *) &flag, sizeof(int));
```

Hiba detektálás

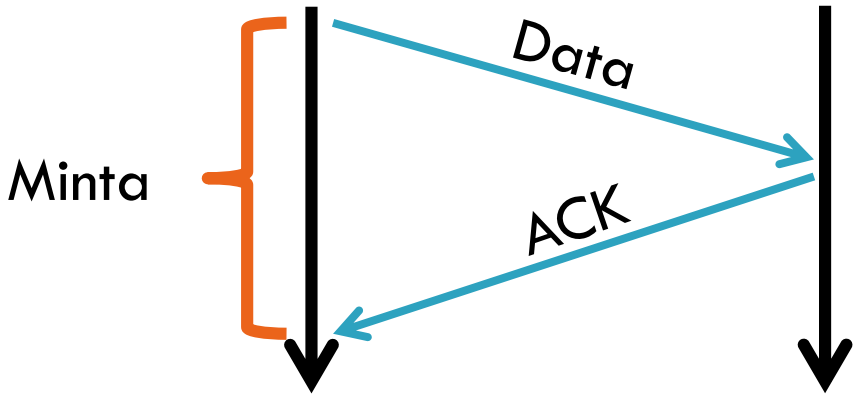
- A kontrollösszeg detektálja a hibás csomagokat
  - Az IP, TCP fejlécből és az adatból számoljuk
- A sorszámok segítenek a sorrendhelyes átvitelben
  - Duplikátumok eldobása
  - Helytelen sorrendben érkező csomagok sorba rendezése vagy eldobása
  - Hiányzó sorszámok elveszett csomagot jeleznek
- A küldő 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

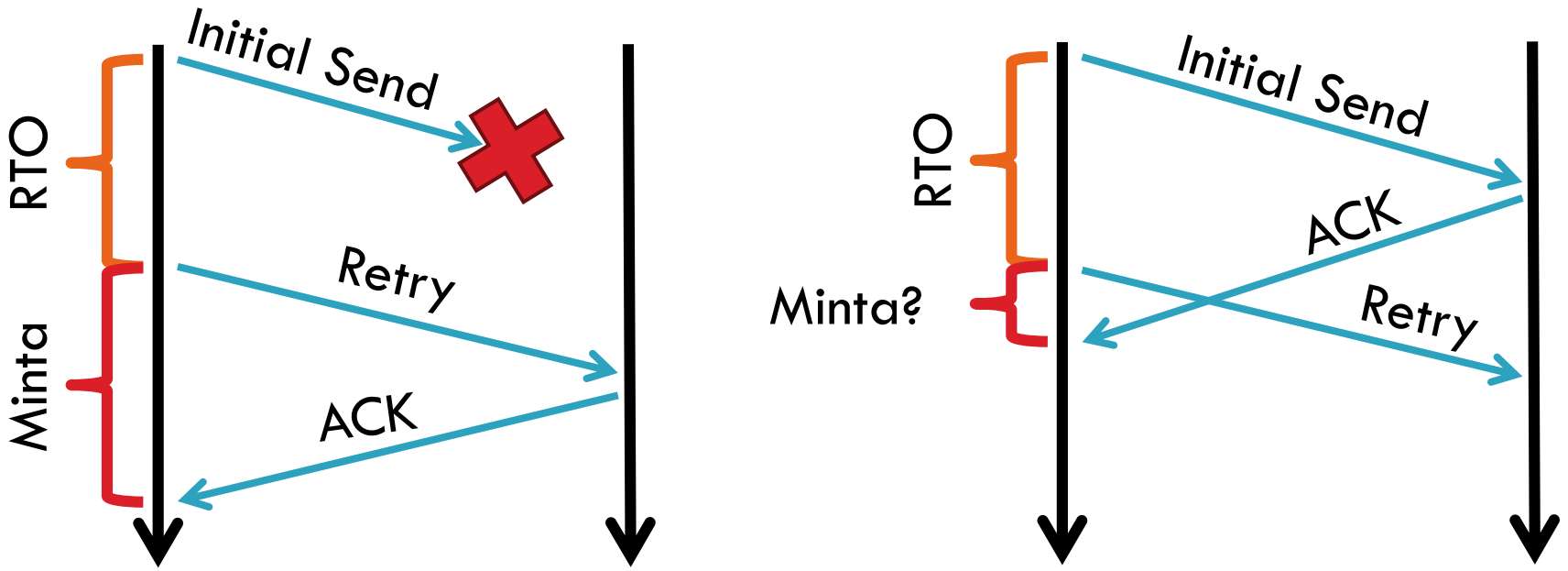


Round Trip Time becslés



- 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

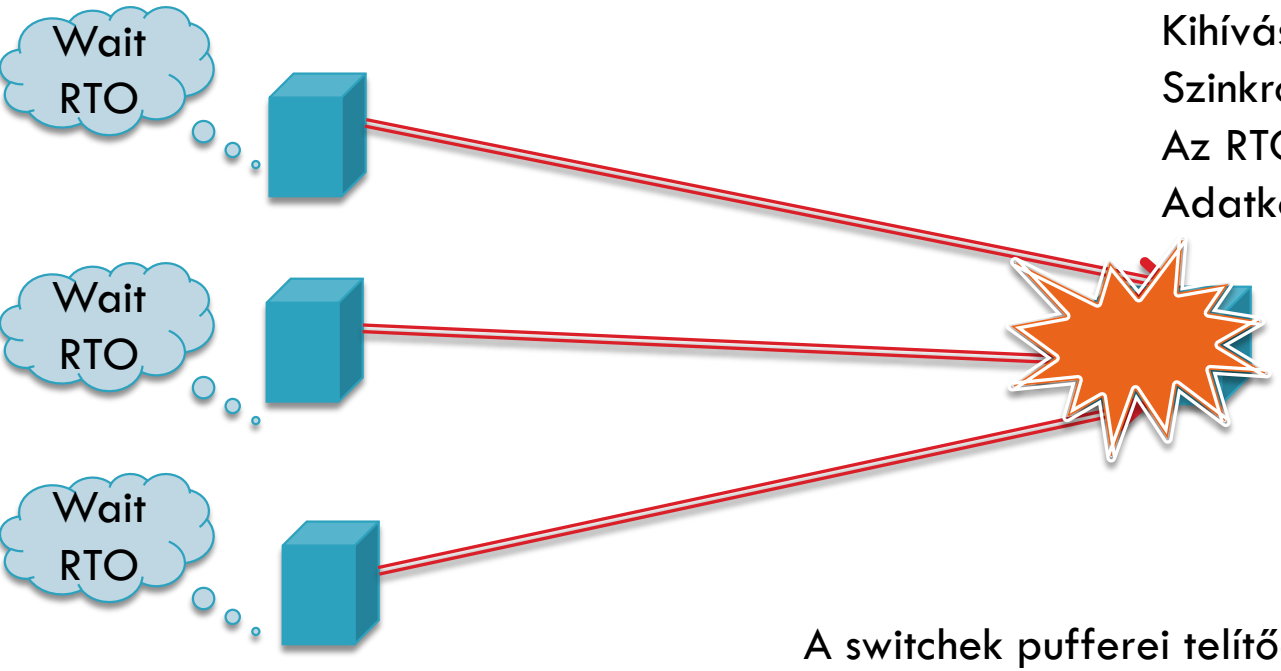


- **Karn algoritmusa**: dobjuk el azokat a mintákat, melyek egy csomag újraküldéséből származnak

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

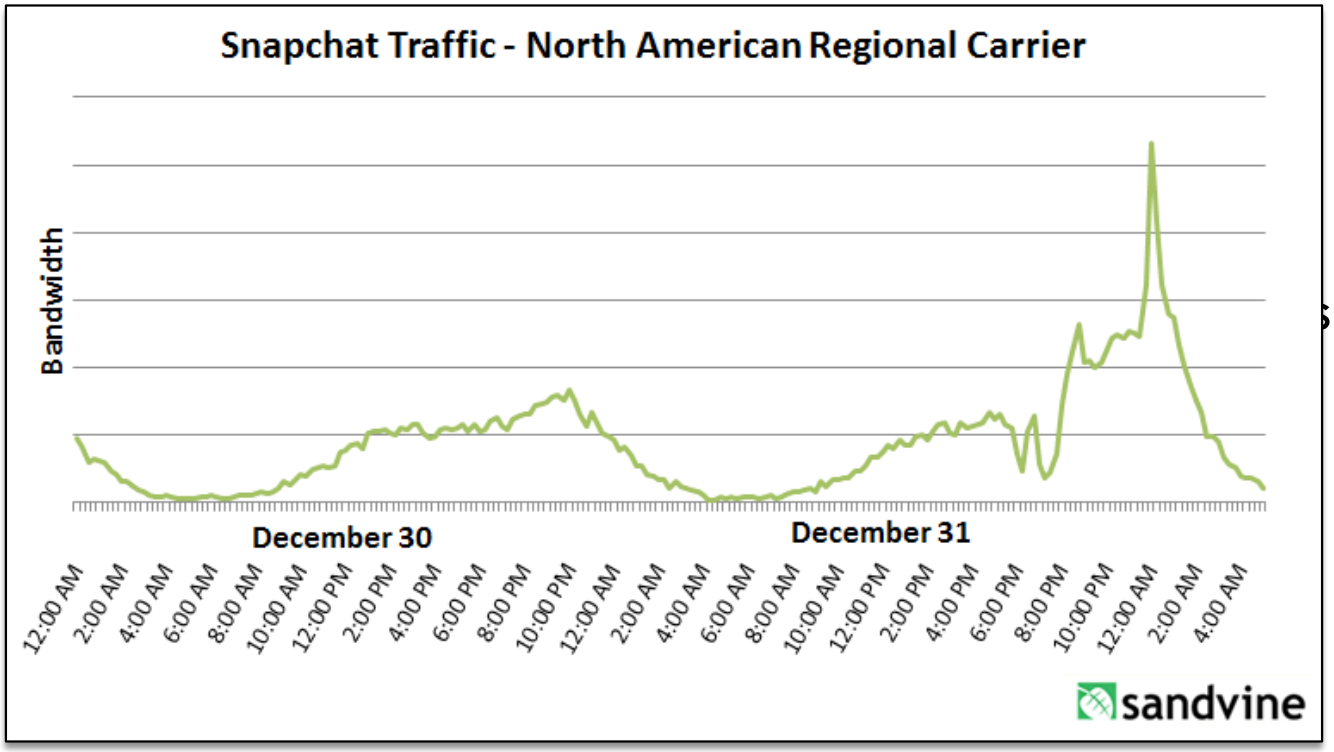
A switchek pufferei telítődnek és csomagok vesznek el!
Nyugta nem megy vissza ☹

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

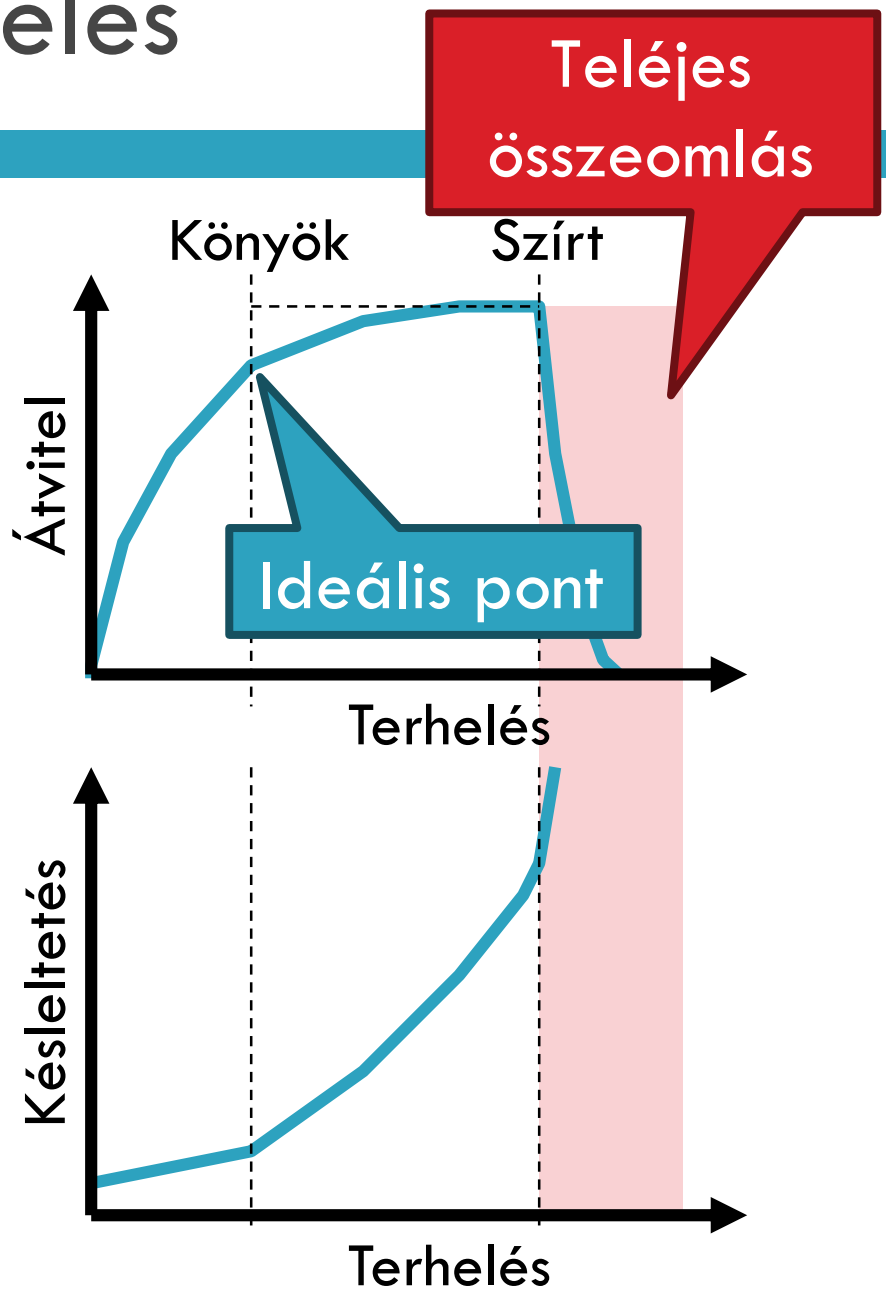


Miért rossz a torlódás?

- 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

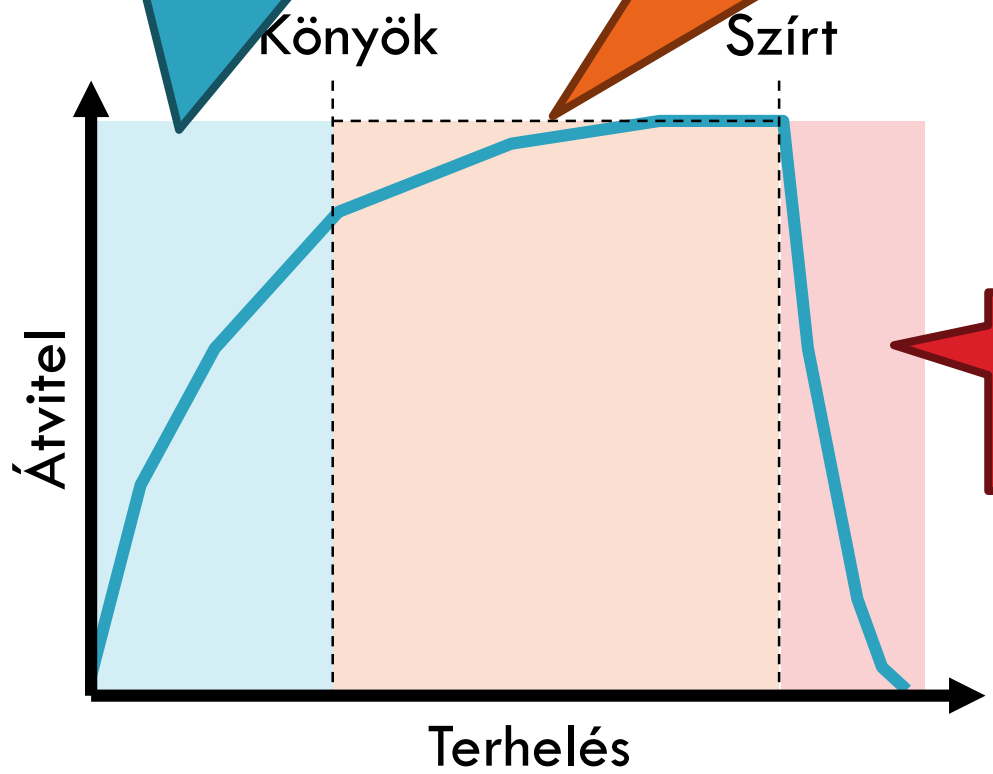
- Könyök („knee”)– a pont, ami után
  - Az átvitel szinte alig nő
  - Késleltetés viszont gyorsan emelkedik
- Egy egyszerű sorban (M/M/1)
  - Késleltetés = 1/(1 – utilization)
- Szírt („cliff”) – a pont, ami után
  - Átvitel lényegében leesik 0-ra
  - A késleltetés pedig → ∞



Torlódás vezérlés vs torlódás elkerülés

Torlódás elkerülés:
Maradj a könyök bal oldalán

Torlódás vezérlés
Maradj a szírt bal oldalán



Teljes
összeomlás

Advertised Window
Meghirdetett ablak, újragondolva

- Megoldja-e a torlódás problémáját a TCP esetén a meghirdetett ablak használata?

NEM

- Ez az ablak csak a fogadót védi a túlterheléstől
- Egy kellően gyors fogadó kimaxolhatja ezt az ablakot
  - Mi van, ha a hálózat lassabb, mint a fogadó?
  - Mi van, ha vannak konkurens folyamok is?
- Következmények
  - Az ablak méret határozza meg a küldési rátát
  - Az ablaknak állíthatónak kell lennie, hogy elkerüljük a torlódás miatti teljes összeomlást…

Általános megoldások

- Ne csináljunk semmit, küldjük a csomagokat megkülönböztetés nélkül
  - Nagy csomagvesztés, jósolhatatlan teljesítmény
  - Teljes összeomláshoz vezethet
- Erőforrás foglalás
  - Folyamokhoz előre sávszélességet allokálunk
  - Csomagküldés előtt egy tárgyalási szakaszra is szükség van
  - Hálózati támogatás kell hozzá
- Dinamikus beállítás
  - Próbák használata a torlódási szint megbecsléséhez
  - Gyorsítás, ha torlódási szint alacsony
  - Lassítás, amint nő a torlódás
  - Nem rendezett dinamika, elosztott koordináció

TCP Torlódásvezérlés

- Minden TCP kapcsolat rendelkezik egy ablakkal
  - A nem-nyugtázott csomagok számát vezérli
- Küldési ráta ~ window/RTT
- Ötlet: ablak méretének változtatása a küldési ráta vezérléséhez
- Vezessünk be egy torlódási ablakot (congestion window) a küldő oldalon
  - Torlódás vezérlés egy küldő oldali probléma
  - Jelölése: cwnd

Két fő komponens

1. Torlódás detektálás
   - Eldobott csomag egy megbízható jel
     - Késleltetés alapú megoldások – nehéz és kockázatos
   - Hogyan detektáljuk a csomag eldobását? Nyugtával
     - Időkorlát lejár ACK fogadása nélkül
     - Számos duplikált ACK jön be sorban (később lesz róla szó)
2. Ráta beállító algoritmus
   - *cwnd módosítása*
   - Sávszélesség próba
   - Válasz lépés a torlódásra

Ráta vezérlés

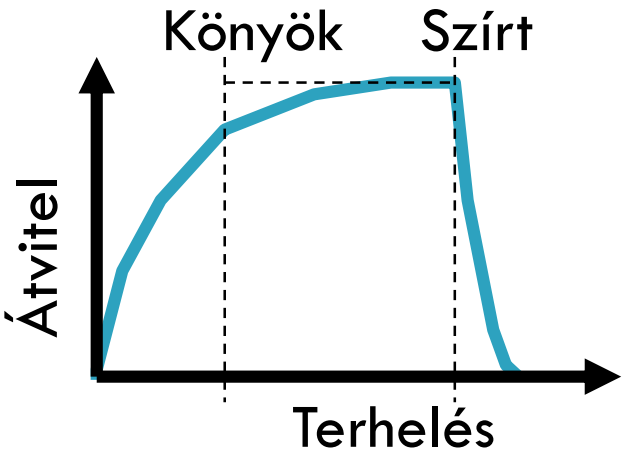
- Tudjuk, hogy a TCP ACK ütemezett
  - Torlódás = késleltetés = hosszú várakozás a nyugták között
  - Nincs torlódás = alacsony késleltetés = gyors ACK
- Alapvető algoritmus
  - ACK fogadása esetén: növeljük a cwnd ablakot
    - Adat leszállítva, valószínűleg gyorsabban is küldhetünk
    - *cwnd* növekedése arányos az RTT-vel
  - Csomagvesztés esetén: csökkentsük a cwnd ablakot
    - Adat elveszett, torlódásnak kell lennie a hálózatban
- Kérdés: milyen függvényt használjuk a növeléshez és csökkentéshez? !!!!

Torlódás vezérlés megvalósítása

- Három változót kell nyilvántartani:
  - *cwnd*: torlódási ablak
  - *adv_wnd*: a fogadó meghirdetett ablaka
  - *ssthresh*: vágási érték (a *cwnd* frissítésére használjuk)
- Küldésnél használjuk: *wnd* = *min*(*cwnd, adv_wnd*)
- A torlódás vezérlés két fázisa:
  1. Lassú indulás („Slow start”) (*cwnd* < *ssthresh*)
     - Az ún. bottleneck (legszűkebb) sávszélesség meghatározása a cél.
  2. Torlódás elkerülés (*cwnd* >= *ssthresh*)
     - AIMD – Additive Increase Multiplicative Decrease

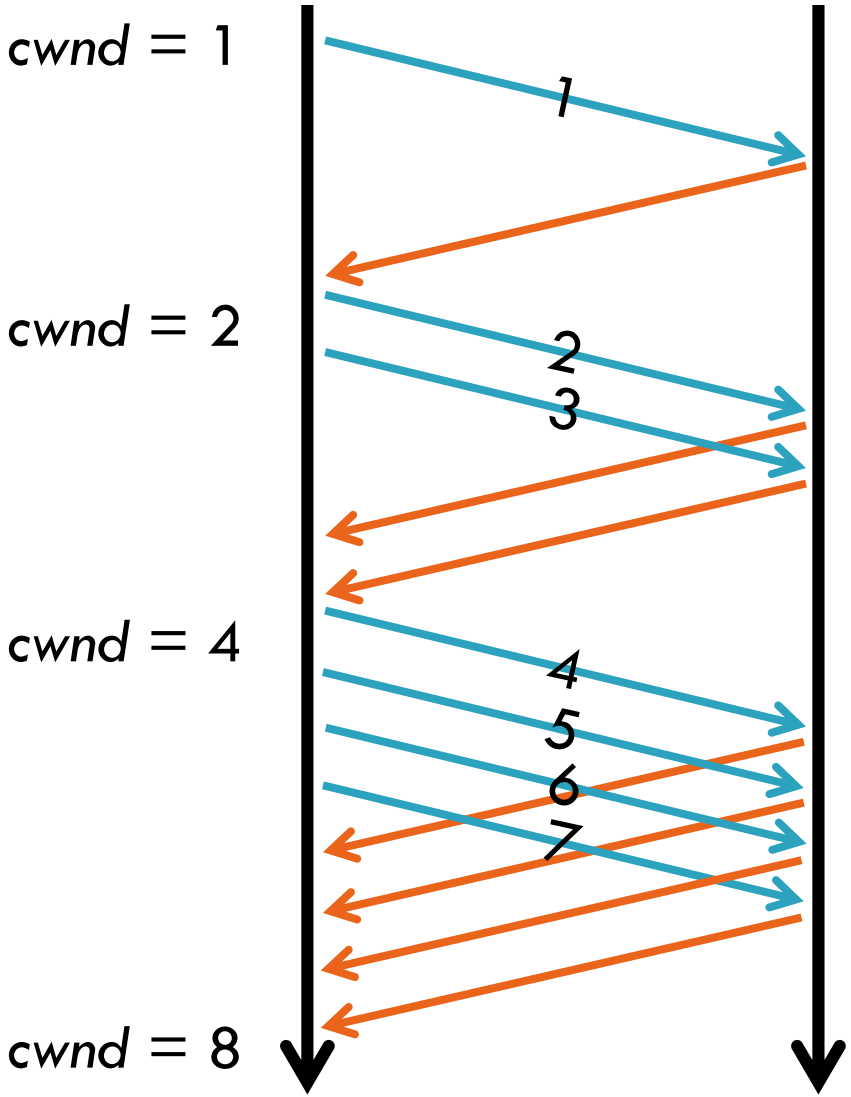
Lassú indulás - Slow Start

- Cél, hogy gyorsan elérjük a könyök pontot
- Egy kapcsolat kezdetén (vagy újraindításakor)
  - *cwnd* =1
  - *ssthresh* = *adv_wnd*
  - Minden nyugtázott szegmensre: *cwnd*++
- Egészen addig amíg
  - El nem érjük az *ssthresh* értéket
  - Vagy csomagvesztés nem történik
- A Slow Start valójában nem lassú
  - *cwnd* exponenciálisan nő



Slow Start példa

- *cwnd* gyorsan nő
- Lelassul, amikor…
  - *cwnd* >= *ssthresh*
  - Vagy csomagvesztés történik

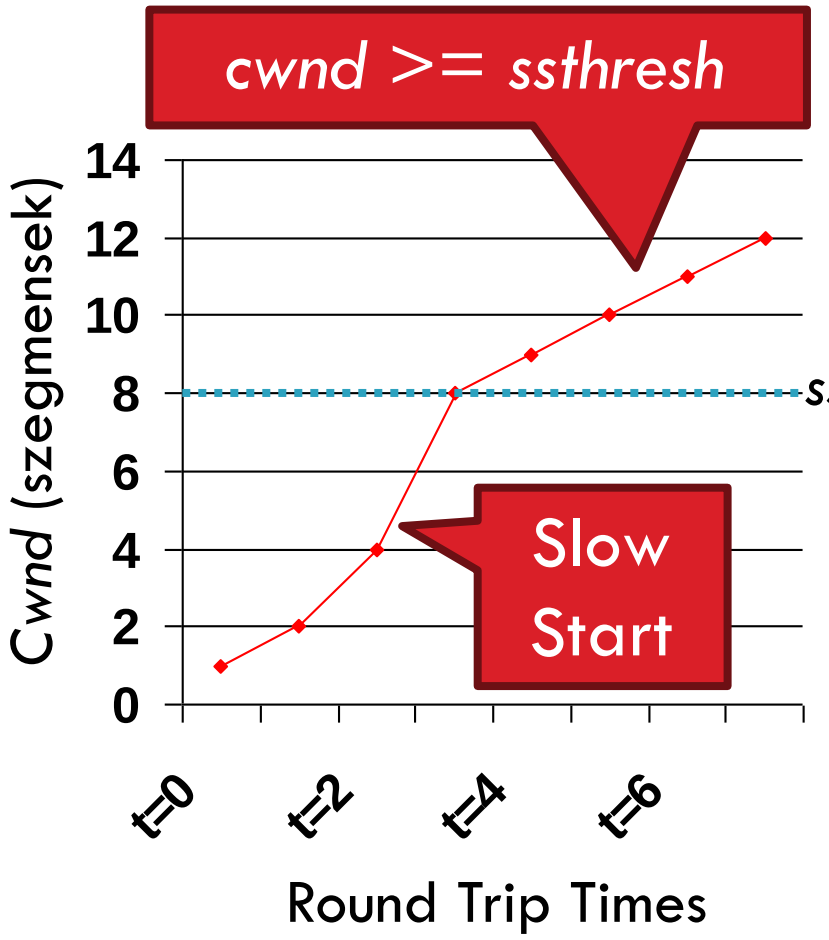


Torlódás elkerülés

- Additive Increase Multiplicative Decrease (AIMD) mód
- *ssthresh* valójában egy alsóbecslés a könyök pontra
- **Ha** *cwnd* >= *ssthresh* **akkor**

  Minden nyugtázott szegmens alkalmával növeljük a *cwnd értékét (1/cwnd)-vel* *(azaz cwnd += 1/cwnd)*.
- Azaz a *cwnd* eggyel nő, ha minden csomag nyugtázva lett.

Torlódás elkerülés példa



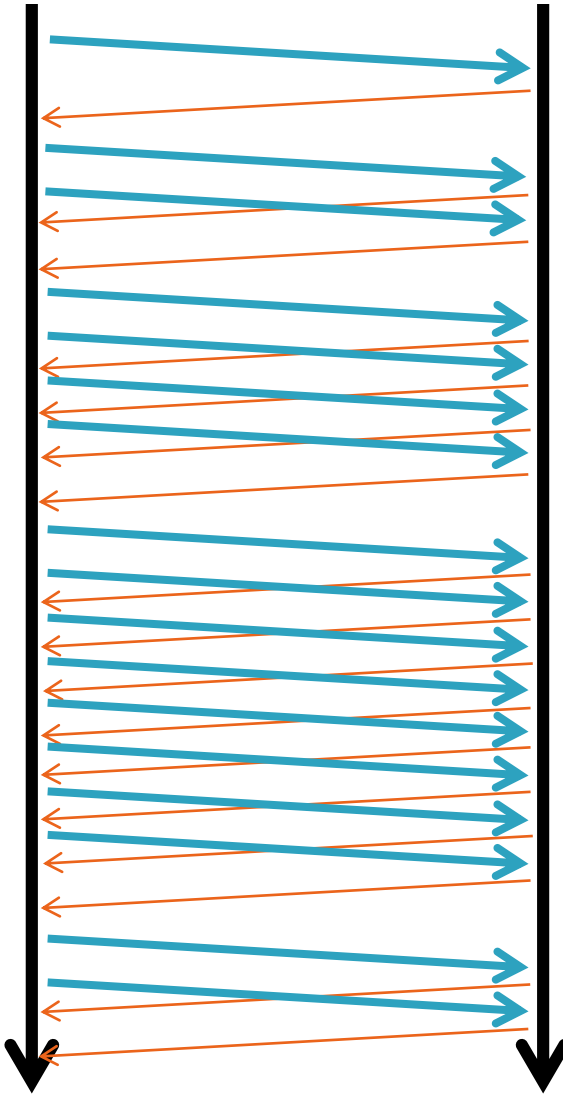
cwnd = 1

cwnd = 2

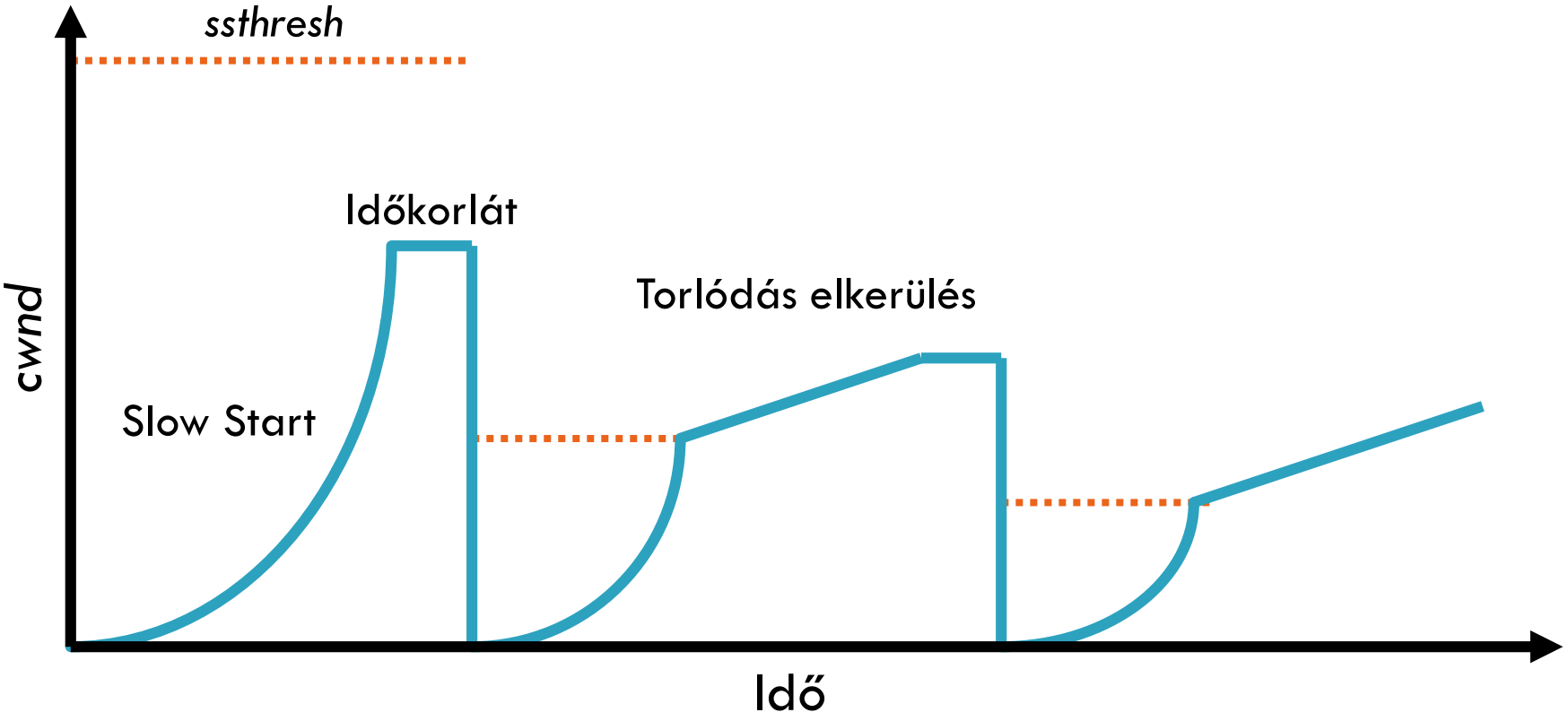
cwnd = 4

cwnd = 8

cwnd = 9



A teljes kép – TCP Tahoe (az eredeti TCP)



Összefoglalás - TCP jellemzői

„A *TCP* egy kapcsolatorientált megbízható szolgáltatás kétirányú bájt-folyamokhoz.”

KAPCSOLATORIENTÁLT

- Két résztvevő, ahol egy résztvevőt egy *IP-cím* és egy *port* azonosít.
- A kapcsolat egyértelműen azonosított a résztvevő párral.
- Nincs se *multi-*, se *broadcast* üzenetküldés.
- A kapcsolatot fel kell építeni és le kell bontani.
- Egy kapcsolat a lezárásáig aktív.

Összefoglalás - TCP jellemzői

„A *TCP* egy kapcsolatorientált megbízható szolgáltatás kétirányú bájt-folyamokhoz.”

MEGBÍZHATÓSÁG

- Minden csomag megérkezése nyugtázásra kerül.
- A nem nyugtázott adatcsomagokat újraküldik.
- A fejléchez és a csomaghoz ellenőrzőösszeg van rendelve.
- A csomagokat számozza, és a fogadónál sorba rendezésre kerülnek a csomagok a sorszámaik alapján.
- Duplikátumokat törli.

Összefoglalás - TCP jellemzői

„A *TCP* egy kapcsolatorientált megbízható szolgáltatás kétirányú bájt-folyamokhoz.”

KÉTIRÁNYÚ BÁJTFOLYAM

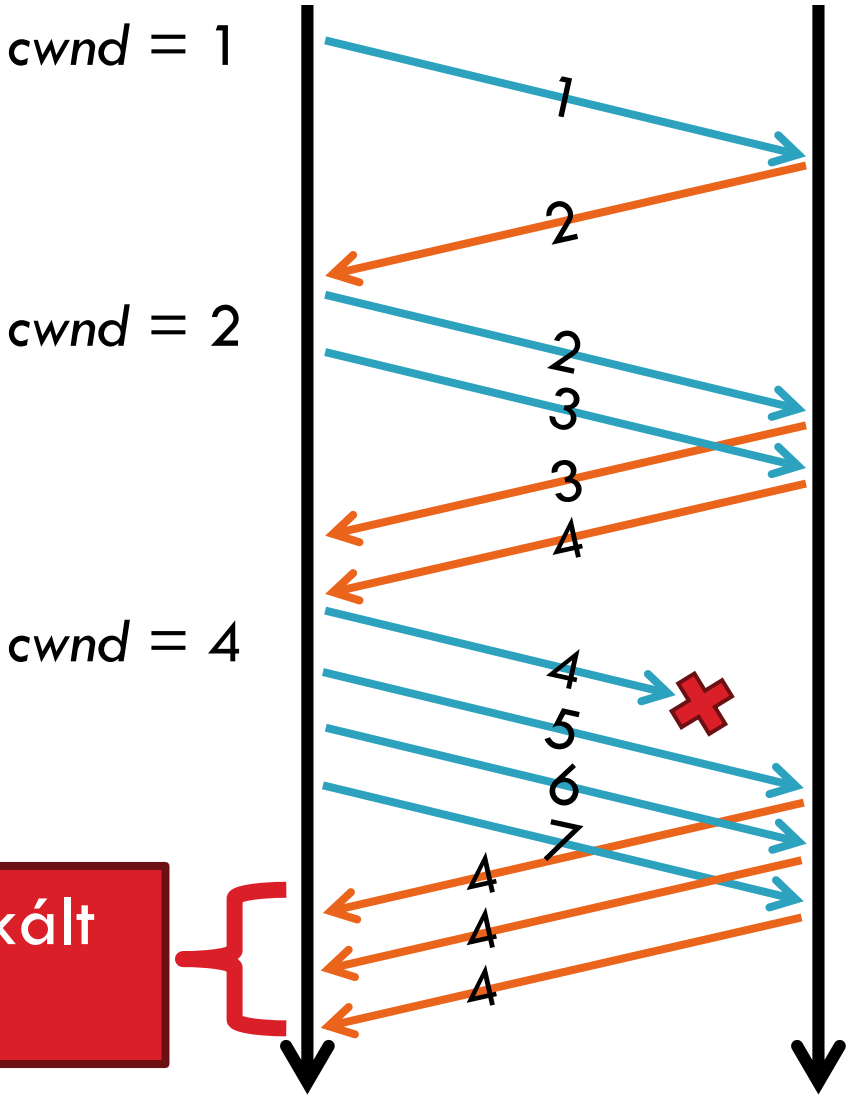
- Az adatok két egymással ellentétes irányú bájt-sorozatként kerülnek átvitelre.
- A tartalom nem interpretálódik.
- Az adatcsomagok időbeli viselkedése megváltozhat: átvitel sebessége növekedhet, csökkenhet, más késés, más sorrendben is megérkezhetnek.
- Megpróbálja az adatcsomagokat időben egymáshoz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

A TCP evolúciója

- Az eddigi megoldások a TCP Tahoe működéshez tartoztak
  - Eredeti TCP
- A TCP-t 1974-ben találták fel!
  - Napjainkba számos változata létezik
- Kezdeti népszerű változat: TCP Reno
  - Tahoe lehetőségei, plusz…
  - Gyors újraküldés (Fast retransmit)
    - 3 duplikált ACK? -> újraküldés (ne várjunk az RTO-ra)
  - Gyors helyreállítás (Fast recovery)
    - Csomagvesztés esetén:
      - set cwnd = cwnd/2 (ssthresh = az új cwnd érték)

TCP Reno: Gyors újraküldés

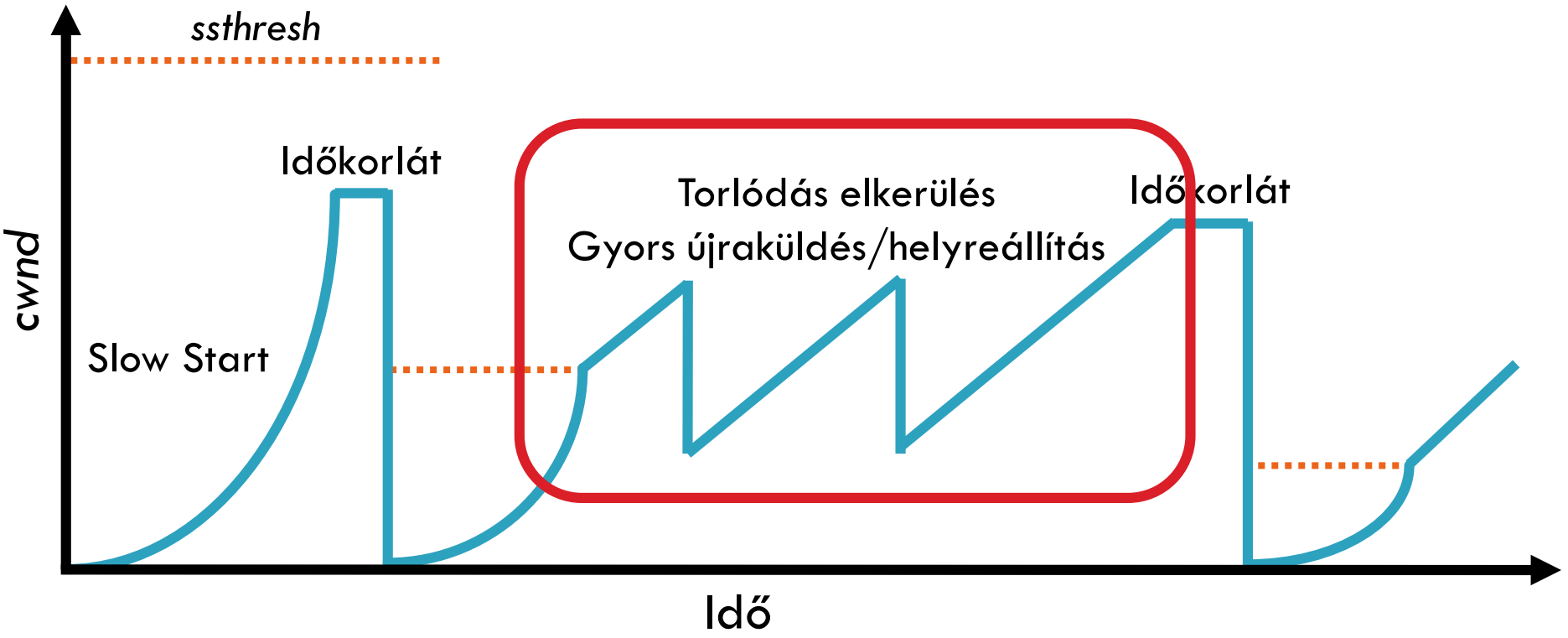
- Probléma: Tahoe esetén ha egy csomag elveszik, akkor hosszú a várakozás az RTO-ig
- Reno: újraküldés 3 duplikált nyugta fogadása esetén
- Duplikált: ugyanaz a sorszám
  - Explicit jele a csomagvesztésnek



TCP Reno: Gyors helyreállítás

- Gyors újraküldés után módosítjuk a torlódási ablakot:
  - *cwnd* := *cwnd*/2 *(valójában ez a Multiplicative Decrease)*
  - ssthresh := az új cwnd
  - Azaz nem álltjuk vissza az eredeti 1-re a cwnd-t!!!
  - Ezzel elkerüljük a felesleges slow start fázisokat!
  - Elkerüljük a költséges időkorlátokat
- Azonban ha az RTO lejár, továbbra is *cwnd* = 1
  - Visszatér a slow start fázishoz, hasonlóan a Tahoe-hoz
  - Olyan csomagokat jelez, melyeket egyáltalán nem szállítottunk le
  - A torlódás nagyon súlyos esetére figyelmeztet!!!

Példa: Gyors újraküldés/helyreállítás



- Stabil állapotban, a cwnd az optimális ablakméret körül oszcillál
- TCP mindig csomagdobásokat kényszerít ki…

Számos TCP változat…

- Tahoe: az eredeti
  - Slow start és AIMD
  - Dinamikus RTO, RTT becsléssel
- Reno:
  - fast retransmit (3 dupACKs)
  - fast recovery (cwnd = cwnd/2 vesztés esetén)
- NewReno: javított gyors újraküldés
  - Minden egyes duplikált ACK újraküldést vált ki
  - Probléma: >3 hibás sorrendben fogadott csomag is újraküldést okoz (hibásan!!!)…
- Vegas: késleltetés alapú torlódás elkerülés
- …

TCP a valóságban

- Mi a legnépszerűbb variáns napjainkban?
  - Probléma: TCP rosszul teljesít nagy késleltetés-sávszélesség szorzattal rendelkező hálózatokban (a modern Internet ilyen)
  - Compound TCP (Windows)
    - Reno alapú
    - Két torlódási ablak: késleltetés alapú és vesztés alapú
    - Azaz egy összetett torlódás vezérlést alkalmaz
  - TCP CUBIC (Linux)
    - Fejlettebb BIC (Binary Increase Congestion Control) változat
    - Az ablakméretet egy harmadfokú egyenlet határozza meg
    - A legutolsó csomagvesztéstől eltelt T idővel paraméterezett

Nagy késleltetés-sávszélesség szorzat (Delay-bandwidth product)

- Probléma: A TCP nem teljesít jól ha
  - A hálózat kapacitása (sávszélessége) nagy
  - A késleltetés (RTT) nagy
  - Vagy ezek szorzata nagy
    - b * d = maximális szállítás alatt levő adatmennyiség
    - Ezt nevezzük késleltetés-sávszélesség szorzatnak
- Miért teljesít ekkor gyengén a TCP?
  - A slow start és az additive increase csak lassan konvergál
  - A TCP ACK ütemezett (azaz csak minden ACK esetén történik esemény)
    - A nyugták beérkezési gyorsasága határozza meg, hogy milyen gyorsan tud reagálni
    - Nagy RTT → késleltetett nyugták → a TCP csak lassan reagál a megváltozott viszonyokra

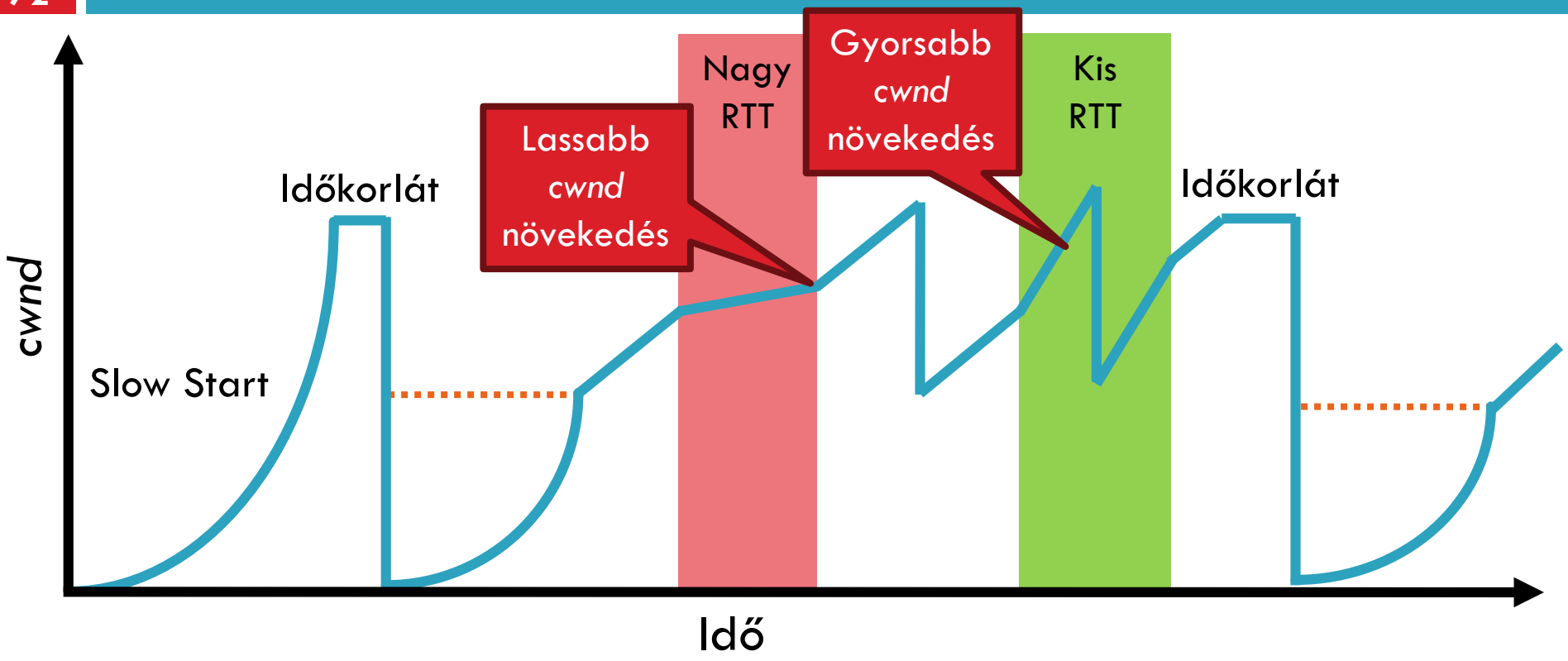
Célok

- A TCP ablak gyorsabb növelése
  - A slow start és az additive increase túl lassú, ha nagy a sávszélesség
  - Sokkal gyorsabb konvergencia kell
- Fairség biztosítása más TCP változatokkal szemben
  - Az ablak növelése nem lehet túl agresszív
- Javított RTT fairség
  - A TCP Tahoe/Reno folyamok nem adnak fair erőforrás-megosztást nagyon eltérő RTT-k esetén
- Egyszerű implementáció

Compound TCP

- Alap TCP implementáció Windows rendszereken
- Ötlet: osszuk a *torlódási ablakot* két különálló ablakba
  - Hagyományos, vesztés alapú ablak
  - Új, késleltetés alapú ablak
- *wnd* = min(*cwnd* + *dwnd*, *adv_wnd*)
  - *cwnd*-t az AIMD vezérli AIMD
  - *dwnd* a késleltetés alapú ablak
- A *dwnd* beállítása:
  - Ha nő az RTT, csökken a *dwnd* (*dwnd* >= 0)
  - Ha csökken az RTT, nő a *dwnd*
  - A növekesés/csökkenés arányos a változás mértékével

Compound TCP példa



- Agresszívan reagál az RTT változására
- Előnyök: Gyors felfutás, sokkal fairebb viselkedés más folyamokkal szemben eltérő RTT esetén
- Hátrányok: folyamatos RTT becslés

TCP CUBIC

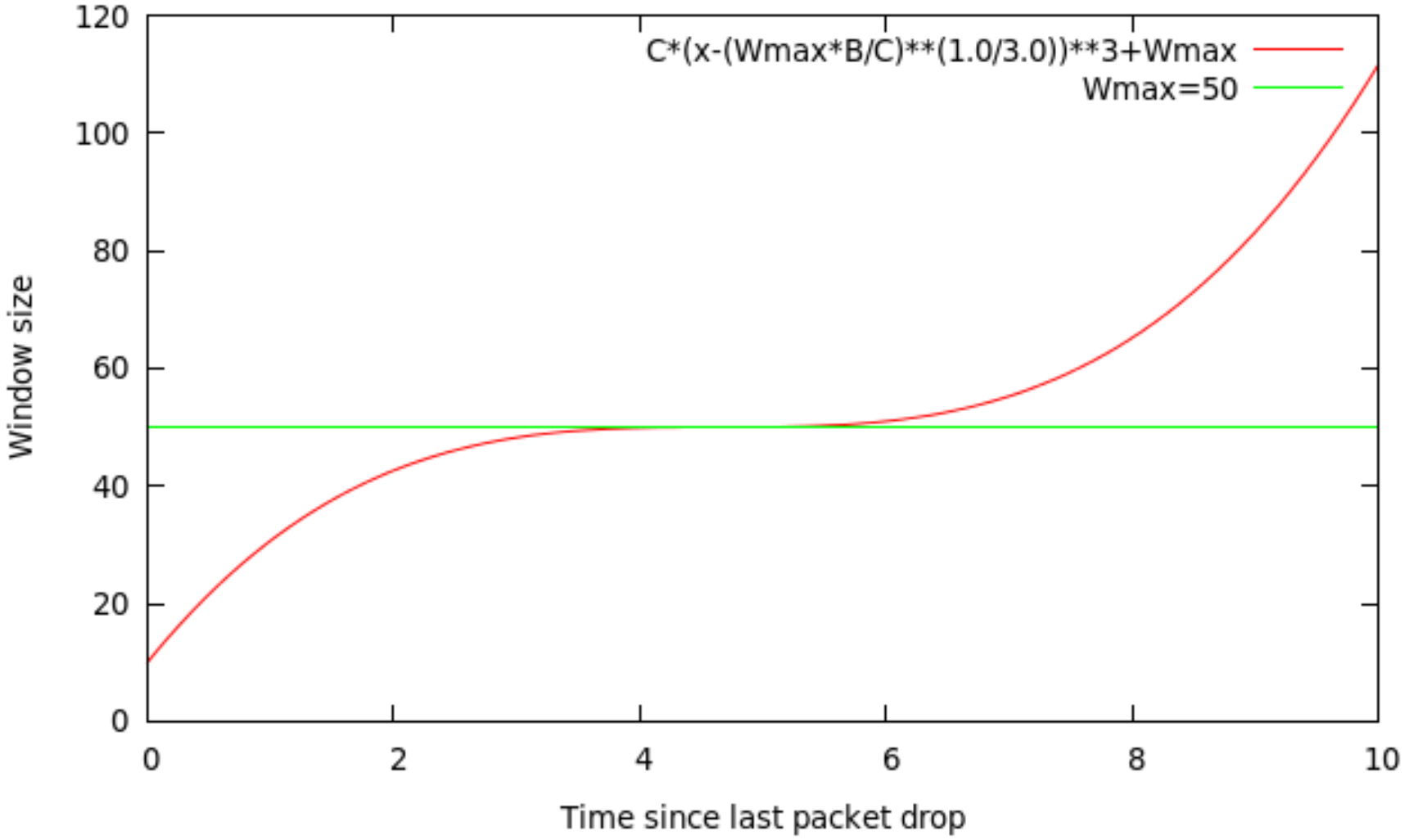
- Alap TCP implementáció Linux rendszereken
- Az AIMD helyettesítése egy „köbös” (CUBIC) függvénnyel

$$W_{cubic} = C(T - K)^3 + W_{max} \quad (1)$$

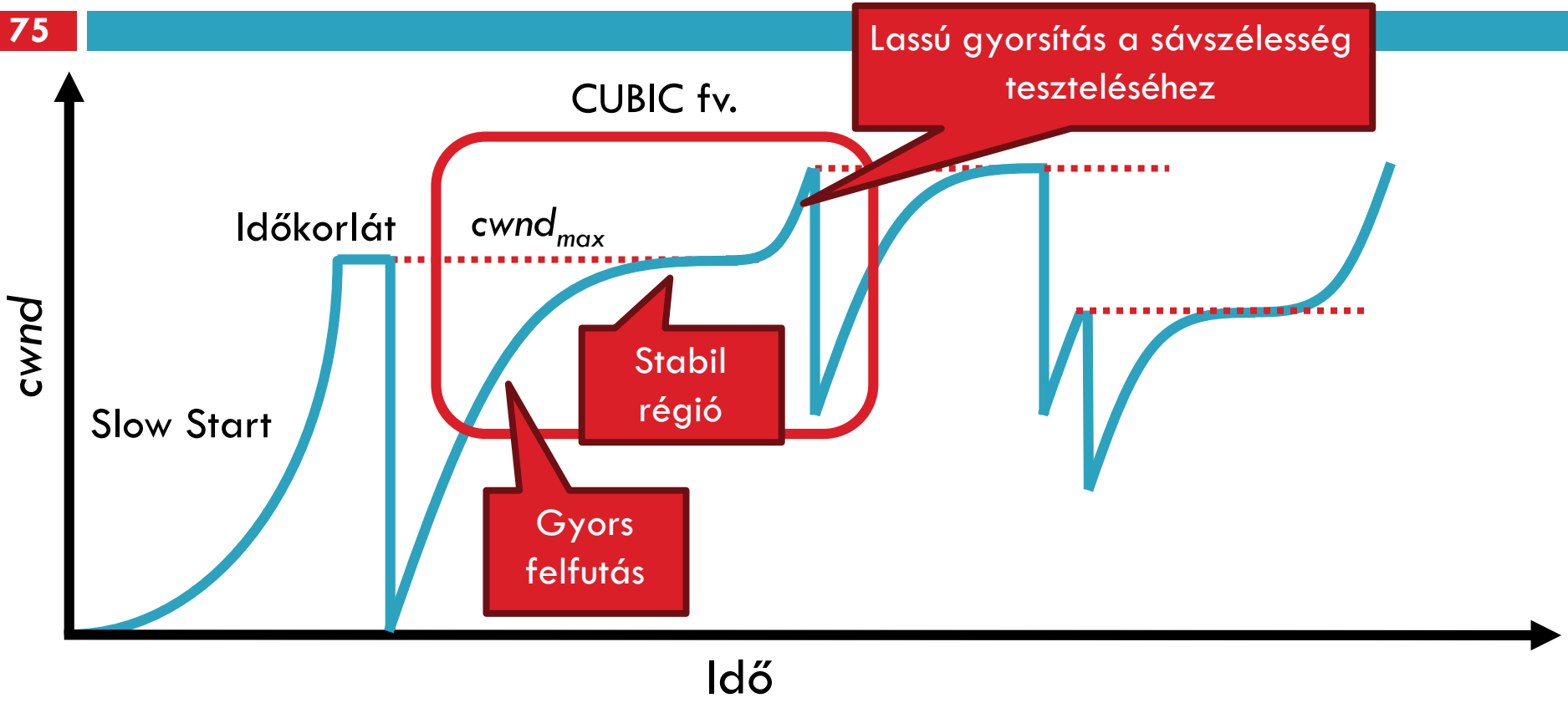
$$\text{C is a scaling constant, and K} = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

  - B → egy konstans a multiplicative increase fázishoz
  - T → eltelt idő a legutóbbi csomagvesztés óta
  - W_max → cwnd a legutolsó csomagvesztés idején

TCP CUBIC


C*(x-(Wmax*B/C)**(1.0/3.0))**3+Wmax
Wmax=50
Window size
Time since last packet drop
0
20
40
60
80
100
120
0
2
4
6
8
10

TCP CUBIC példa



- Kevésbé pazarolja a sávszélességet a gyors felfutások miatt
- A stabil régió és a lassú gyorsítás segít a fairség biztosításában
  - A gyors felfutás sokkal agresszívabb, mint az additive increase
  - A Tahoe/Reno variánsokkal szembeni fairséghez a CUBIC-nak nem szabad ennyire agresszívnak lennie

Problémák a TCP-vel

- Az Internetes forgalom jelentős része TCP
- Azonban számos probléma okozója is egyben
  - Gyenge teljesítmény kis folyamok esetén
  - Gyenge teljesítmény wireless hálózatokban
  - DoS támadási felület

Kis folyamok (flows)

- Probléma: kis folyamok esetén torz viselkedés
  - 1 RTT szükséges a kapcsolat felépítésére (SYN, SYN/ACK)
    - pazarló
  - *cwnd* mindig 1-gyel indul
    - Nincs lehetőség felgyorsulni a kevés adat miatt
- Az Internetes forgalom nagy része kis folyam
  - Többnyire HTTP átvitel, <100KB
  - A legtöbb TCP folyam el se hagyja a slow start fázist!!!
- Lehetséges megoldás (Google javaslat):
  - Kezdeti *cwnd* megnövelése 10-re
  - TCP Fast Open: kriptográfiai hashek használata a fogadó azonosítására, a három-utas kézfogás elhagyható helyette hash (cookie) küldése a syn csomagban

Wireless hálózatok

- Probléma: A Tahoe és Reno esetén csomagvesztés = torlódás
  - WAN esetén ez helyes, ritka bit hibák
  - Azonban hamis vezeték nélküli hálózatokban, gyakori interferenciák
- TCP átvitel ~ $1/\sqrt{vesztési\ ráta}$
  - Már néhány interferencia miatti csomagvesztés elég a teljesítmény drasztikus csökkenéséhez
- Lehetséges megoldások:
  - Réteg modell megsértése, adatkapcsolati információ a TCP-be
  - Késleltetés alapú torlódás vezérlés használata (pl. TCP Vegas)
  - Explicit torlódás jelzés - Explicit congestion notification (ECN)

Szolgáltatás megtagadása
Denial of Service (DoS)

- Probléma: a TCP kapcsolatok állapottal rendelkeznek
  - A SYN csomagok erőforrásokat foglalnak az szerveren
  - Az állapot legalább néhány percig fennmarad (RTO)
- SYN flood: elég sok SYN csomag küldése a szervernek ahhoz, hogy elfogyjon a memória és összeomoljon a kernel
- Megoldás: SYN cookie-k
  - Ötlet: ne tároljunk kezdeti állapotot a szerveren
  - Illesszük az állapotot a SYN/ACK csomagokba (a sorszám mezőbe (sequence number mező))
  - A kliensnek vissza kell tükrözni az állapotot…

Kitekintés

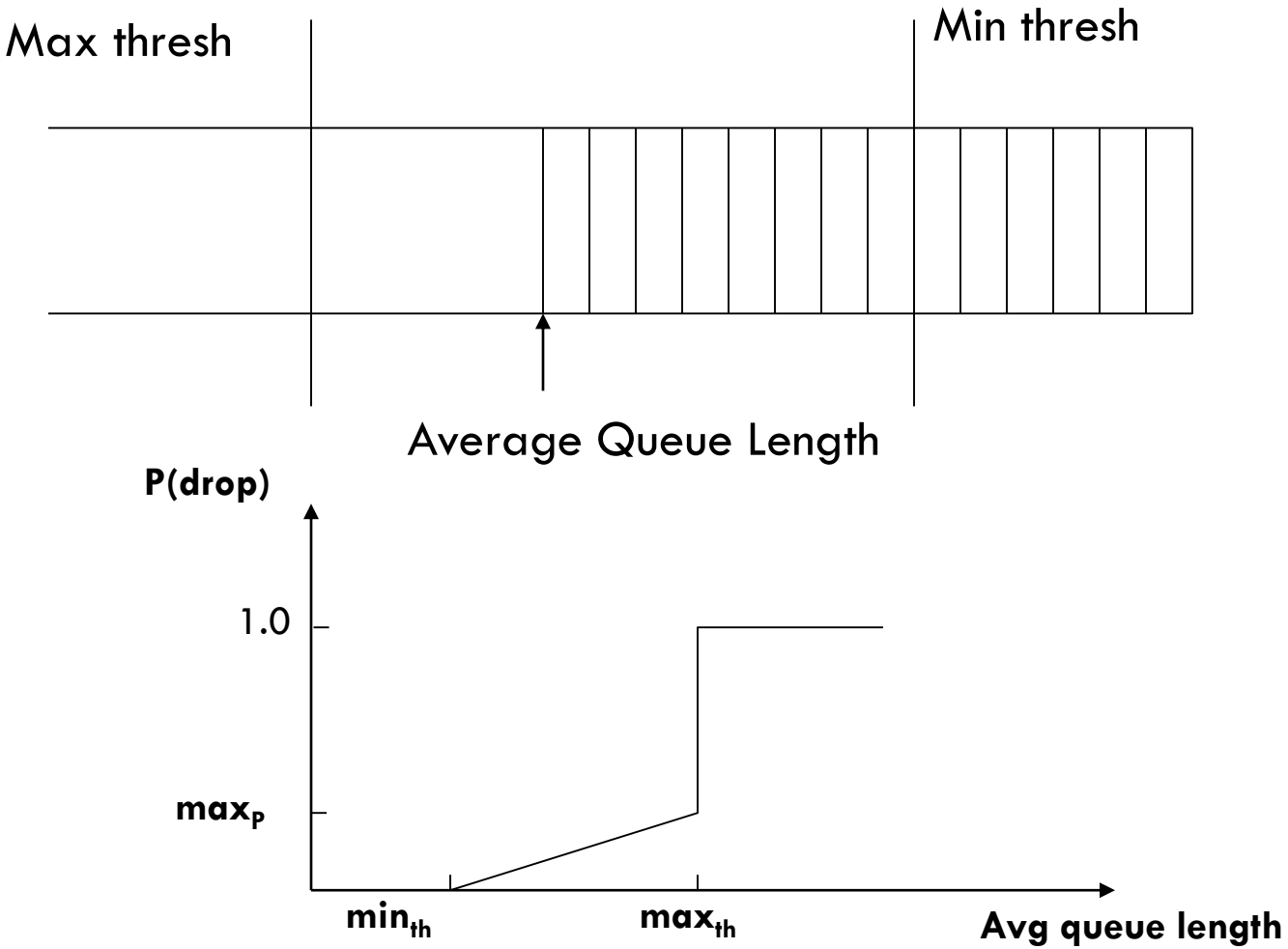
Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
- FIFO (first-in-first-out)
  - Implies single class of traffic
- Drop-tail
  - Arriving packets get dropped when queue is full regardless of flow or importance
- Important distinction:
  - FIFO: scheduling discipline
  - Drop-tail: drop policy

RED Algorithm

- Maintain running average of queue length
- If avgq < min_{th} do nothing
  - Low queuing, send packets through
- If avgq > max_{th}, drop packet
  - Protection from misbehaving sources
- Else mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion
  - E.g. by ECN IP field or dropping packets with a given probability

RED Operation

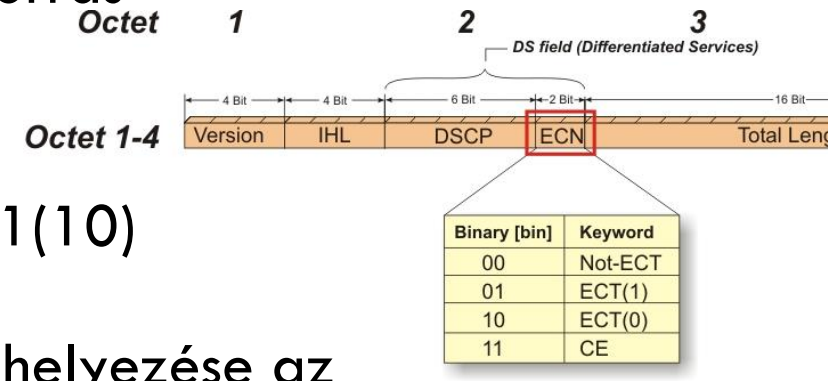


RED Algorithm

- Maintain running average of queue length
- For each packet arrival
  - Calculate average queue size (avg)
  - If $min_{th} \leq avgq < max_{th}$
    - Calculate probability P_a
    - With probability P_a
      - Mark the arriving packet: drop or set-up ECN
    - Else if $max_{th} \leq avg$
      - Mark the arriving packet: drop, ECN

Csomag dobás vagy ECN jelölés

- Csomag dobás
  - Újraküldés szükséges
  - Egyszerűbb megvalósítás
  - Timout lejárta után tud reagálni a forrás
- ECN jelölés
  - Végpont támogatás szükséges
  - Az IP csomag ECT-0 (01) vagy ECT-1(10) jelöléssel
  - Dobás helyett -> ECN CE (11) jel elhelyezése az IP fejlécben
  - A fogadó érzékeli a CE jelet, majd a visszamenő TCP nyugtába bebillent egy ECE flaget, mely jelzi a forrásnak a torlódást
  - Hagyományos TCP (CUBIC, RENO, stb.) források az ECE flaget csomagvesztésnek értelmezik, de újraküldés nem szükséges.



Binary [bin]	Keyword
00	Not-ECT
01	ECT(1)
10	ECT(0)
11	CE

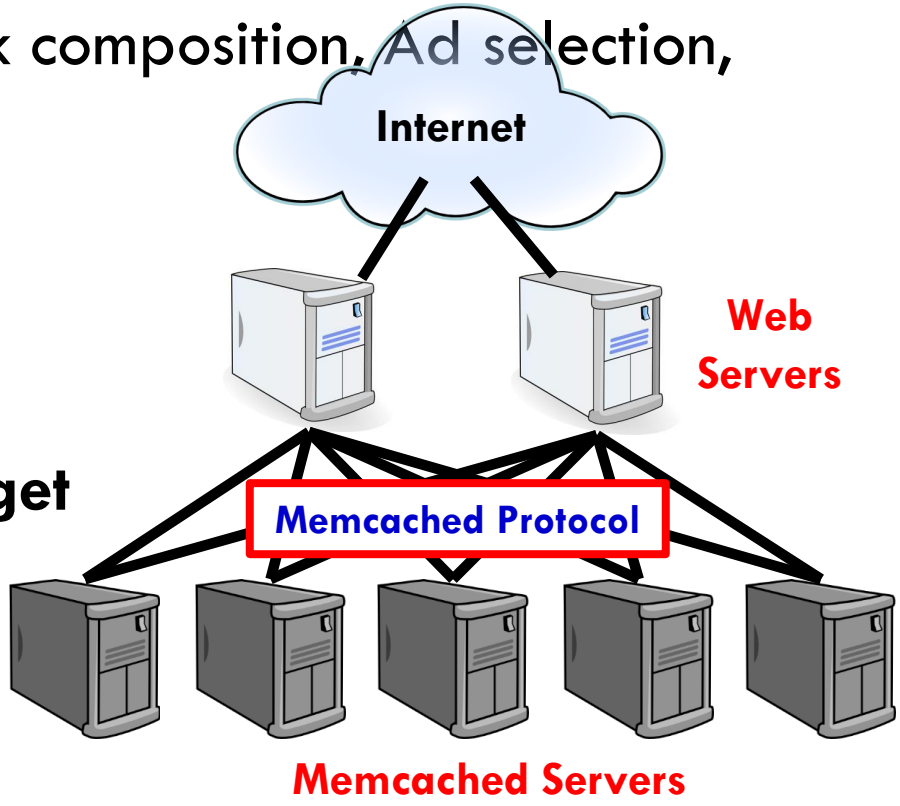
Data Center TCP: DCTCP

Generality of Partition/Aggregate

- The foundation for many large-scale web applications.
  - Web search, Social network composition, Ad selection, etc.
- Example: **Facebook**

Partition/Aggregate ~ Multiget

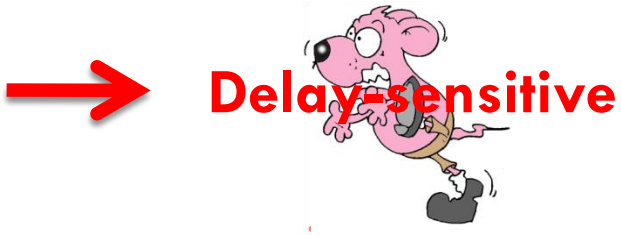
- Aggregators: **Web Servers**
- Workers: **Memcached Servers**



Workloads

- Partition/Aggregate

  (Query)



- Short messages [50KB-1MB]

  (Coordination, Control state)

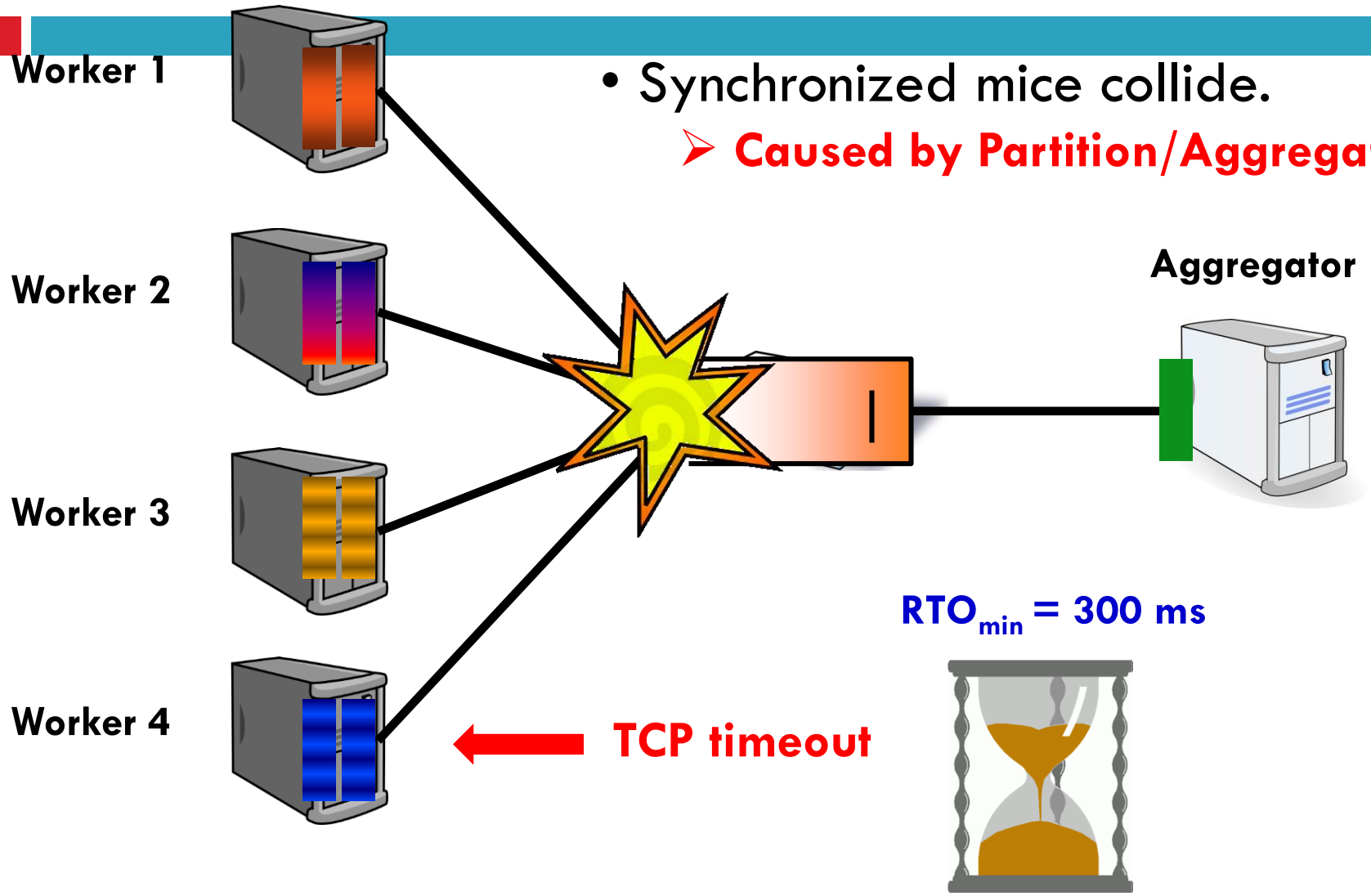
- Large flows [1MB-50MB]

  (Data update)

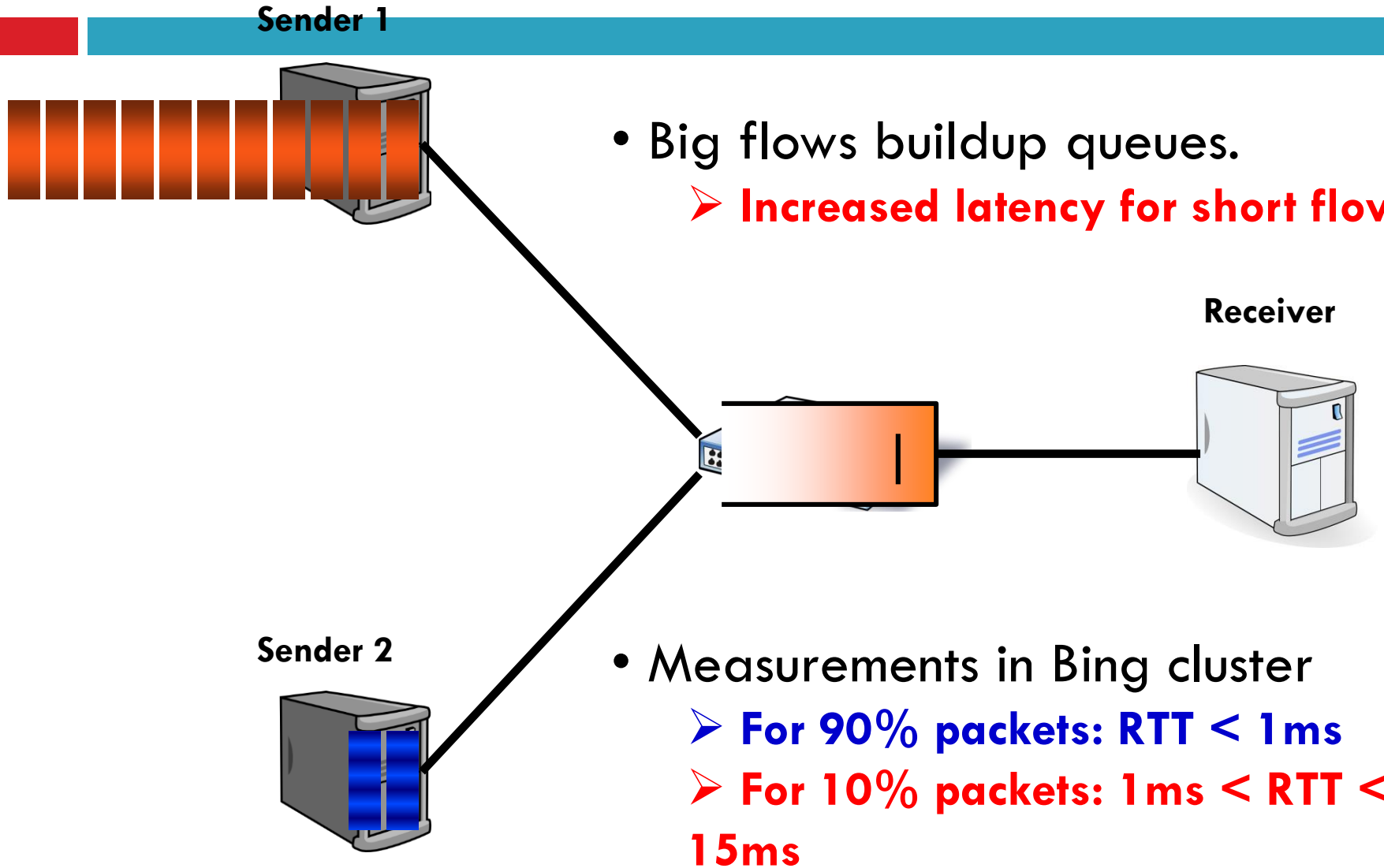
Impairments

- Incast
- Queue Buildup
- Buffer Pressure

Incast


Worker 1
Worker 2
Worker 3
Worker 4
• Synchronized mice collide.
➢ Caused by Partition/Aggregate.
Aggregator
RTO min = 300 ms
TCP timeout

Queue Buildup



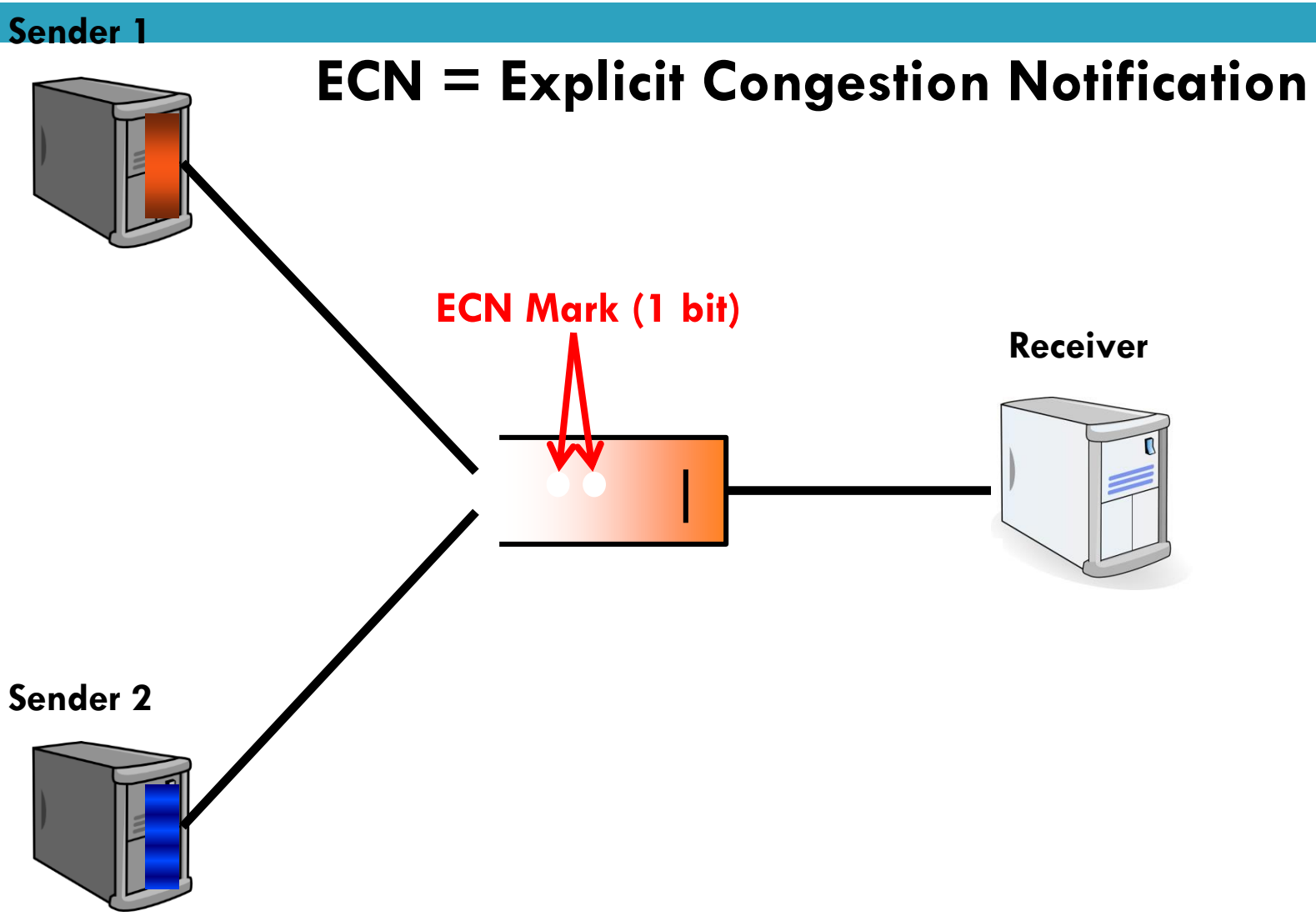
- Big flows buildup queues.
  - **Increased latency for short flows.**
- Measurements in Bing cluster
  - **For 90% packets: RTT < 1ms**
  - **For 10% packets: 1ms < RTT < 15ms**

Data Center Transport Requirements

1. **High Burst Tolerance**
   - Incast due to Partition/Aggregate is common.
2. **Low Latency**
   - Short flows, queries
3. **High Throughput**
   - Continuous data updates, large file transfers

The challenge is to achieve these three together.

DCTCP: The TCP/ECN Control Loop



DCTCP: Two Key Ideas

1. React in proportion to the **extent** of congestion, not its **presence**.
   - ✓ Reduces **variance** in sending rates, lowering queuing requirements.

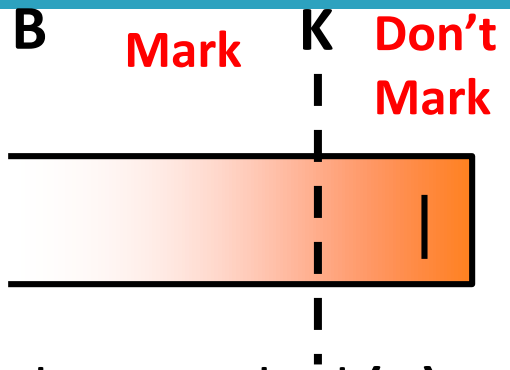
ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

2. Mark based on **instantaneous** queue length.
   - ✓ Fast feedback to better deal with bursts.

Data Center TCP Algorithm

Switch side:

- Mark packets when **Queue Length > K.**



Sender side:

– Maintain running average of ***fraction*** of packets marked **(α)**.

In each RTT:

$$F = \frac{\#\ of\ marked\ ACKs}{Total\ \#\ of\ ACKs} \qquad \alpha \leftarrow (1-g)\alpha + gF$$

➢ **Adaptive window decreases:** $Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$

– Note: decrease factor between 1 and 2.

Köszönöm a figyelmet!