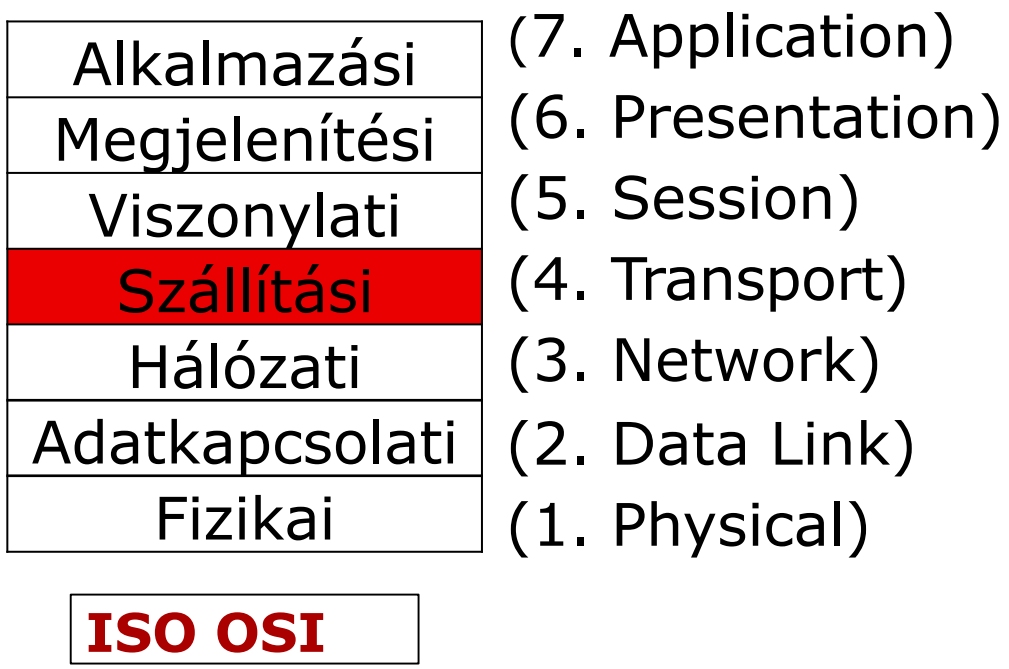
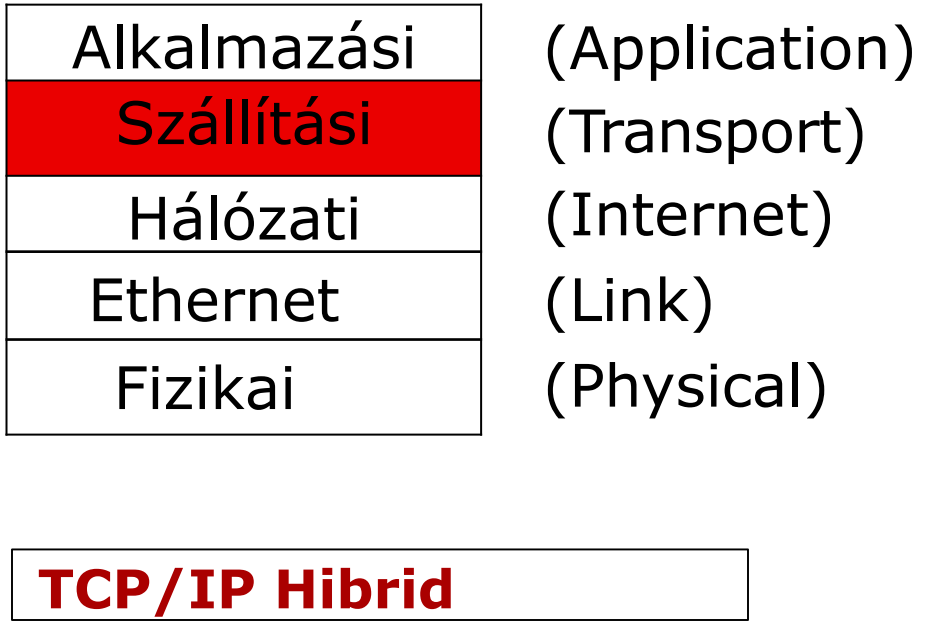
# Hálózatok -

# Szállítási réteg

Varga Tibi 2022

# A TCP és az UDP helye a rétegekben



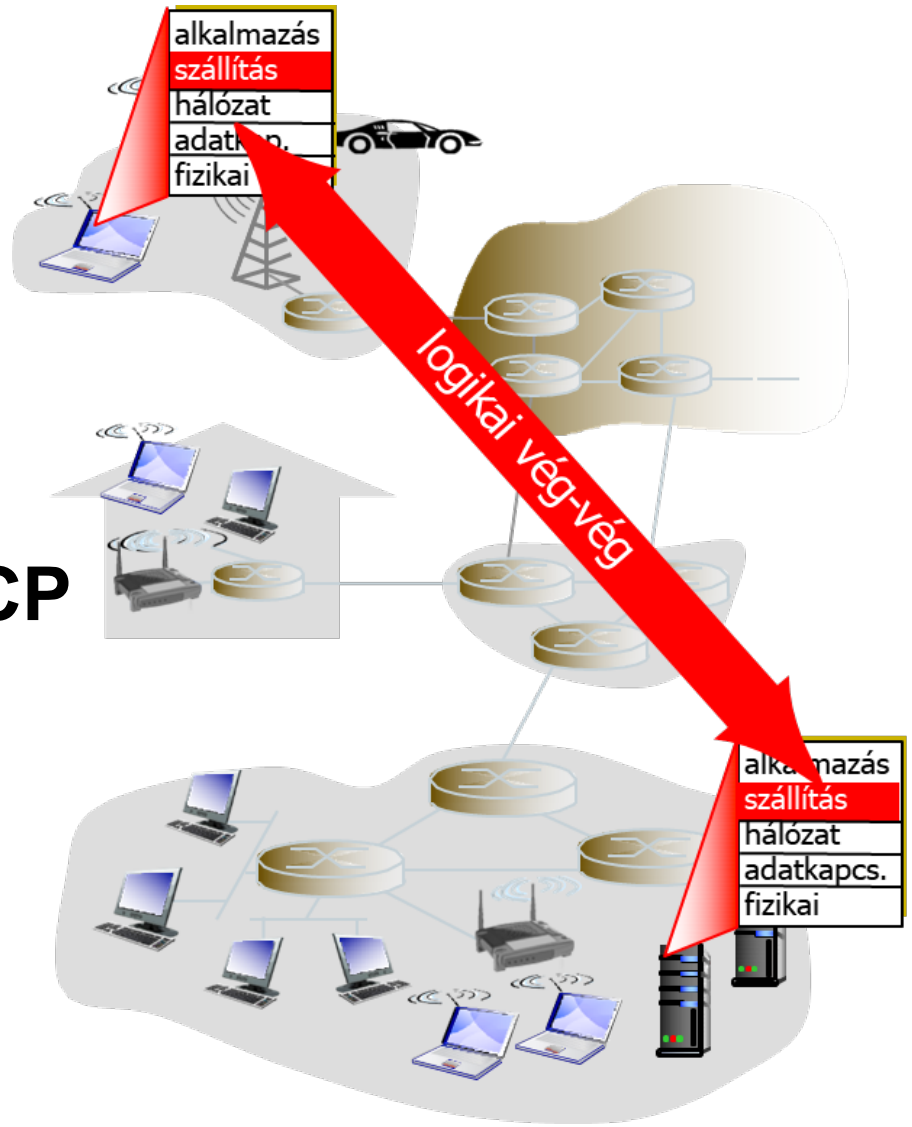

- *hálózati réteg:* datagramok átvitele a végpontok között
- *szállítási réteg:* a végpontokon futó alkalmazások folyamatai (processzek) közötti logikai kapcsolatok
  - a hálózati réteg szolgáltatásainak igénybevételére alapozva

# A szállítási protokollok működési helye

- Logikai kapcsolatok a végpontokon futó alkalmazások folyamatai között
- A szállítási protokollok legtöbbször csak a végpontokban futnak, a  köztes csomópontokban nem
- Az alkalmazások adategységeit *szállítási protokoll-adategység*ekbe (szegmensekbe) tördeljük, a kapottakból pedig összerakjuk
- A szállítási réteg egyfajta kapocs az alkalmazási réteg, valamint a hálózati  átvitelért felelős alsóbb (hálózati) réteg között.

# Szállítási réteg szolgáltatások

- Multiplexálás / demultiplexálás
- Adatok szegmensekre bontása és a szegmensek adattá egyesítése
- **Kapcsolat mentes** átivtel**: UDP**
- **Kapcsolatorientál**t átviteli protokoll**: TCP**
- A megbízható átvitel biztosítása
- Folyamat vezérlés
- Torlódás vezérlés



# Szállítási protokollok: TCP és UDP

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

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

- Alapvető **különbség** az TCP és az UDP között:
  - A TCP összeköttetés-alapú (connection-oriented) transzport- szolgáltatást nyújt
  - Az UDP összeköttetés-menteset (connectionless)

# Gyakori szállítási protokollok

**TCP** megbízható sorrendhelyes továbbítás

- Garantált szolgáltatás
- torlódás vezérlés
- folyam vezérlés
- kapcsolat felépítés és fenttartás

**UDP** megbízhatatlan, nem rendezett kézbeítés:

- az IP "legjobb szándék„ szerinti kiegészítése Tehát nincs garancia hogy az adat célba ér és arra sem ha odaért akkor sorrendben érkezett

Amire **nincs** szolgáltatás:

- késleltetés garancia
- sávszéleség garancia

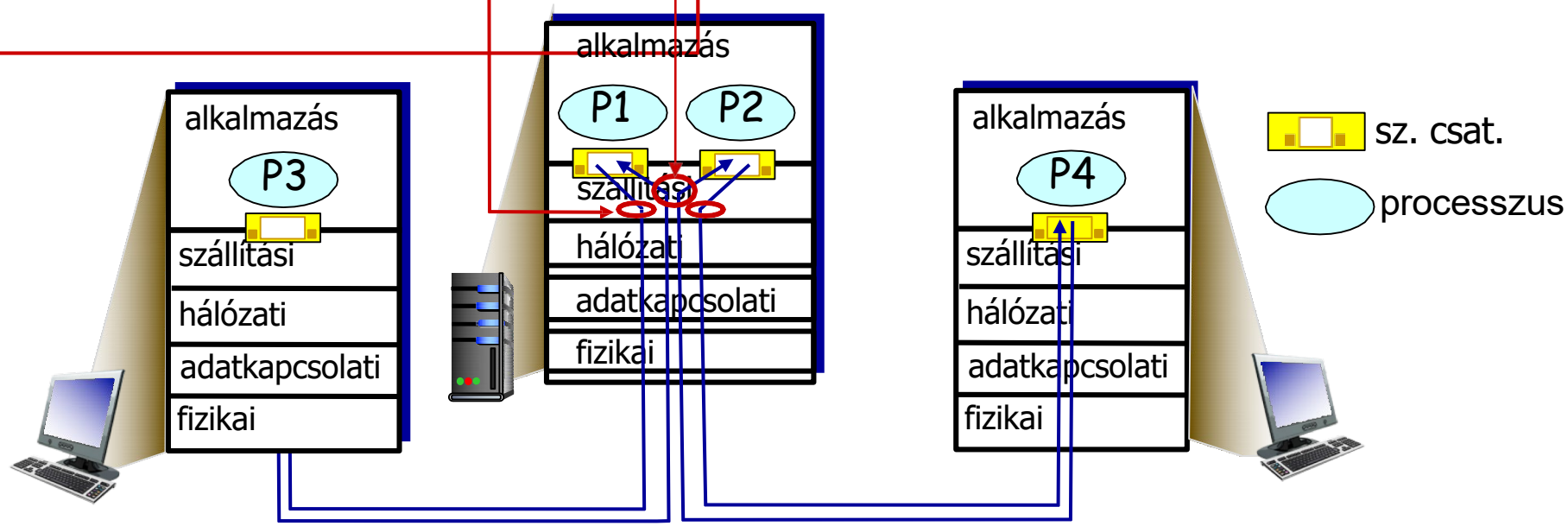
# 1. Multiplexálás / demultiplexálás

*multiplexálás a küldőnél:*

több szoftver több folyamat adatait kezeli hozzáadja az szállítási réteg fejlécét (ezt használják a demultiplexálásra)

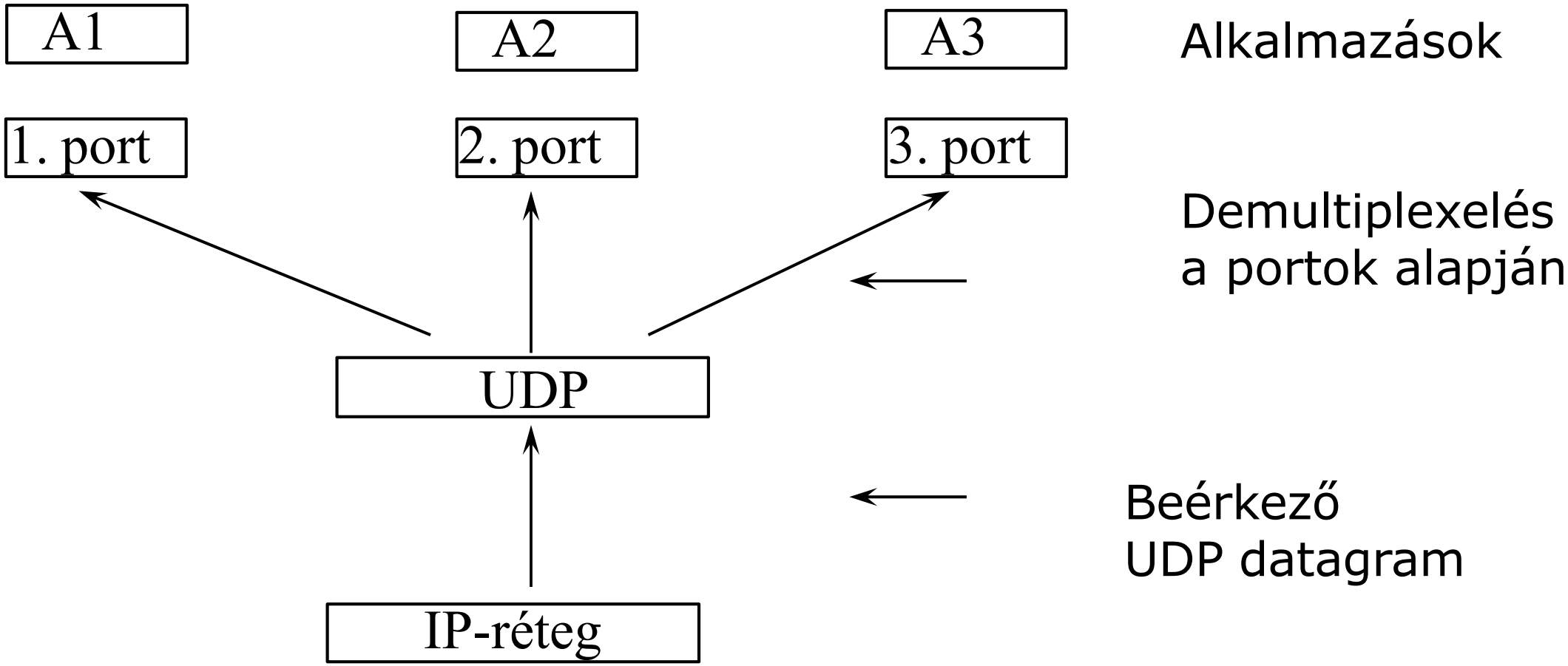
*demultiplexálás a fogadónál:*

a fejléc információ port számait használja a megkapott szegmensek kézbesítésére



# Multiplexelés és demultiplexelés

Példa:



# A TCP és az UDP közös képességei

## Portok kezelése

- Az IP-rétegben a csomagok a végberendezéseknek vannak címezve
- A végberendezéseken belül több alkalmazás, azon belül folyamat futhat egyszerre (pl.: böngésző több nyitott ablakkal és közben e-mail levél fogadás IMAP )
- Megkülönböztetésük: portok használatával
- Portok: 16 bites számok (0-tól 65535-ig terjedő értékkészlettel)
- Adott alkalmazás eléréséhez a megfelelő porthoz kell csatlakozni.
- Egyes alkalmazások TCP-t, mások UDP-t használnak (és vannak, amelyek mindkettőt)

- Multiplexelés/demultiplexelés
  - A portmechanizmus segítségével lesz megvalósítva

# Portszámok szabványos kiosztása

**Három tartományt definiáltak**

- **0 - 1023**: System Ports (korábban: Well Known Ports)
  - Ezeket a portokat használják az alapvető, általánosan használt hálózati szolgáltatások. Ezekhez a portokhoz csak privilegizált szerver programok kapcsolódhatnak szolgáltatás nyújtása céljából.

- **1024 - 49151**: User ports (korábban: Registered Ports)
  - Ezeken a portokon egyéb szolgáltatások érhetők el. (Ha valaki kinál egy szolgáltatást, igényelhet hozzá ilyen portszámot.)

- **49152 - 65535**: Dynamic and/or Private Ports
  - Ebből a tartományból az operációs rendszer oszt ki portokat a kommunikációhoz a programoknak.

# 2. Kapcsolat mentes átvitel

- Gyors ″sallangments, ″nyers″ Internet szállítási protokoll
- ″legjobb szándék” szerinti szolgáltatás, az UDP szegmensek:
  - elveszhetnek
  - az alkalmazásnak sorrendet nem betartva kézbesíthetőek

***kapcsolatmentes:***

- nincs kézfogás az UDP küldő és
  - fogadó között
- minden UDP szegments függtelenül kezelnek

UDP-t használ:

- médiafolyam stream alkalamzások (késleltés, sávszélesség érzékeny)
- DNS
- SNMP

UDP feletti megbízható átvitel:

- alakalmazás rétegben kell ezt biztosítani
- Alkalmazás specifikus hibakezelés!

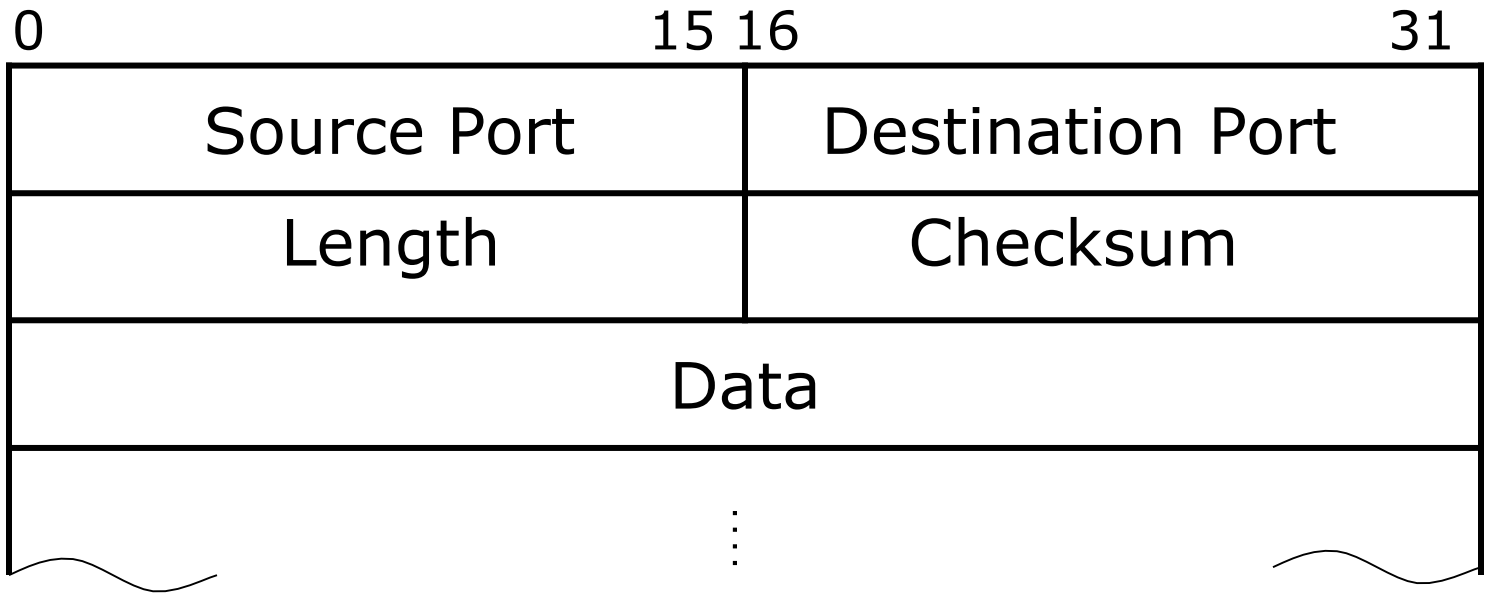
# UDP

# USER DATAGRAM PROTOCOL

# UDP ( User Datagram Protocol )

- Az UDP egyszerűbb szállítási rétegbeli protokoll, ami **nem kínál megbízhatóságot** és adatfolyam-vezérlést, ami azt jelenti, hogy kevesebb fejlécmezőre van szüksége
- Mivel a küldő és fogadó UDP-folyamatoknak nem kell kezelniük a megbízhatóságot és az adatfolyam-vezérlést, ez azt jelenti, hogy az UDP-datagramok **gyorsabban** feldolgozhatók.
- Az UDP csupán alapfunkciókat biztosít az adatcsomagok (datagramok) alkalmazások között történő szállítása során, így nagyon **csekély többletterhelést** okoz és adatellenőrzést sem végez.
- Az UDP az adatokat **datagramokra** bontja, amelyeket **szegmensekne**k is nevezünk. Az UDP egy összeköttetés-mentes protokoll.
- Mivel az UDP nem kínál megbízhatóságot vagy adatfolyam-vezérlést, **nincs szüksége kapcsolat létrehozására.**
- Mivel az UDP nem követi nyomon a kliens és a szerver között küldött vagy fogadott információkat, az UDP-t **állapot nélküli** protokollnak is nevezzük.

# A User Datagram felépítése



# A User Datagram mezői

- **Source Port** (16 bit)
  - A portszám azon a gépen, ahonnan a user datagramot küldték.
- **Destination Port** (16 bit)
  - A portszám azon a gépen, ahova a user datagramot küldték.
- **Length** (16 bit)
  - A user datagram hossza oktettekben. (min. 8)
- **Checksum** (32 bit)
  - Ellenőrző összeg
  - Opcionális: ha nincs, akkor a mező értéke: 0.

# UDP ellenőrzőösszeg

*Cél:* az átvitt szegmensben "hibák" detektálása (pl.: átfordult bitek)

## küldő:

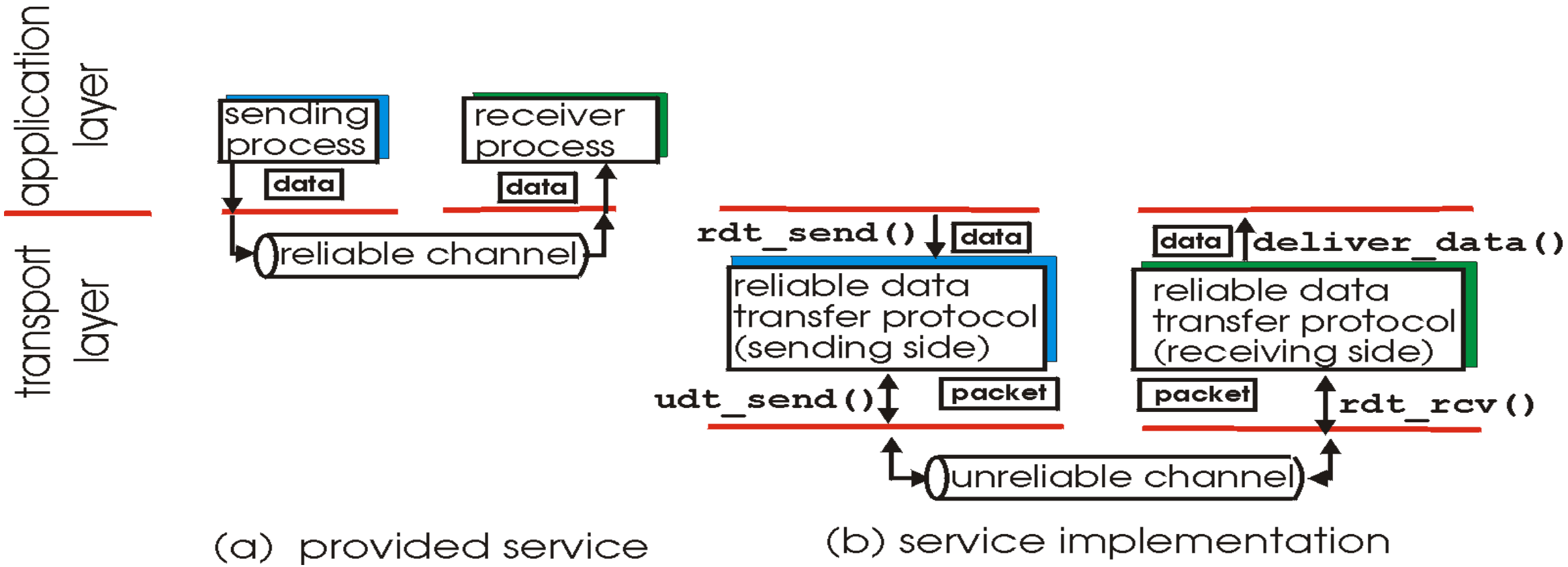
- a szegmens tartalmát (a fejlécet is) mint 16 bites integer számok sorozataként kezeli
- ellenőrző összeg: a szegmens tartalmának összeadása (egyes komplemens)
- a küldő azellenőrző összeget az UDP ellenőrző összeg mezejébe teszi

## fogadó:

- kiszámítja a kapott szegmens ellenőrző összegét
- megnézi, hogy egyezik-e a kiszámított és a kapott:
  - NEM – hibát detektált
  - IGEN- nincs hiba detektálva. *De lehet mégis hiba?* ….

# 3. Kapcsolatorientált átviteli protokoll

- a hálózatok 10 legfontosabb problémájának egyike




- a megbízhatatlan csatorna tulajdonságai határozzák meg a megbízható átvitel protokoll komplexitását

# TCP

# TRANSMISSION CONTROL PROTOCOL

# TCP (Transmission Control Protocol)

**A TCP egy megbízható, teljes körű szállítási rétegbeli protokoll, amely garantálja az összes adat célba érkezését** sorszámozás és pozitív nyugtázás segítségével**.**

**A TCP egy összeköttetés-alapú k**ommunikációt hoz létre a végpontok folyamatai között mely a kapcsolat felépítésével kezdődik és a lebontással fejeződik be

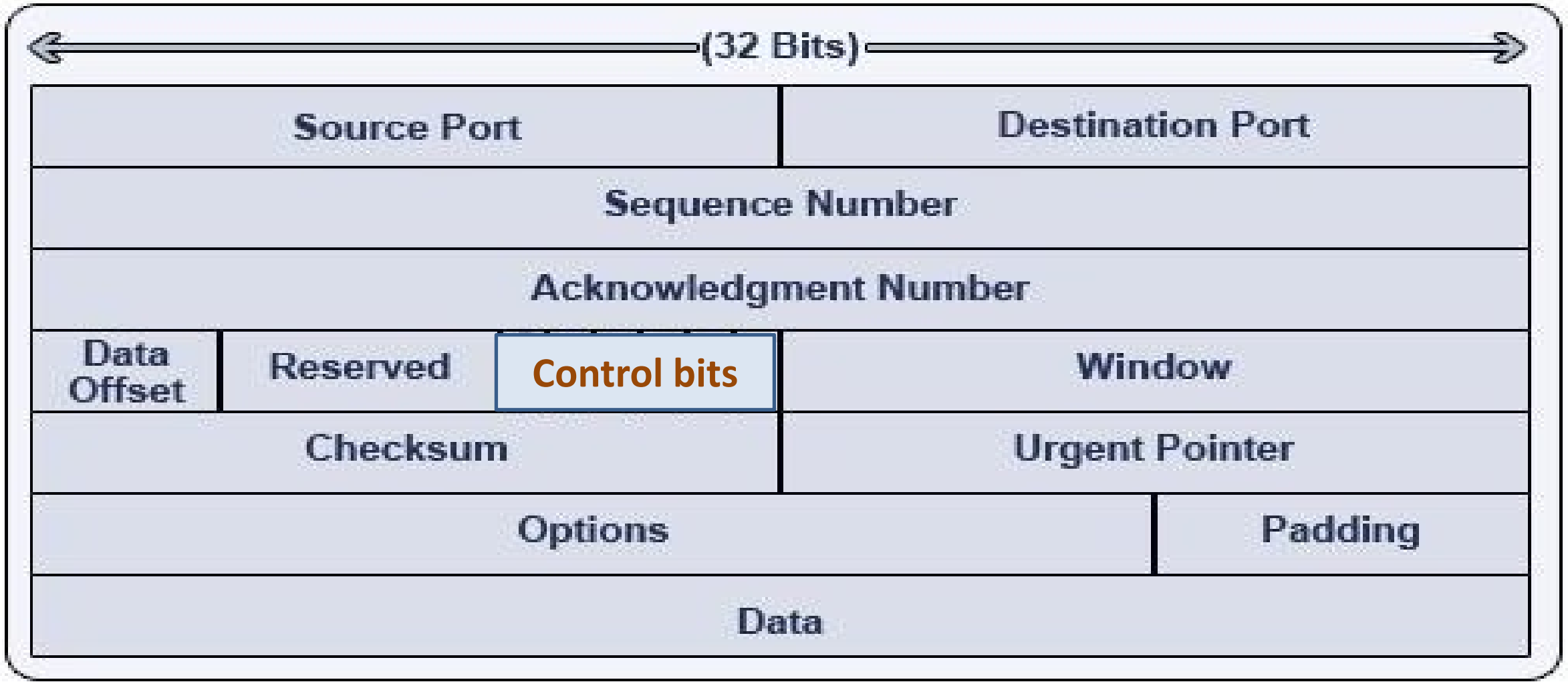
**A TCP szegmensekre** bontja az adatokat. A TCP szegmensek sokkal több pluszinformációt tárolnak, mint az UDP, így nagyobb lesz a csomagméret is.

**A TCP olyan mezőket tartalmaz,** amelyek **biztosítják** az alkalmazásadatok kézbesítését. Ezek a mezők további feldolgozást igényelnek a küldő és fogadó állomásokon.

**Forgalomszabályozás** van (flow control) az ablakmechanizmus segítségével

**Torlódásvezérlés** (congestion control)

# TCP szegmens és fejléc



# A TCP szegmens mezői – 1

- **Source Port** (16 bit)
  - A portszám azon a gépen, ahonnan a szegmenst küldték.
- **Destination Port** (16 bit)
  - A portszám azon a gépen, ahova a szegmenst küldték.
- **Sequence Number** (32 bit)
  - Az átvitel során az oktetteket sorszámozzuk. A sorszám megmutatja, hogy a szegmens adatmezőjének kezdő oktettje hányas sorszámot kapott.
- **Acknowledgement Number** (32 bit)
  - Nyugta szám: a visszaigazoláshoz szükséges mezők (ha ACK=1) azon oktet sorszáma az adatfolyamban, amelyet a vevő a legközelebb beérkező szegmensben elvár, ha az adatok átvitele hibátlan. Ez az ellenkező irányú adatfolyamra vonatkozik!

# A TCP szegmens mezői – 2

- **Data Offset** (4 bit)
  - 32 bites egységekben mérve megadja az adatmező kezdetét a TCP szegmens kezdetéhez képest. Tulajdonképpen a fejrész hossza.
- **Reserved** (4 bit)
  - Későbbi használatra fenntartva.
- **Control bits** (8 bit)
  - RFC 793 szerint csak 6 darab volt, az RFC 3168 vezette be a CWR és az ECE vezérlőbiteket a korábban fenntartott 6 bites terület végén.
  - A vezérlőbiteknél mindig azok 1 értéke esetén áll fenn az, amit a nevük jelent.

# A TCP szegmens mezői – 3

**Control bits -** A vezérlőbitek és értelmezésük

- **CWR** (Congestion Window Reduced)
  A torlódási ablakot szűkítették.
- **ECE** (ECN-Echo) - RFC 3168
  Torlódás jelző visszhang.
- **URG** (urgent)
  A sürgős adat mutató érvényes.
- **ACK** (acknowledgment)
  Ez egy  nyugta ha mező értéke 1.
- **PSH** (push)
  Jelzi az adat késedelem nélküli továbbításának igényét.

# A TCP szegmens mezői – 4

**Control bits** - A vezérlőbitek és értelmezésük (folytatás)

- **RST** (reset)
  - Egy host összeomlása, vagy összezavart összeköttetés helyreállítására szolgál, illetve ha egy porton semmilyen program sem figyel.
- **SYN** (synchronize)
  - Összeköttetés létesítésére szolgál a szegmens ha az értéke 1 .
  - A kapcsolat felépítés és lebontás esetén jelez
- **FIN** (finish)
  - Összeköttetés bontására szolgál jelzés

- **Window** (16 bit)
  - Az ablakméretet adja meg. A forgalomszabályozáshoz szükséges mechanizmus használja.

# A TCP szegmens mezői – 5

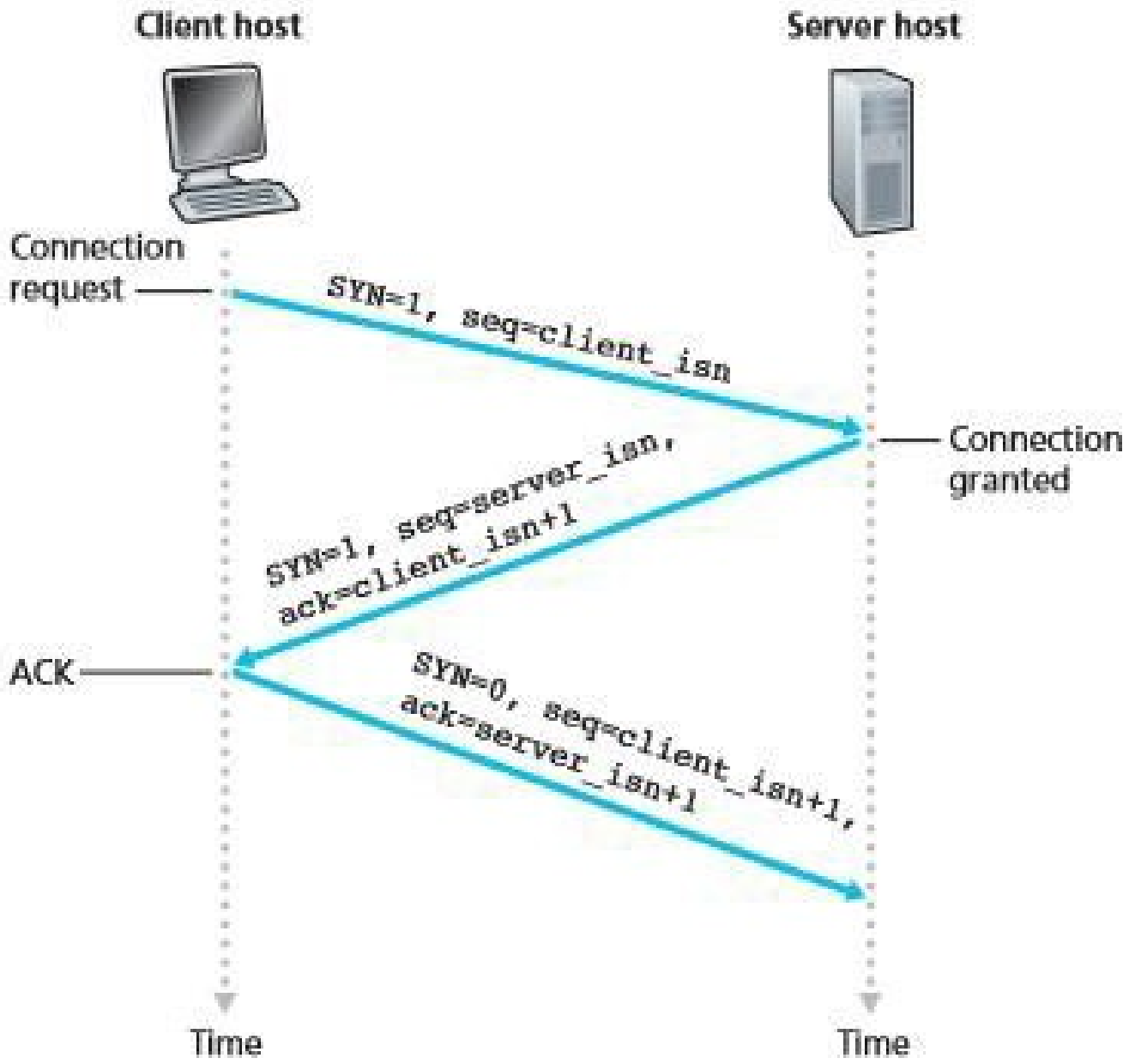
- **Checksum** (16 bit)
  - Ellenőrző összeg
  - A számítás ugyanazzal a módszerrel történik, mint az UDP-nél.
- **Urgent Pointer** (16 bit)
  - A *sürgős adat* az adatfolyam menetét megszakítva a többi adatot megelőzve feldolgozandó adat. A sürgős adat az adatmező elejétől kezdődik, a mutató a sürgős adat után következő (már nem sürgős) adat kezdetére mutat.
- **Options**, **Padding**
  - Options field: Nem mindig van kitöltve és változó méretű lehet. Nagysebességű hálózatoknál használja a küldő és a fogadó a szegmensméret egyeztetésére.
  - Padding: Ha szükséges lehet helykitöltésre.

# TCP kapcsolat felépítése

Egy TCP adattovábbításnak egy nem túl érdekes, ugyanakkor nagyon fontos része a kapcsolat felépítése.

**Ez egy háromlépéses folyamat**, melynek során adat nélküli TCP szegmenseket küld egymásnak a két hoszt (kliens és szerver) és megállapodnak az átvitel részleteiről.

A három lépés miatt szokták ez a folyamatot „Three-way handhsakenek”  is hívni.



# A TCP háromfázisú kézfogásának célja

Az **állomások nyomon követik az egyes adatszegmenseket** egy munkameneten belül, valamint a TCP-fejlécben szereplő adatok alapján arról is információt cserélnek, hogy mely adatok érkeztek meg.

A TCP egy teljes duplex (full-duplex) protokoll, ahol minden egyes kapcsolat két, egyirányú kommunikációs adatfolyam.

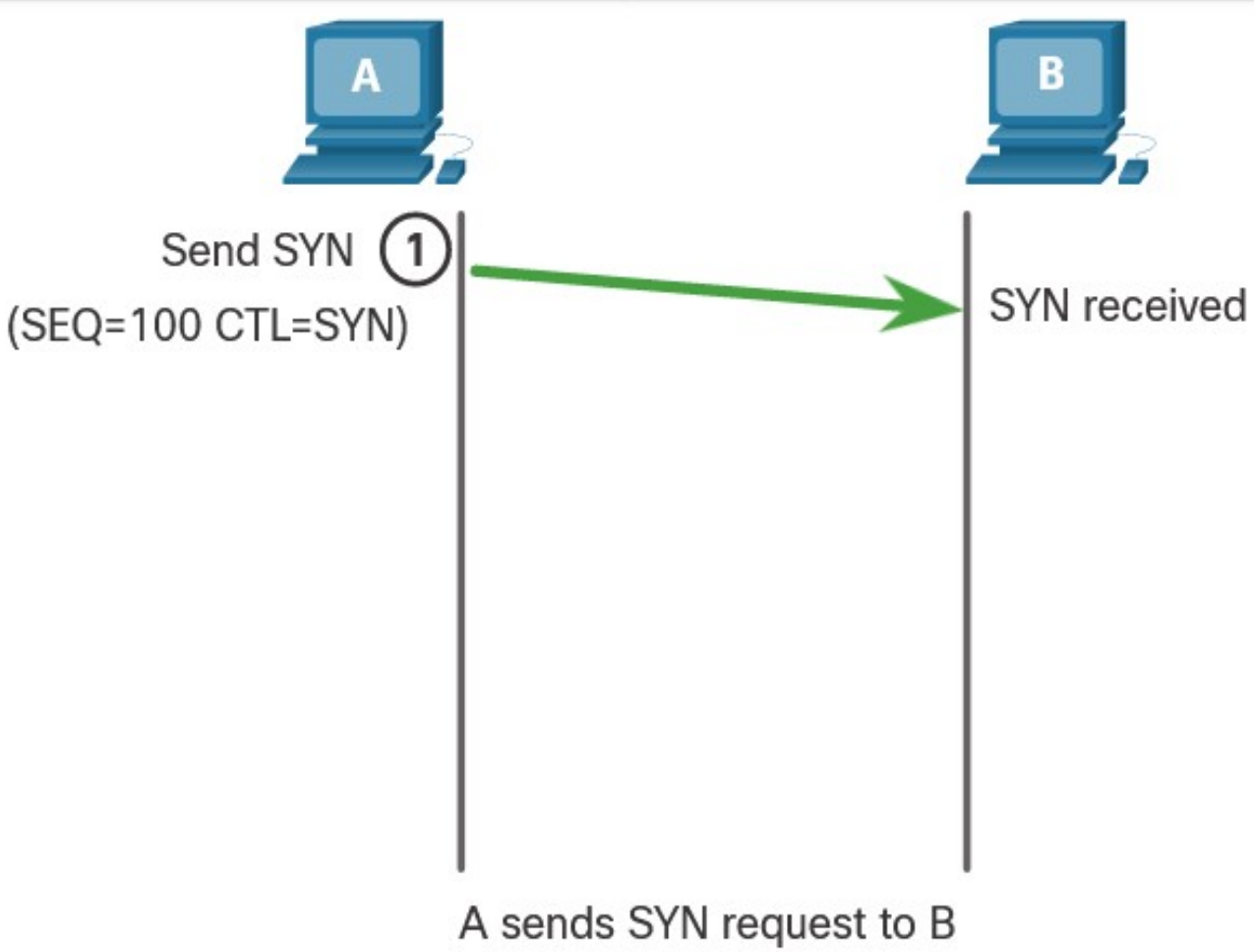
**A háromfázisú kézfogás feladatai az alábbiak:**

1. Megállapítja a céleszköz jelenlétét a hálózaton.
2. Ellenőrzi, hogy a céleszköz aktív szolgáltatással rendelkezik-e, és elfogadja-e az olyan célportra érkező kéréseket, amelyet a kezdeményező kliens használni kíván a munkamenet idejére.
3. Tájékoztatja a céleszközt arról, hogy a küldő kliens kapcsolatot kíván létrehozni az adott porton.

Miután a kommunikáció befejeződött, a munkameneteket le kell zárni, a kapcsolatot pedig bontani kell. A kapcsolathoz és munkamenethez tartozó mechanizmusok adják a TCP megbízhatóságát.

# 1. Kapcsolat kezdeményezése

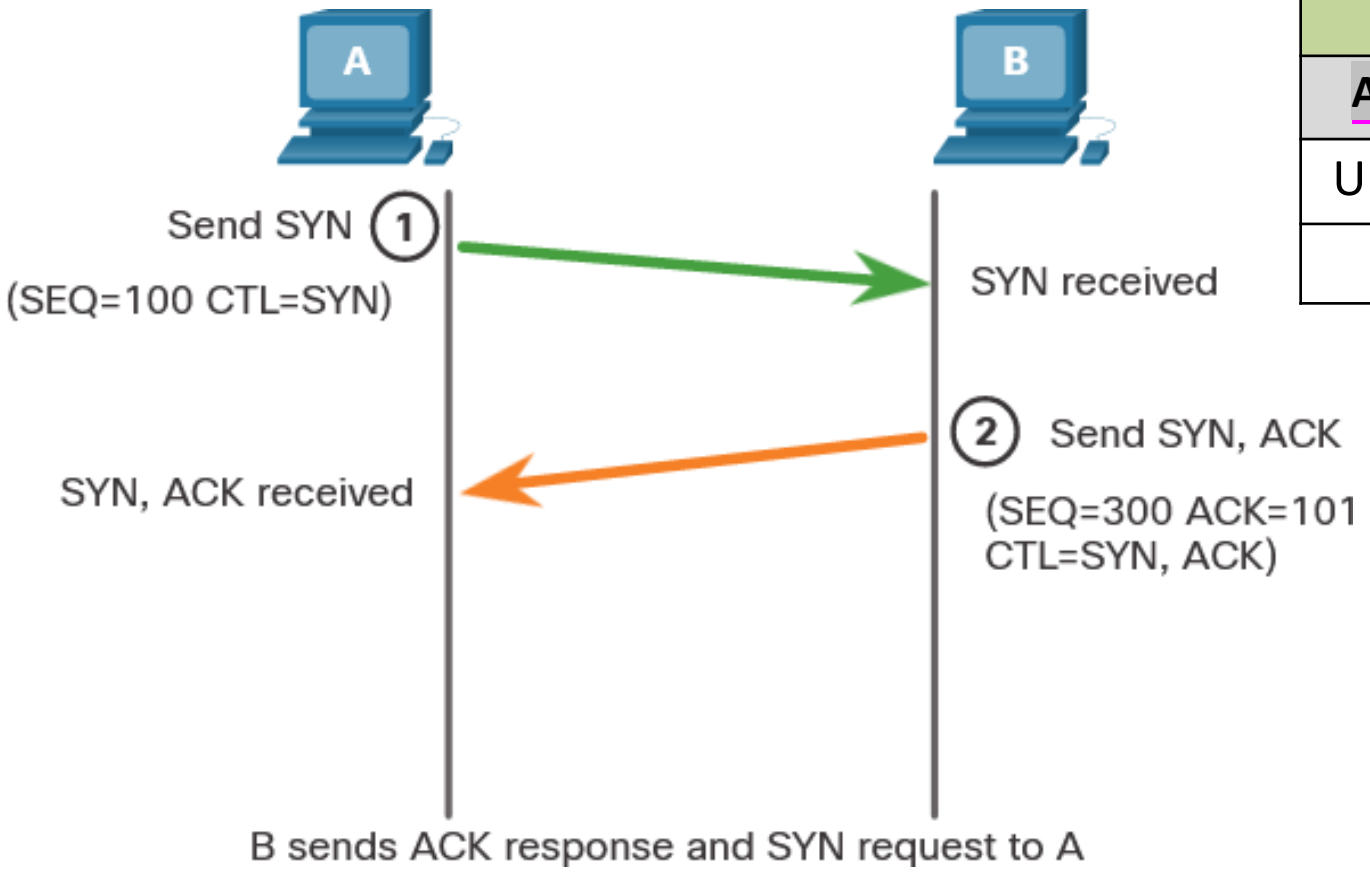
1. Kliens küld egy üres TCP szegmenst a szervernek. Ezt nevezzük SYN szegmensnek.




| Seqence Number | | | | | client_sn 100 |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | |
| URG | **ACK** | PSH | RST | **SYN** | **FIN** |
| | 0 | | | 1 | |

# 2. Válasz a szervertől

2. Ha a SYN csomag megérkezik a fogadóhoz (szerver), a fogadó előké-szíti a TCP buffert a fogadásra, valamint visszaküld egy válaszszegmenst. Ebben a szegmensben szintén nincs adat
Ezt a csomagot szokták **SYNACK szegmensnek is nevezni**

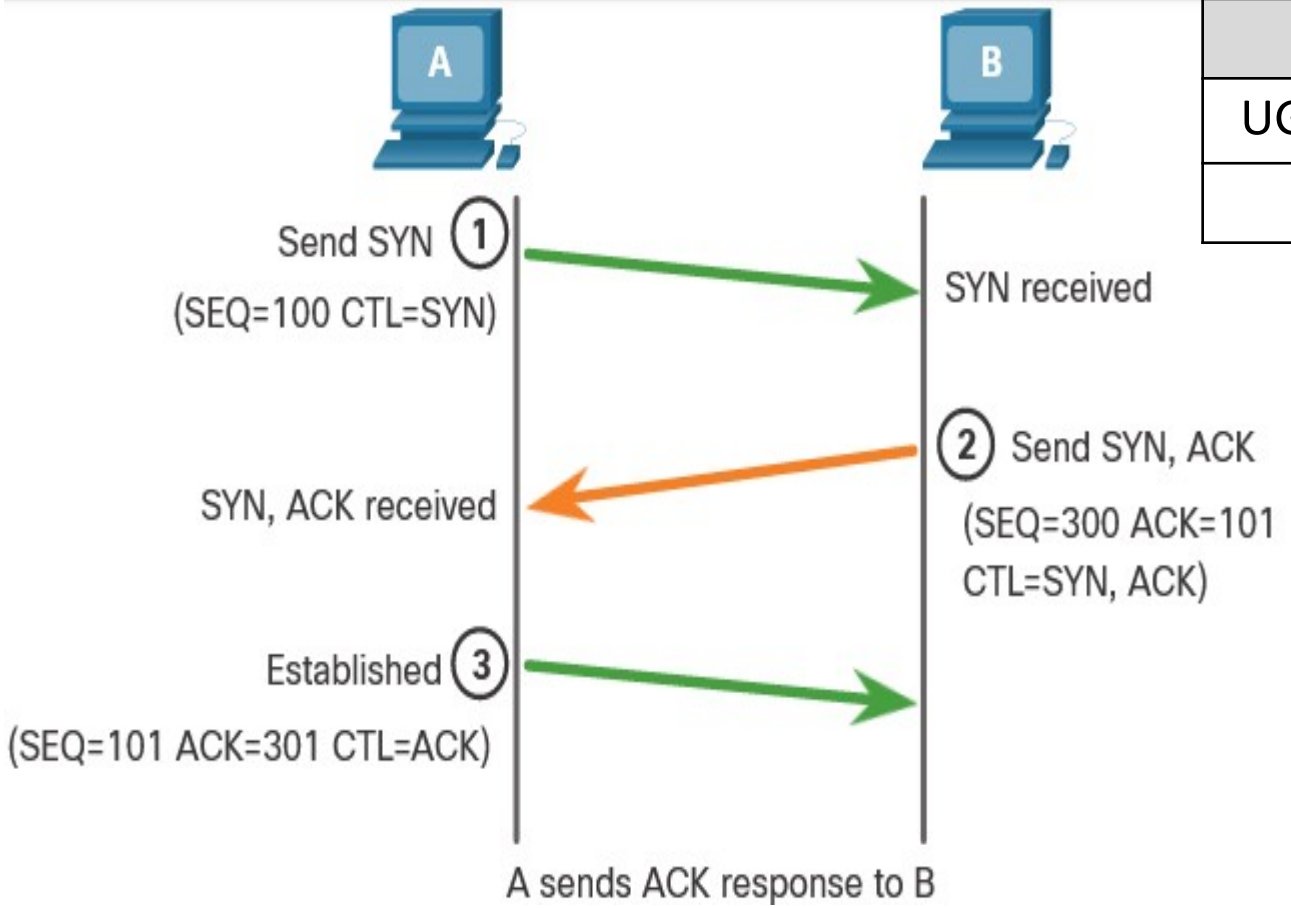



| Seqence Number | | | | | serv_sn (300) |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | client_sn+1 (101) |
| URG | ACK | PSH | RST | SYN | FIN |
| | 1 | | | 1 | |

# 3. Nyugtázás

3. Ha a kezdeményező kliens megkapta a SYNACK szegmenst, akkor ő is felkészül az adatok fogadására
A kezdeményező ügyfél nyugtázza a szerver-kliens irányú kapcsolat létrehozását. Ez az ACK szegmens.

| Seqence Number | | | | | client_sn+1 101 |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | Serv_sn+1 (301) |
| UGR | ACK | PSH | RST | SYN | FIN |
| | 1 | | | 0 | |




Ha ez a három lépés sikeresen lejátszódott, akkor innentől kezdve a kliens és a szerver küldhetnek egymásnak adatokat. Ezekben a szegmensekben a SYN mező már végig 0 lesz.

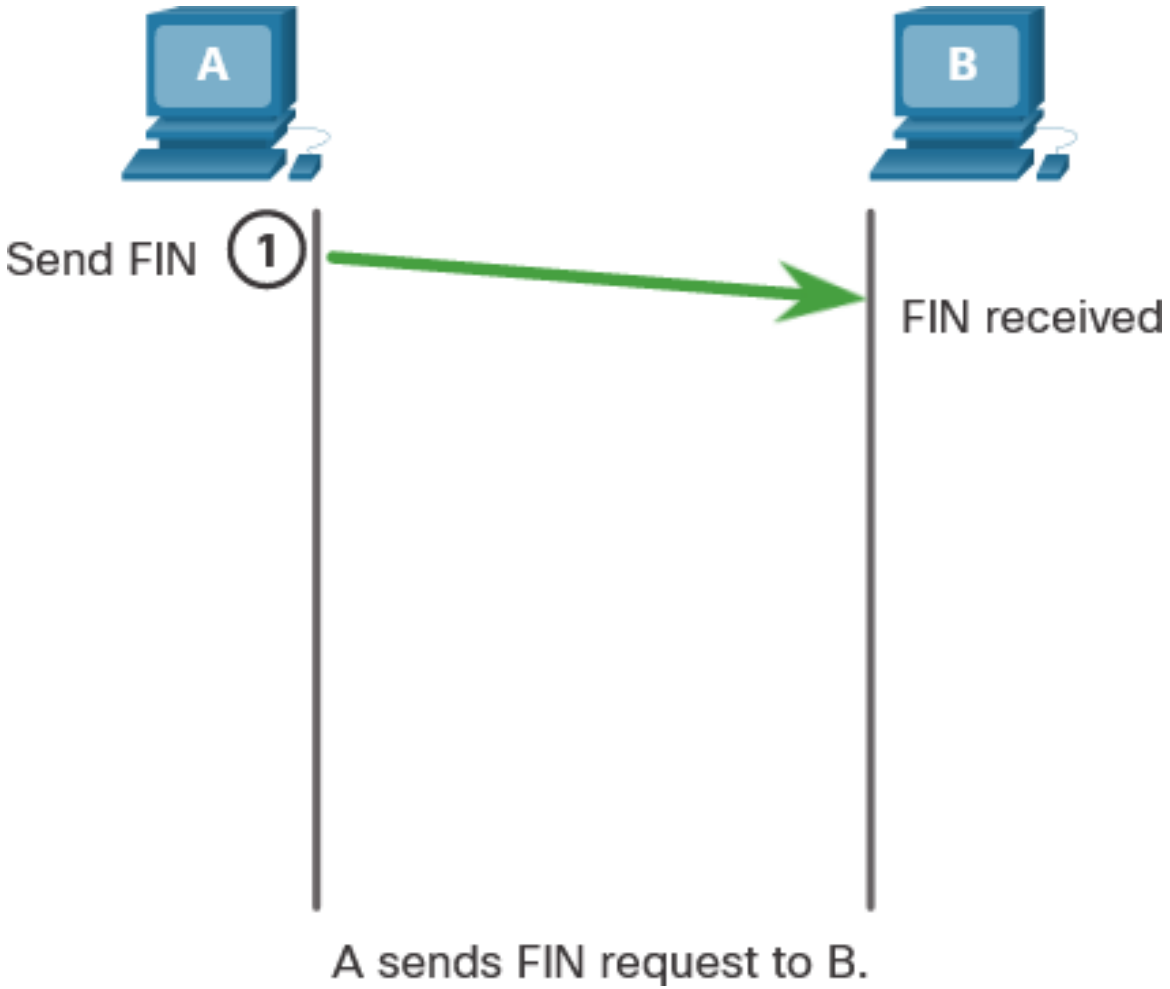
# TCP kapcsolat lezárása

A kapcsolat lezárásához be kell állítani a szegmens fejlécében található FIN vezérlőbit értékét**. Minden egyes, egyirányú TCP-munkamenet lezárásához kétfázisú kézfogást használunk, amely egy FIN és egy ACK szegmensből áll**. Ezért egy TCP által támogatott párbeszéd bontásához, mindkét munkamenetet meg kell szüntetni, amelyhez négy adatcsere szükséges. A kliens és a szerver egyaránt kezdeményezheti a lezárást.

A  lezárás folyamatát bármely, nyitott munkamenettel rendelkező állomás kezdeményezheti.

# 1. lépés: FIN (kliens)

1. Amikor már nincs több átküldendő adat, a kliens egy olyan szegmenset küld, amelyben a FIN jelzőbit beállítása megtörtént.

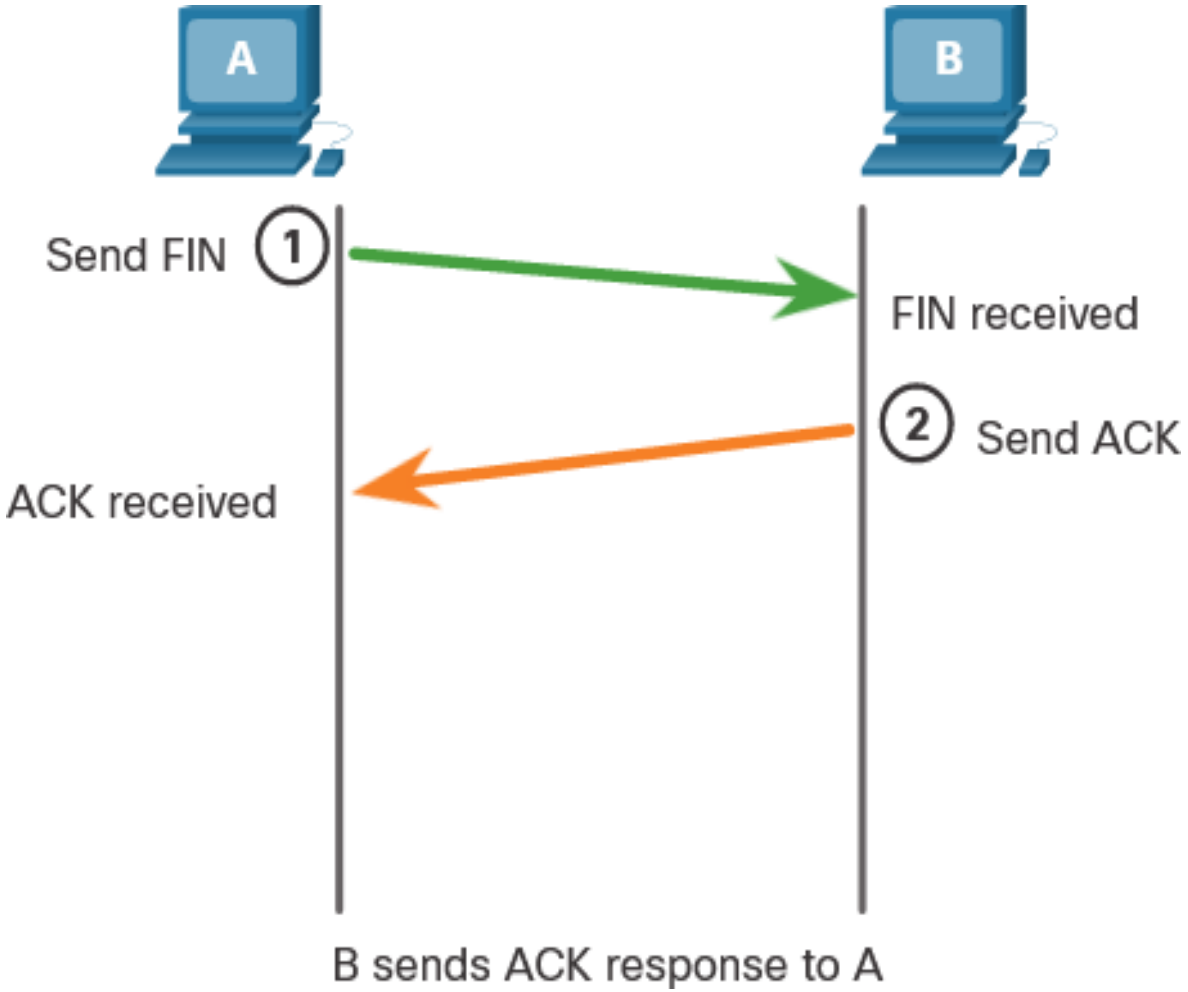



| Seqence Number | | | | | klient sn.200 |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | |
| UGR | **ACK** | PSH | RST | **SYN** | **FIN** |
| | | | | **0** | **1** |

A példában a kliens kezdeményezi a kapcsolat lezárását de a valóság bárki kezdeményezheti!

# 2. lépés: ACK (server)

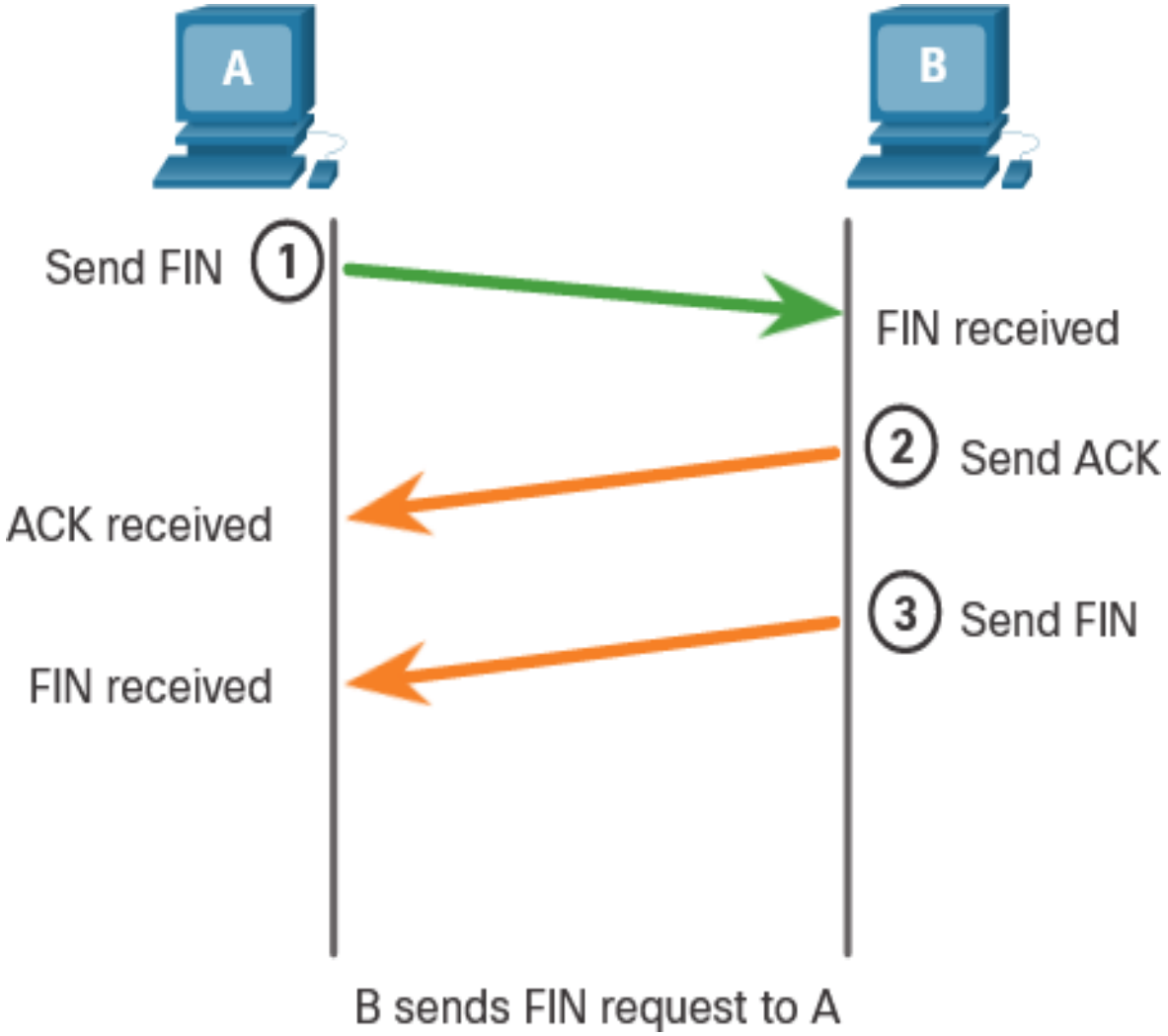
2. A kliens-szerver irányú munkamenet bontásához a szerver egy ACK üzenet küldésével nyugtázza a FIN üzenet megérkezését.




| Seqence Number | | | | | |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | klient sn (+1) .201 |
| UGR | ACK | PSH | RST | SYN | FIN |
| | 1 | | | 0 | 0 |

# 3. lépés: FIN (server)

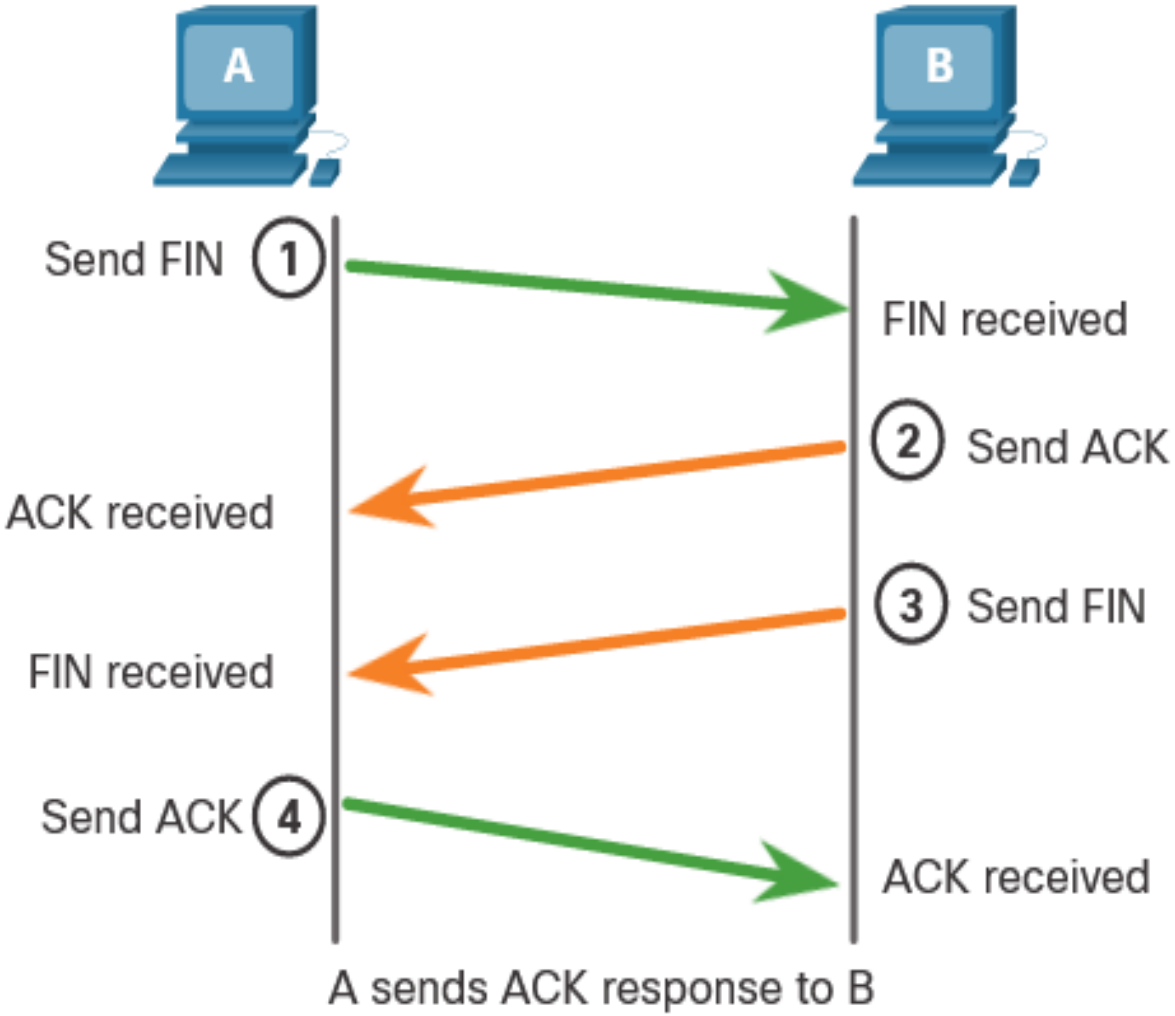
3. A szerver-kliens irányú munkamenet bontásához a szerver egy FIN üzenetet küld a kliensnek.




| Seqence Number | | | | | Ser sn. 500 |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | |
| UGR | **ACK** | PSH | RST | **SYN** | **FIN** |
| | **0** | | | **0** | **1** |

# 4. lépés: ACK (kliens)

A kliens válaszként egy ACK üzenet küldésével nyugtázza a szervertől érkező FIN üzenetet.




| Seqence Number | | | | | |
|---|---|---|---|---|---|
| Ackknowledgement Number | | | | | Ser sn.(+1) 501 |
| UGR | ACK | PSH | RST | SYN | FIN |
| | 1 | | | 0 | 0 |

# Alkalmazások és jellemző protokolljuk

| Alkalmazás | Protokoll neve | Jellemző szállítási rétegbeli protokoll |
|---|---|---|
| **E-mail** | SMTP | TCP |
| **Távoli szerver elérés** | Telnet | TCP |
| **Web** | http | TCP |
| **Fájlátvitel** | FTP | TCP |
| **Hálózati könyvtár megosztás** | NFS | Tipikusan UDP |
| **Multimédia stream** | általában saját fejlesztés | UDP vagy TCP |
| **Internet telefon** | általában saját fejlesztés | UDP vagy TCP |
| **Hálózat adminisztráció** | SNMP | Tipikusan UDP |
| **Forgalomirányítás** | RIP | Tipikusan UDP |
| **Domain név fordítás** | DNS | Tipikusan UDP |

# 4. A megbízható átvitel biztosítása

Sorszámozás és nyugtázás

Nem számít, hogy mennyire jól megtervezett a hálózat, adatvesztés időnként előfordul.
A TCP ezen szegmensveszteségek kezelésére is biztosít módszereket. **Ezek közé tartozik a nem nyugtázott adatszegmensek újra küldésének mechanizmusa is.**
A TCP protokoll úgy biztosítja az összes csomag megérkezését, hogy **visszacsatolást (nyugtázást) küld a megérkezett szegmensekről.**

- A visszacsatolást a TCP csomag fejlécének Sequence number és Acknowledgement number mezőivel valósítja meg.

Opcióként: SACK: SelectiveACK (RFC 2018)

- SYN szegmensben engedélyezhető („SACK Permitted” opció) így több szegmesenként elég nyugtázni

# A hibamentes adatátvitel biztosítása

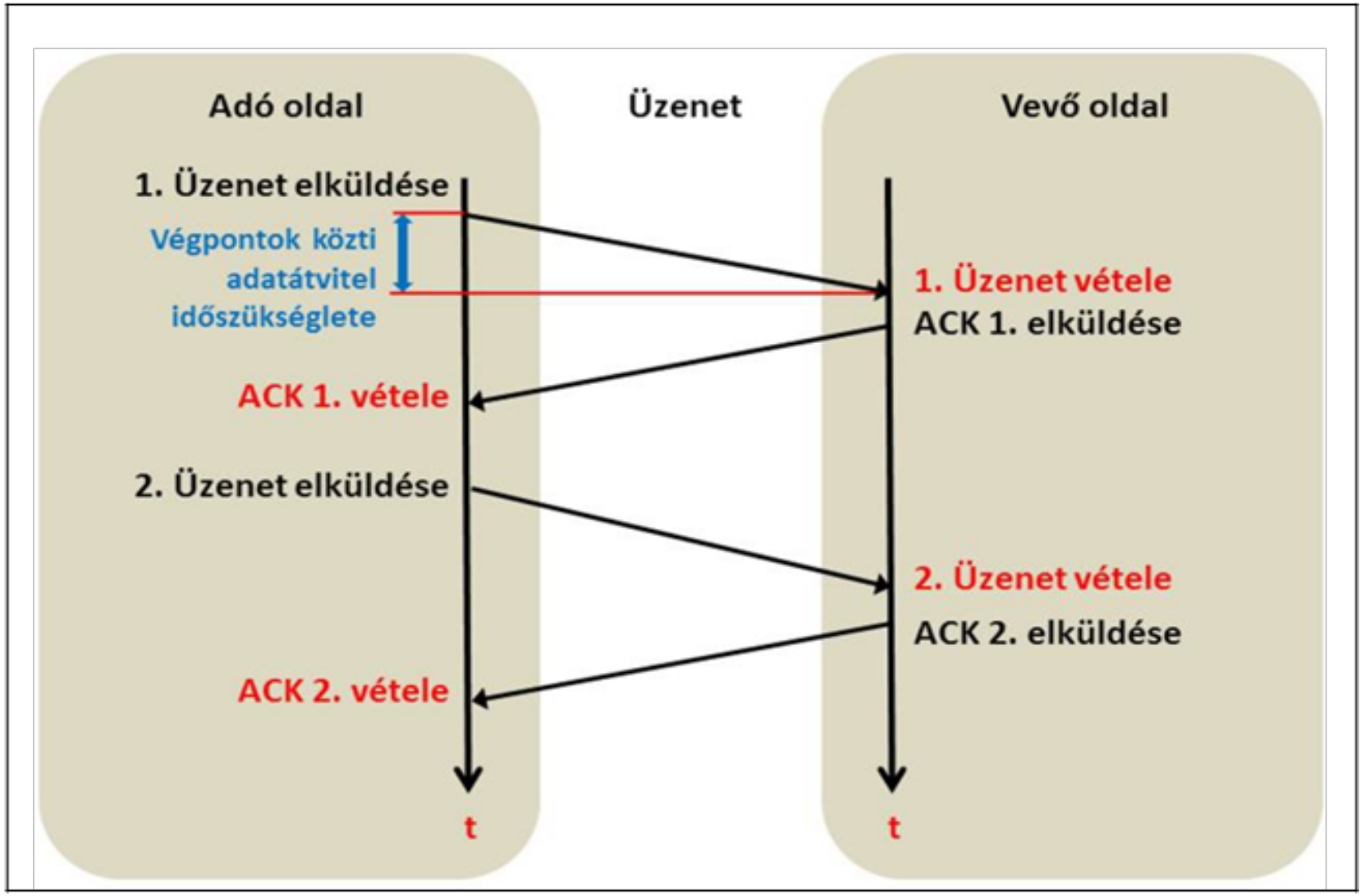
A megoldás: pozitív (kumulatív) nyugtázás, ismételt (visszalépéses) átvitellel [A működés elvét az alábbi ábrák szemléltetik:

**Pozitív nyugtázás újraküldéssel**

A vételi oldalon működő gép nyugtát (Acknowledgement, röviden ACK) küld a feladónak, ha adat érkezik. Az adó minden átküldött üzenetet nyilvántart, és vár a nyugtára.

**Megjegyzés:**

**A két függőleges időtengely a végpontokban eltelő idő múlását jelképezi. A csomag kiküldését szemléltető nyilak ferde volta az adatátvitel időszükségletére utal.**



# Újraküldés

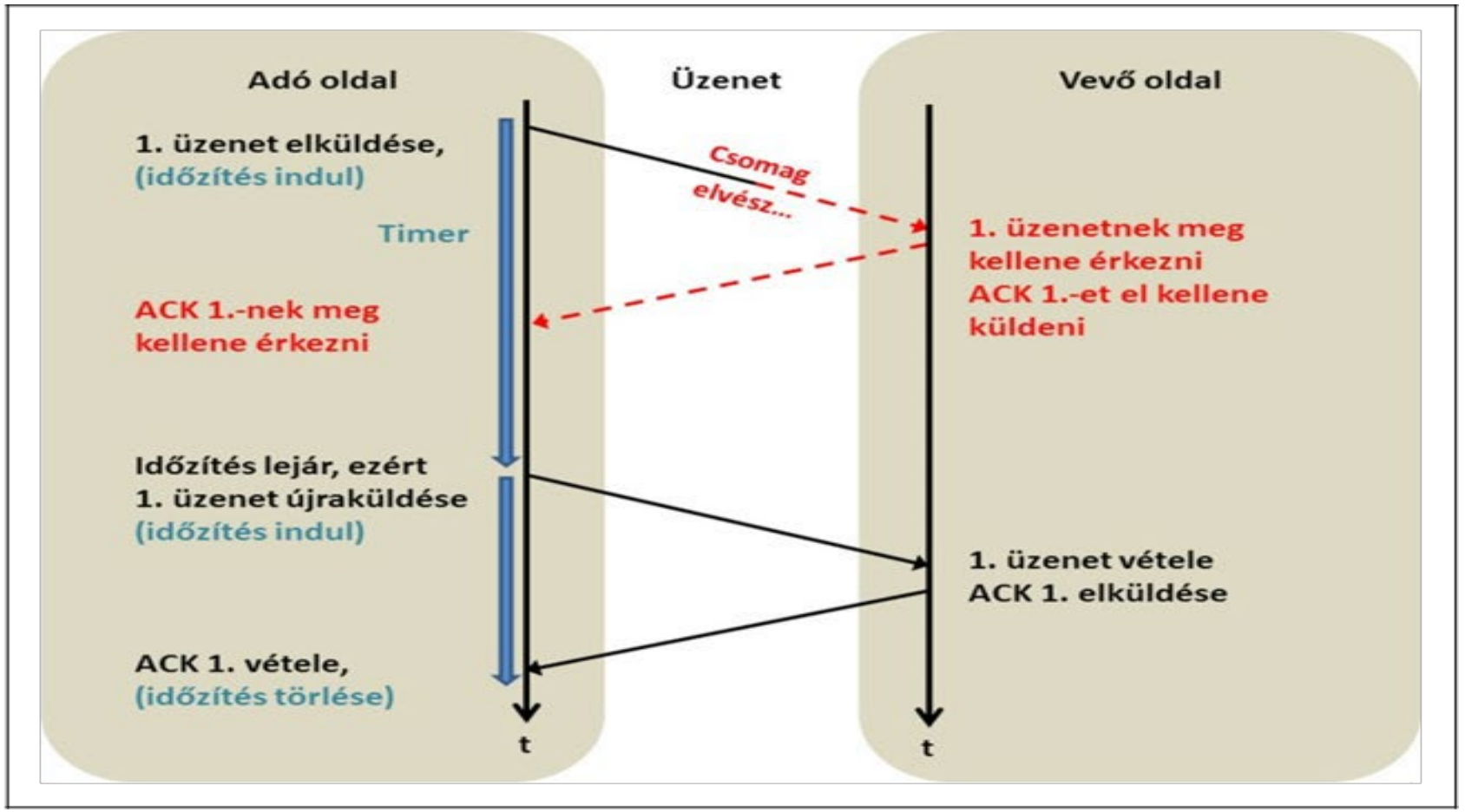
Minden egyes csomag elküldésekor a TCP egy újabb időzítőt indít el, majd ha ennek lejártáig nem érkezik meg a csomag vételét nyugtázó üzenet, újraküldi az adatot.
Az adóként működő protokoll-gép minden üzenetet sorszámmal lát el, a vevőnek pedig emlékeznie kell, mely sorszámú üzenetek érkeztek már meg. A nyugtában a vevő gép visszaküldi a kapott adat sorszámát az adónak, így az a nyugtákat és az elküldött üzeneteket egymáshoz rendelheti.

**Újraküldés, ha a csomag elvész**

---

**Ha az időzítéshez viszonyítva túl nagy a hálózati késleltetés, az üzenetek – adat és nyugtája egyaránt – a Timer lejárta miatt hibátlan átvitel esetén is megkettőződhetnek, a ténylegesen felesleges újraküldés miatt.**

**A nyugták nem a hibát, hanem az adatok hibátlan vételét jelzik. Ezt pozitív nyugtázásnak nevezzük.**



# A megbízható átvitel biztosítása 2.

Sorszámozás

- A irányonként  bájtokat sorszámozzuk

A kezdőértékeket a kapcsolatfelvétel már beállította általában ez véletlenszerű érték.

# 5. Forgalomszabályozás

- A *forgalomszabályozás* (flow control) célja annak az elkerülése, hogy egy gyorsabb állomás egy lassabbat forgalommal elárasszon úgy, hogy a lassabb nem képes azt feldolgozni.

- A problémára a nyugta megvárása nem jó megoldás.
  - Túl sok idő veszteség – lassú átvitel
  - A nyugtát tehát nem szabad megvárni!
  - **A csúszó ablakos technikával több üzenetet is elküldhetünk, mielőtt a nyugtára várnánk. Így az adó folyamatosan dolgozhat.**

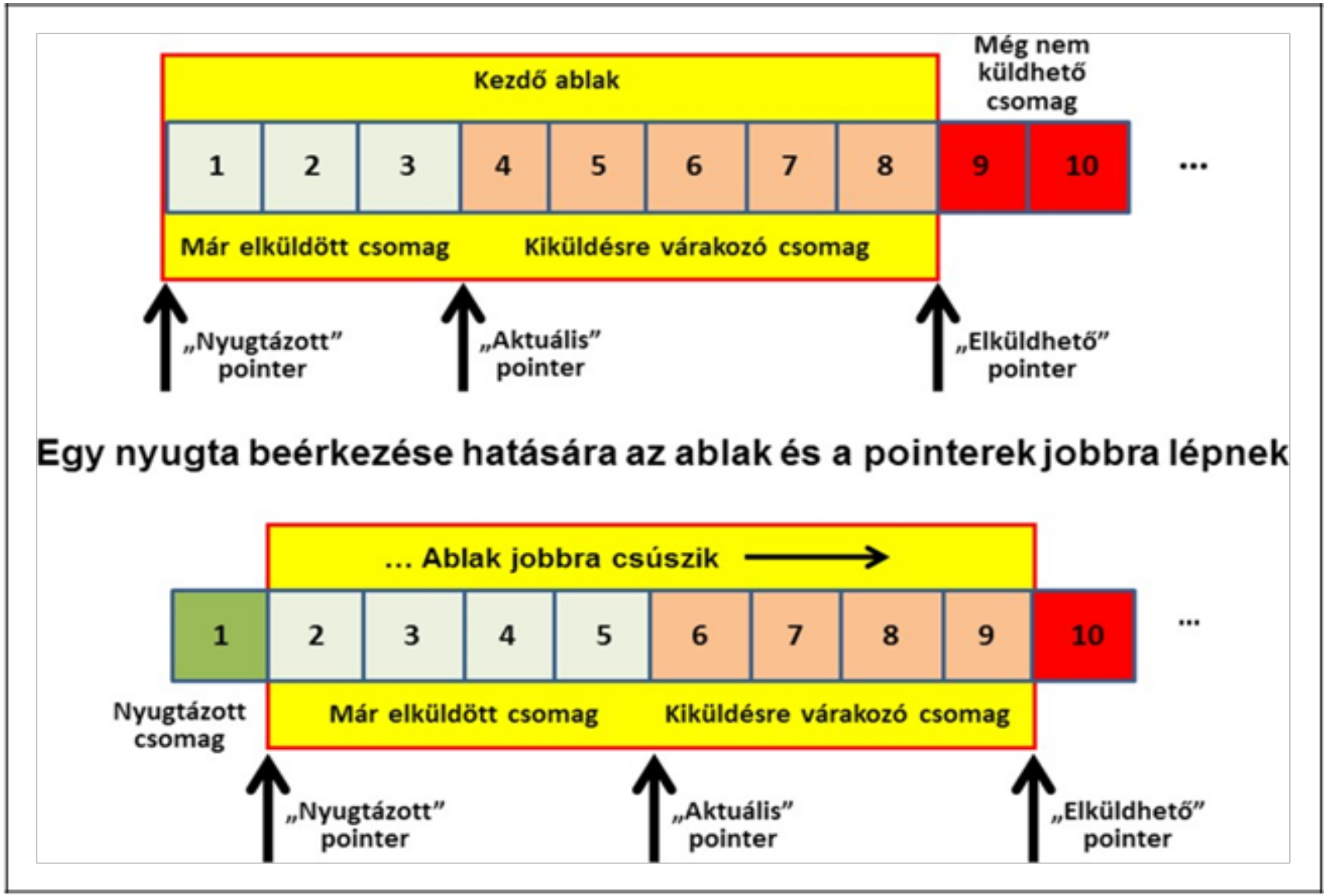
# Forgalomszabályozás

- A TCP forgalomszabályozásra a ***csúszó ablak* (sliding window)** módszert használja.
  - Az ablak értéke egy hitelkeretnek tekinthető, azt fejezi ki, hogy egy állomás ennyi adat elküldését engedélyezi a másik fél számára úgy, hogy a másik fél még nem kapott nyugtát róla.

  - A kapcsolat felépülésekor az állomások közlik a másik féllel a kezdő ablakméretüket is
  - A kommunikáció során az állomások az ablakméretet megváltoztathatják vagy 0-ra állíthatják, utóbbi ez azt jelzi hogy a küldést fel kell függeszteni
  - Az ablak méretének helyes megválasztása nagyban befolyásolja a protokoll hatékonyságát.

# Csúszó ablak (Sliding Window)

**Csúszó ablakos protokoll működése példaként 8 csomag méretű ablakkal**




---

**Megjegyzés:**

**Az ábrán látható pointerek követik az adatátvitel folyamatát, magyarázatuk a szövegben olvasható.**

**A két szélső pointer közti távolság az ablak méretét jelöli ki, a példában ez az érték 8.**

# Csúszó ablak (Sliding Window)

**Az adó működését három pointer segíti, ezeket „Nyugtázott”, „Aktuális” és „Elküldhető” pointernek nevezzük.**

- A Nyugtázott pointertől balra álló adatok átvitele a vevőhöz már sikeresen lezajlott, erről a beérkező nyugta tanúskodott. Mivel ezek az adatok már a vevő puffereiben vannak, megőrzésük a vevő felelőssége, az adó oldalon tárolni azokat már nem szükséges.
- A Nyugtázott és az Aktuális pointer között található adatcsomagok kiküldése már megtörtént, de az átvitel sikeréről tudósító nyugták még nem érkeztek meg az adóhoz.
- Az Aktuális pointer arra az adatcsomagra mutat, amelynek hálózatra küldése éppen most zajlik.
- Az Aktuális és Elküldhető pointerek között található adatcsomagok azok, amelyek az ablak adta határok között még a vonalra küldhetők anélkül, hogy újabb nyugta érkezne.
- Az Elküldhető pointertől jobbra található adatcsomagok kívül esnek az ablak határain, ezért adásuk nem kezdődhet meg, még akkor sem, ha az adó egyébként szabad lenne. Ezek a csomagok majd akkor küldhetők el, ha – egy nyugta beérkezésének hatására – az ablak jobbra mozdul..

# 6. Az adatcsomagok szegmentálása és ismételt összeállítása

- A szállítási réteg feladata, hogy az alkalmazások adatait megfelelő méretű blokkokra ossza fel.
- A szállítási réteg az adatokat kisebb blokkokra (azaz szegmensekre vagy datagramokra) osztja, amelyek könnyebben kezelhetők és szállíthatók.
- Minden szegmens egy fix kiosztású 20 bájtos fejrésszel kezdődik, amit fejrészopciók követhetnek. Ezek után – ha van –legfeljebb 65495 bájt adat állhat. Az adatot nem tartalmazó szegmensek érvényesek, általában nyugtázásra és vezérlőszegmensként használják azokat.

**Szokás szerint, a TCP szegmens Fejlécet (Header) + Adatmezőt tartalmaz**

# 7. Torlódásvezérlés

- **A *torlódásvezéslés*** (congestion control) célja annak az elkerülése, hogy valamely közbenső hálózati eszköz (router, link) túlterhelése miatt a hálózat teljesítőképessége radikálisan csökkenjen (congestive collapse).
- Miért is fordulhatna elő ilyen állapot?
  - Ha a hálózat terhelése túl nagy (megközelíti a kapacitását), akkor a csomagvesztés megnő, és a TCP elkezdi újra küldeni a nyugtázatlan szegmenseket.
  - Az újraküldés okozta terhelés növekedés további csomagvesztéseket okoz, és az önmagát gerjesztő folyamat odáig jut, hogy a hálózat kapacitásának csak a töredékét kitevő átvitelre képes, ráadásul rendkívül rossz minőségi jellemzők mellett.

# Torlódásvezérlés a TCP-ben (elvek)

- Módszer: ha úgy érzékeljük, van elég átbocsátóképesség, akkor növeljük az adatsebességet, amíg torlódásra utaló jeleket nem tapasztalunk

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