



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

HÁLÓZATOK ALAPJAI ÉS ÜZEMELTETÉSE

TCP torlódáskezelés, QUIC

2019. február 26.

Zsóka Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék

zsoka@hit.bme.hu

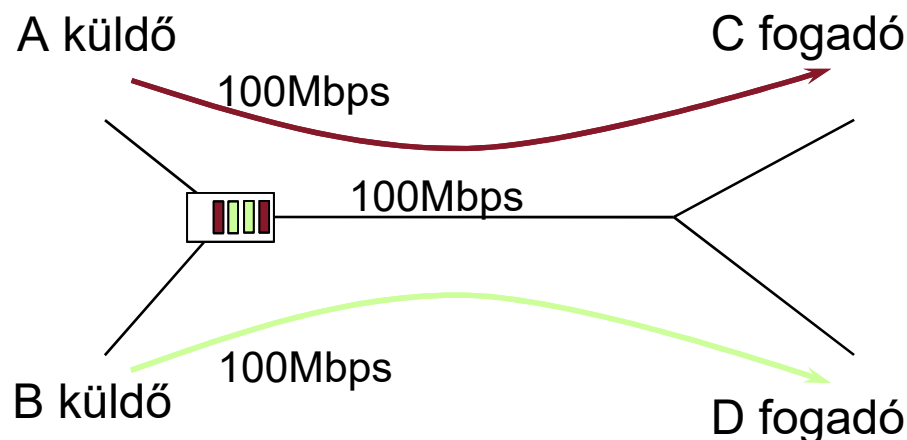


1. Torlódáskezelés a TCP-ben
2. TCP átbocsátás és igazságosság
3. További TCP funkciók
4. A QUIC protokoll

A fóliák elkészítéséhez felhasználtuk Jim Kurose és Keith Ross „Számítógép hálózatok működése” című könyvéhez készült fóliákat.

A TÚLTERHELÉS PROBLÉMÁJA

- A probléma okai:
 - Minden küldő a lehető legnagyobb sávszélességgel küldi az adatokat
 - A küldők összes forgalma meghaladja egy link átocsátó kapacitását
- **Végtelen kimeneti puffer** esetén
 - A csomagok várakozási ideje végtelen nagyra nőhet
 - A lejáró időzítők miatti újraküldések csak súlyosbítanak
- **Véges kimeneti puffer** esetén
 - A várakozási idő korlátos (de nagy lehet)
 - Csomagvesztés lép fel
 - Újraküldés veszteség miatt is
- A csomagok **torlódása (congestion)**
 - Jó lenne elkerülni
 - A küldő oldal tud érte tenni

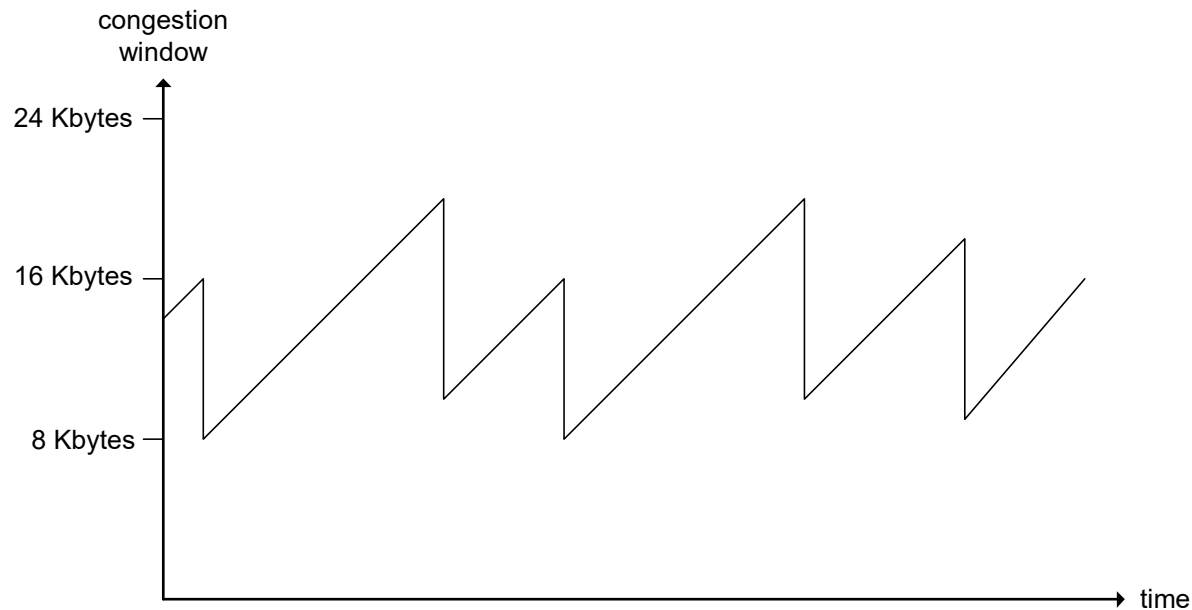


- A TCP-ben kihasználhatók a megbízható szállításra használt mechanizmusok
- A **végpont-végpont** forgalom információi
 - Nem a hálózati eszközök jelzik a torlódást
 - Késleltetési információk a nyugták alapján
 - Csomagvesztési (szegmensvesztési) információk a nyugták hiánya vagy késése alapján
 - Nem tudni, hogy pontosan hol van torlódás (akár több helyen is lehet)
- Gyorsan kell cselekedni, ezért egyszerű mechanizmus kell
- Sebességillesztési szolgáltatás
 - Nem a fogadóhoz, hanem **a hálózat átbecsátóképességéhez illeszkedik**

AIMD – ADDITÍV NÖVELÉS, MULTIPLIKATÍV CSÖKKENTÉS

- A következő nyugtáig átküldhető adatok mennyiségét korlátozó ablak: torlódási ablak, **CongWin**
- Megközelítés:
 - Ha minden rendben – növeljük az ablakot
 - Ha baj (csomagvesztés) van – csökkentjük az ablakot
- Additív növelés (increase)
 - Lineáris növelés: egy MSS-sel növeljük az ablakot minden RTT-ben
- Multiplikatív csökkentés (decrease)
 - Exponenciális csökkentés: a felére csökkentjük az ablakot ha elveszett egy csomag

- Fűrészfogszerű viselkedés
- A cél a maximális, még átvihető sávszélességgel küldeni
- Torlódás esetén erőteljes beavatkozás



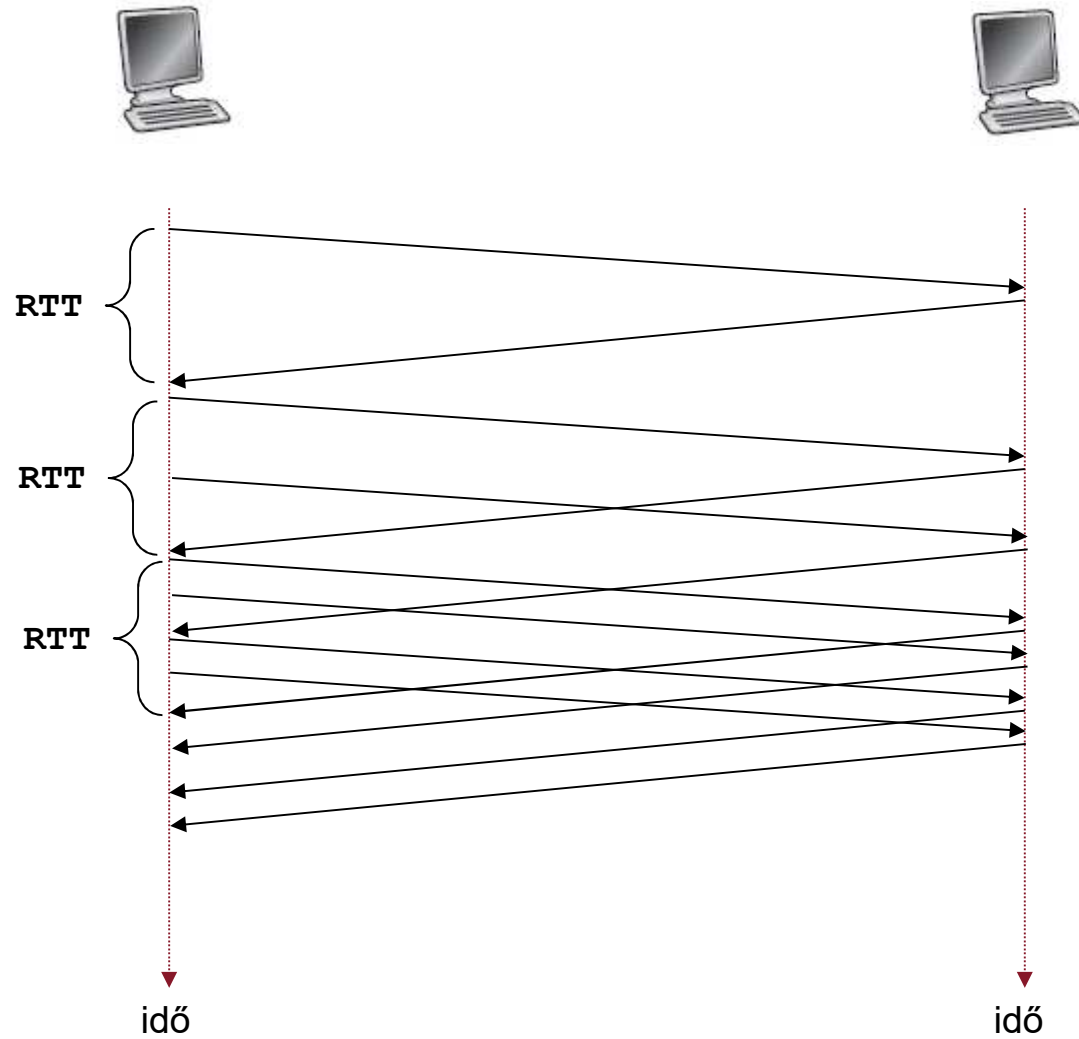
- A küldő betartja az alábbi szabályt
 - **$\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$**
- Átlagos sávszélesség (megközelítőleg)
 - **$\text{CongWin} / \text{RTT}$**
- A torlódási ablak dinamikusan követi a hálózat állapotát
- Minden nem duplikált nyugta érkezésekor a nyugtázott mennyiségnek megfelelően növeljük meg az ablakot
- Torlódás érzékelése a küldőnél
 - Időtúllépés nyugta várása közben
 - Három duplikált nyugta érkezése
- Az ablakot szabályozó mechanizmusok
 - **Lassú indulás (Slow Start, SS)**
 - **Torlódás-elkerülés (Congestion Avoidance, CA)** – AIMD
 - Időtúllépés esetén nagyon konzervatív

- A kapcsolat kezdetén

$$\text{CongWin} = 1 \text{ MSS}$$

- A tényleges átbocsátóképességnél jóval kisebb lehet
- Például
 - MSS: 500 bájt
 - RTT: 20 ms
 - Torlódási ablak: 500 bájt
 - Kezdeti sávszélesség: 200kbps
- Gyorsan fel kellene futtatni
 - Exponenciális növelés
 - Amíg nem lép fel csomagvesztés
- Minden nyugtaérkezéskor egy MSS-sel növeljük az ablakot
- Minden RTT alatt megkétszereződik (megközelítőleg)

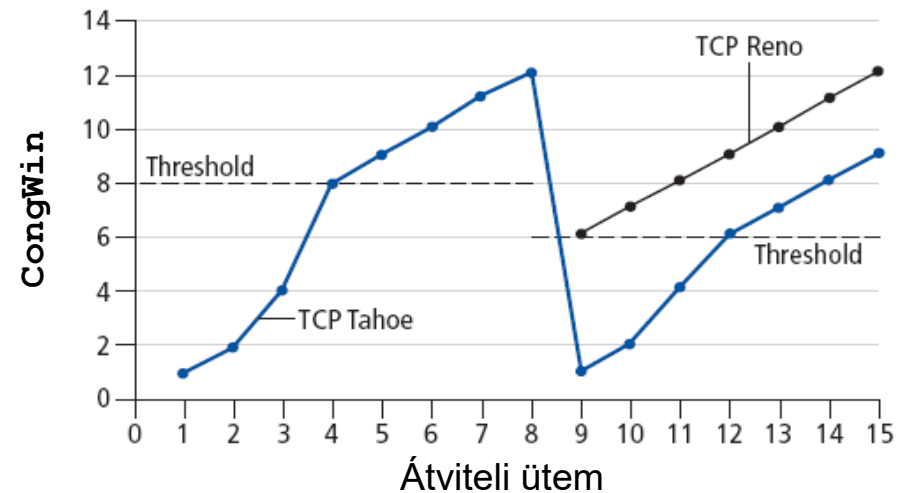
LASSÚ INDULÁS – ILLUSZTRÁCIÓ



- Három duplikált nyugta esetén
 - A hálózaton még átküldhetők csomagok
 - Fast Retransmit újraküldés
 - Felezve az ablakot folytathatjuk additív növekedéssel (CA)
- Időtúllépés után
 - Nagyobb a baj, nem jönnek nyugták
 - Kezdjük előlről, lassú indítással (SS)
- TCP változatok
 - Tahoe – nem tesz különbséget, mindig SS-sel folytatja
 - Reno – hatékonyabb a torlódáskezelés
 - További változatok (NewReno, SACK, FACK, Vegas, CUBIC, stb.) – tovább csiszolnak bizonyos részeken, pl. más függvényeket használnak
 - Nem csak a torlódáskezelésben különbözhetnek, extra opciókat is használnak

- Mikor kellene az exponenciális növekedésből lineáris növekedésre átállni?
 - Az első vesztes helyett egy korábban beállított küszöbnél
 - Amikor elérjük a legutolsó veszteséssel végződő ablak felét

- Megvalósítás
 - **Threshold** változó
 - Csomagvesztés után az ablak felére állítjuk be

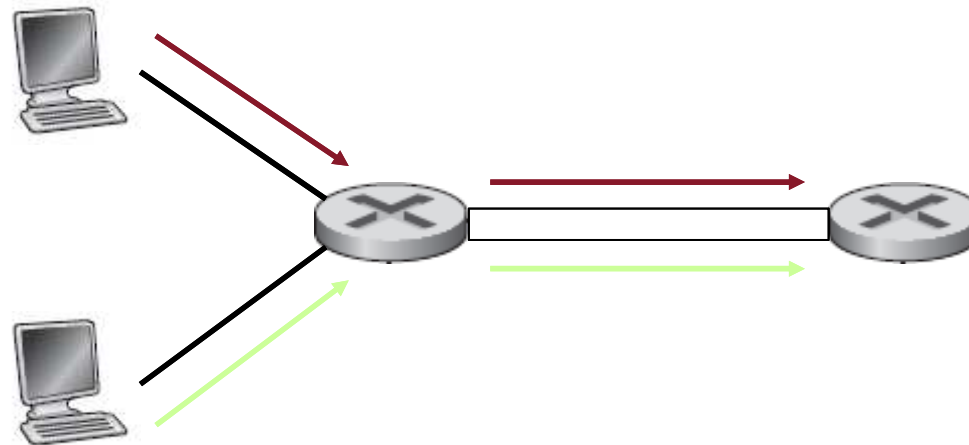


- Ha **CongWin** kisebb mint **Threshold** akkor Slow Start fázis, exponenciális ablaknövelés
- Ha **CongWin** nagyobb mint **Threshold** akkor Congestion Avoidance fázis, lineáris ablaknövelés
- Miután három duplikált nyugta jön, a **Threshold** és a **CongWin** változót is $\text{CongWin}/2$ –re állítjuk
- Ha időtúllépés (timeout) lép fel, a **Threshold** változót $\text{CongWin}/2$ –re, a **CongWin** változót 1 MSS –re állítjuk

1. Torlódáskezelés a TCP-ben
2. TCP átbocsátás és igazságosság
3. További TCP funkciók
4. A QUIC protokoll

- Mekkora átvitel (átbocsátás) adódik ebből a működésből?
 - Figyelman kívül hagyva az SS állapotot
- Tegyük fel, hogy W ablakméretnél lép fel csomagvesztés
 - W méretnél az átvitel W/RTT
 - A felezés után $W/(2*RTT)$ –től lineárisan nő
 - Az átlag $0.75W/RTT$
- Ez nagyon leegyszerűsített modell volt
 - Valójában a hálózat állapotától függ a W és az RTT is
 - A TCP mohón, de szabályozottan fogyasztja a link-kapacitásokat

- A kapcsolatok osztoznak a hálózati erőforrásokon
- Az lenne **fair**, ha átlagban „testvériesen” osztoznának (**fairness**)
 - Egyenletesen?
 - Az átvihető adatmennyiség arányában?



- Annak, hogy csökkentjük is néha az ablakméretet, a fairness a célja
- Példa: két kapcsolat forgalma versenyhelyzetben egy R kapacitású linken
 - Additív ablaknövelés sikeres átvitel esetén
 - Multiplikatív ablakcsökkentés torlódás esetén

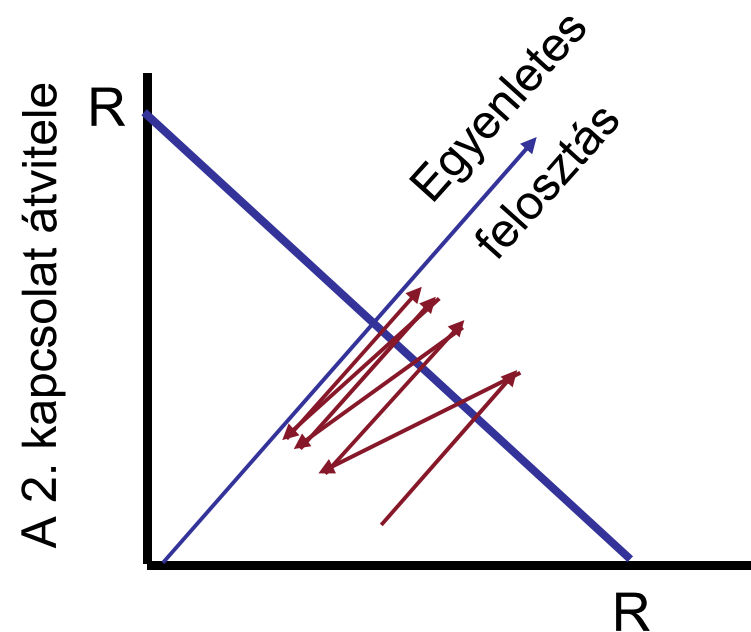
CA: ablakméret lineáris növelése

Csomagvesztés: ablakfelezés

CA: ablakméret lineáris növelése

Csomagvesztés: ablakfelezés

CA: ablakméret lineáris növelése



- Sok kapcsolat esetén azért nem ennyire egyszerű

- UDP
 - A multimédia alkalmazások gyakran nem TCP alapúak
 - Nem nagy baj, ha nem megbízható az átvitel
 - Nem praktikus a torlódáskezelés
 - Konstans sávszélességgel lehet küldeni
 - Veszteség lehet
 - Nem igazán barátságos
- Vannak ötletek torlódáskezelés nélküli megbízható protokollokra is
- TCP torlódáskezelés kihasználása magasabb rétegben
 - Több párhuzamos kapcsolat is nyitható egy applikációban
 - Például web-böngészők
 - Példa: egy R kapacitású linken 9 kapcsolat osztozik
 - Egy újabb applikáció egy kapcsolatot nyit: $R/10$ átbecsátás jut neki
 - Egy újabb applikáció tizenegy kapcsolatot nyit: $R/2$ átbecsátás jut neki

1. Torlódáskezelés a TCP-ben
2. TCP átbocsátás és igazságosság
3. További TCP funkciók
4. A QUIC protokoll

- Egy elveszett szegmens miatt duplikált nyugták jöhetnek
 - Csak azt adják meg, hogy melyik bájt hiányzik
 - A sorrenden kívüli, de **megkapott szegmenseket** nem kellene újraküldeni
 - Nagy RTT esetén nem kapjuk meg időben a kumulatív nyugtát
- **SACK** – a fogadó nyugtában küldi vissza a sorrenden kívüli bájtok tartományait
 - Korlátozott mennyiségben
 - A fejléc opcióit felhasználva
 - A kapcsolat felépítésekor egyeztetnek a felek a használatról
- Az újraküldést önmagában is gyorsíthatja

- Sok szegmens küldése esetén
 - Duplikált nyugták alapján látjuk , hogy melyik szegmenst nem kapta meg a fogadó
 - Gyors újraküldés
- Egy szegmens esetén?
 - Többnyire időtúllépés lesz
- Ötlet
 - Próbáljuk ki, hogy nincs-e csomagvesztés
 - Kis mennyiségű adat újraküldése még az időtúllépés előtt
 - Szelektív nyugtázás adataira támaszkodik

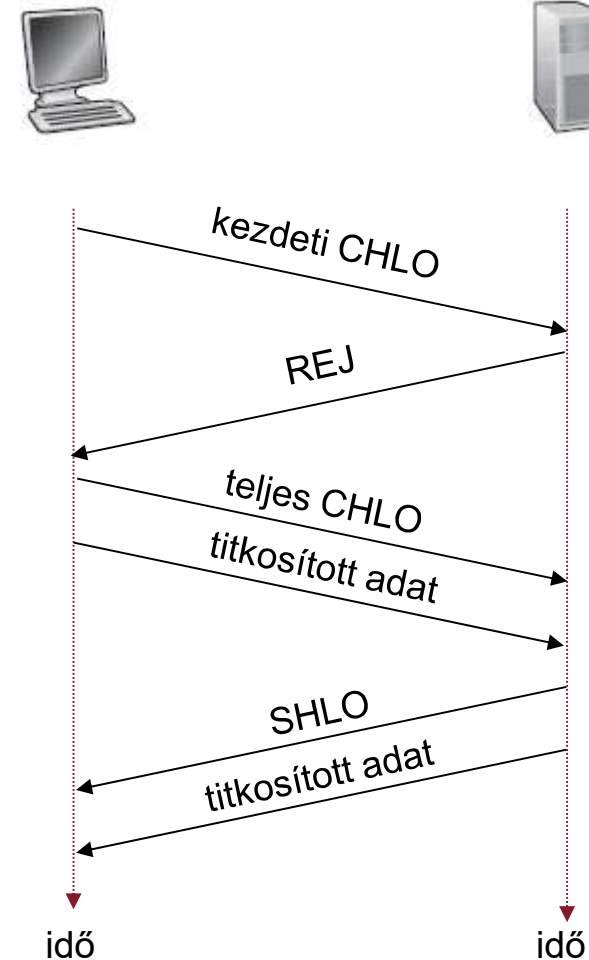
1. Torlódáskezelés a TCP-ben
2. TCP átbocsátás és igazságosság
3. További TCP funkciók
4. **A QUIC protokoll**

- A Google-nál kezdték fejleszteni, jelenleg Internet Draft
- A TCP változatok előnyeit ötvözi
- **Quick UDP Internet Connections**
 - Megbízható adatátvitel UDP felett?
 - Egy fél réteggel feljebb (a szállítási és az alkalmazási rétegek között)
 - Egyelőre nem a TCP helyett, csak mellett
- A HTTP/2 és a HTTP/3 hatékony kiszolgálására

- **Gyors** kapcsolat-felépítés
- **Hatékony** újraküldés és torlódáskezelés
- Adatfolyamok (stream) **multiplexálása**
 - Egy adat nem tartja fel a többi
- Hiányzó adat helyreállítása
 - Forward Error Correction
 - Paritás ellenőrző adat több szegmensre
 - Helyreállítás anélkül, hogy újraküldenénk
- Kapcsolatvégpontok **migrálása**
 - Internet értelemben mozoghatnak a végpontok
 - A kapcsolatnak egyedi azonosítója van
 - Nem a TCP-nél látott négyes információ, tehát akár változhat az IP cím, vagy a port száma
- **Titkosítás** beépítve
 - Az összes adat titkosítva megy át (TLS)

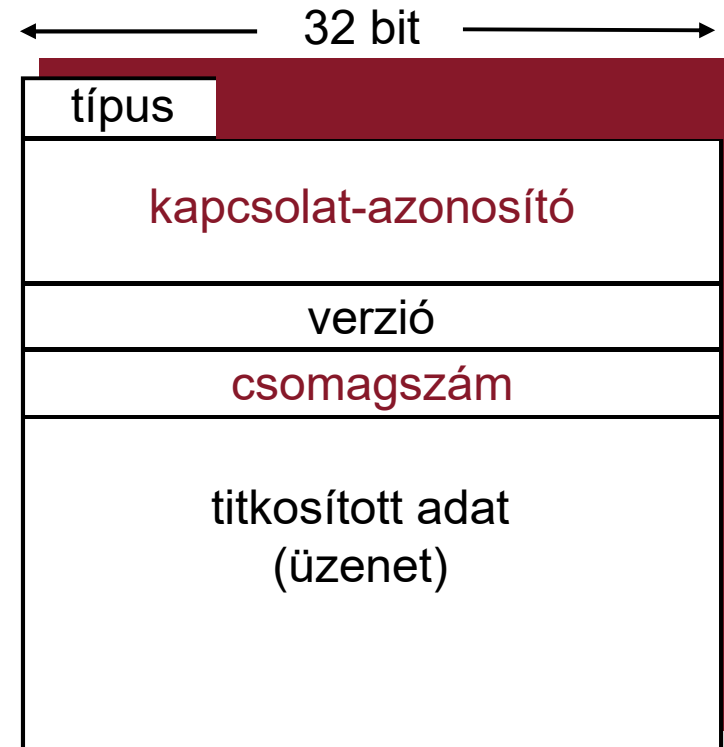
QUIC – KAPCSOLATFELÉPÍTÉS

- Ismeretlen szerver
 - **1-RTT** felépítés
 - CHLO, REJ, SHLO
 - Ebben már a titkosítás kulcskiosztása is benne van
- Ismert szerver
 - **0-RTT** felépítés
 - CHLO/SHLO után egyből küldhető az adat a korábban megismert információk alapján
- Verzió-egyeztetés
 - Kicsit hosszabb lehet
 - Ritkán kell



QUIC - CSOMAGFORMÁTUM

- A csomag az UDP szegmens adata
- Verzióra csak egyeztetéskor van szükség
- **Egyedi csomagszám**
 - Monoton nő, lehet tudni, hogy melyiket küldték előbb
 - Újraküldés esetén már más lesz
- Az adat különböző részekből állhat, pl:
 - Adatfolyamok adatai
 - Nyugta



- A küldésnél torlódáskezelést használ
 - TCP CUBIC (Linux default)
- A nyugta egy azonosított csomagra vonatkozik
 - Időbélyeges
 - RTT számítás könnyebb (csomagonként lehet számláló)
 - **NACK** – inverz SACK információ
 - Explicit megadja, hogy melyik csomagok nem érkeztek még meg
- Újraküldés
 - A NACK miatt nagyon könnyű kezelni
 - TLP mechanizmus is van



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

