

Számítógépes Hálózatok 2007

5. Adatkapcsolati réteg – CRC, utólagos hibajavítás, csúszó ablakok

Hibafelismerés: CRC

- Hatékony hibafelismerés: Cyclic Redundancy Check (CRC)
- A gyakorlatban gyakran használt kód
 - Nagy hibafelismerési ráta
 - Hardware megvalósítás egyszerű
- Polinóm aritmetikán alapul a 2-es maradékosztályok (Z_2) testén
 - A jelsorozatokat polinómnak tekintjük
 - A bitek a polinóm együtthatói

Számolás Z_2 -ben

- Számolás modulo 2:

- Szabályok:

- összeadás mod 2

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

- kivonás mod 2

A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

- szorzás mod 2

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

- Példa: 0110111011
 + 1101010110
 = 1011101101

Polinóm aritmetika modulo 2

- Tekintsük a polinómokat Z_2 maradékosztály test fölött
 - $p(x) = a_n \cdot x^n + \dots + a_1 x^1 + a_0$
 - Az a_i együtthatók és az x változók $\in \{0,1\}$
 - A számítás modulo 2 történik
- Polinómok összeadása, kivonása, szorzása, (maradékos) osztása, mint ahogy ismerjük

Bit sztringek és Z_2 feletti polinómok

- Ötlet:
 - Tekintsük az n hosszúságú bit sztringet mint egy polinóm együtthatóit
- Bit sztring: $b_n b_{n-1} \dots b_1 b_0$
Polinóm: $b_n \cdot x^n + \dots + b_1 \cdot x^1 + b_0$
 - Egy $(n+1)$ bitből álló bit sztring megfelel egy n -ed fokú polinómnak
- Példa
 - $A \text{ xor } B = A(x) + B(x)$
 - Ha A -t k pozícióval balra toljuk (shift), ez a következőnek felel meg:
 - $C(x) = A(x) x^k$
- Ezt az izomorfizmust használva tudunk bit sztringekkel osztani

Maradékos osztás bitsztingekkel

- Példa:
- $1101010101 : 1001 = 1100110$ maradék 11

1001

1000

1001

001101

1001

1000

1001

0011

Redundancia polinómak által: CRC

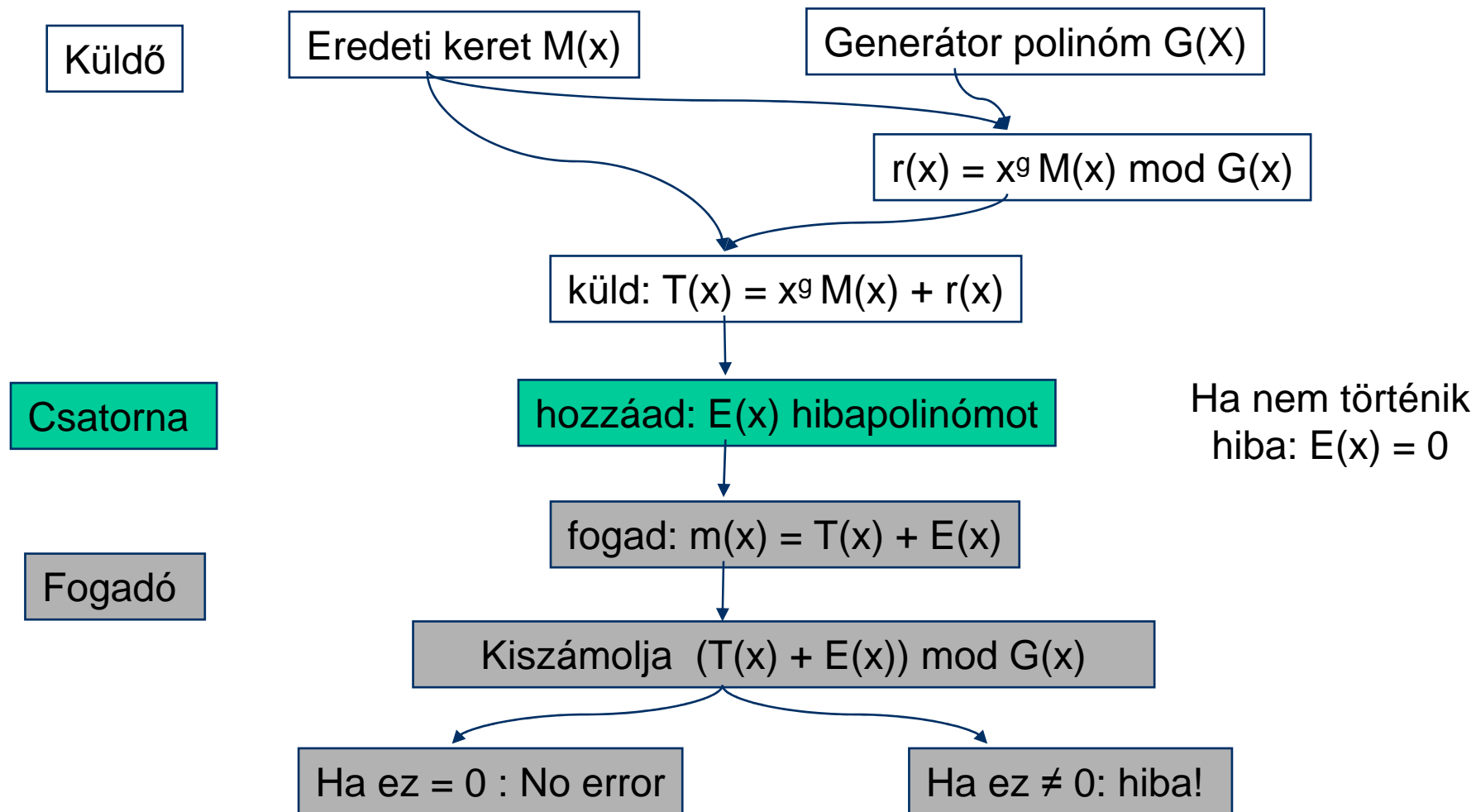
- Definiáljunk egy $G(x)$ generatorpolinómat, melynek a foka g
 - $G(x)$ a küldő és a fogadó által ismert
 - g redundáns bitet generálunk
- Adott:
 - Keret (frame, üzenet) M , mint $M(x)$ polinom
- Küldő
 - Kiszámolja az osztás maradékát $r(x) = x^g M(x) \bmod G(x)$
 - Átvitelre kerül: $T(x) = x^g M(x) + r(x)$
 - Figyeljük meg: $x^g M(x) + r(x)$ többszöröse $G(x)$ -nek
- Fogadó
 - $m(x)$ -et fogad
 - Kiszámítja a maradékot: $m(x) \bmod G(x)$

CRC Átvitel

- Ha nem történt hiba:
 - $T(x)$ fogadása korrekt
- Bithiba: $T(x)$ tartalmaz megváltoztatott bitet
 - Ez ekvivalens egy $E(x)$ hibapolinóm hozzáadásához
 - A fogadóhoz $m(x) = T(x) + E(x)$ érkezik
- Fogadó
 - Kiszámítja $m(x) \bmod G(x)$ maradékot
 - Ha nincs hiba: $m(x) = T(x)$,
 - Ekkor a maradék 0

- Bit hibák: $m(x) \bmod G(x) = (T(x) + E(x)) \bmod G(x)$
 $= \underbrace{T(x) \bmod G(x)}_0 + \underbrace{E(x) \bmod G(x)}_{\text{hibaindikátor}}$

CRC – Áttekintés



A generator meghatározza a CRC tulajdonságait

- A bit-hibákat csak akkor nem ismerjük fel, ha $E(x)$ többszöröse $G(x)$ -nek
- $G(x)$ választásának trükkjei:
 - 1-bit-hiba: $E(x) = x^i$ hiba az i -edik pozíción
 - Ha $G(x)$ legalább 2 nem nulla együthatót tartalmaz, akkor $E(x)$ nem többszörös
 - 2-bit-hiba: $E(x) = x^i + x^j = x^j (x^{i-j} + 1)$, ahol $i > j$
 - $G(x)$ nem szabad, hogy osztója legyen $(x^h + 1)$ -nek semmilyen h -ra, $0 \leq h \leq k$, a maximális kerethosszig,
 - Páratlan számú hiba:
 - Ekkor $E(x)$ nem többszöröse $(x+1)$ -nek
 - Ötlet: legyen $(x+1)$ osztója $G(x)$ -nek
 - ekkor $E(x)$ nem többszöröse $G(x)$ -nek
- $G(x)$ okos megválasztásával minden r hosszú hibasorozat (burst) felismerhető

CRC a gyakorlatban

- Az IEEE 802.3 (Ethernet) standardban felhasznált generátor polinóm (CRC-32):

- $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

- Figyelem:

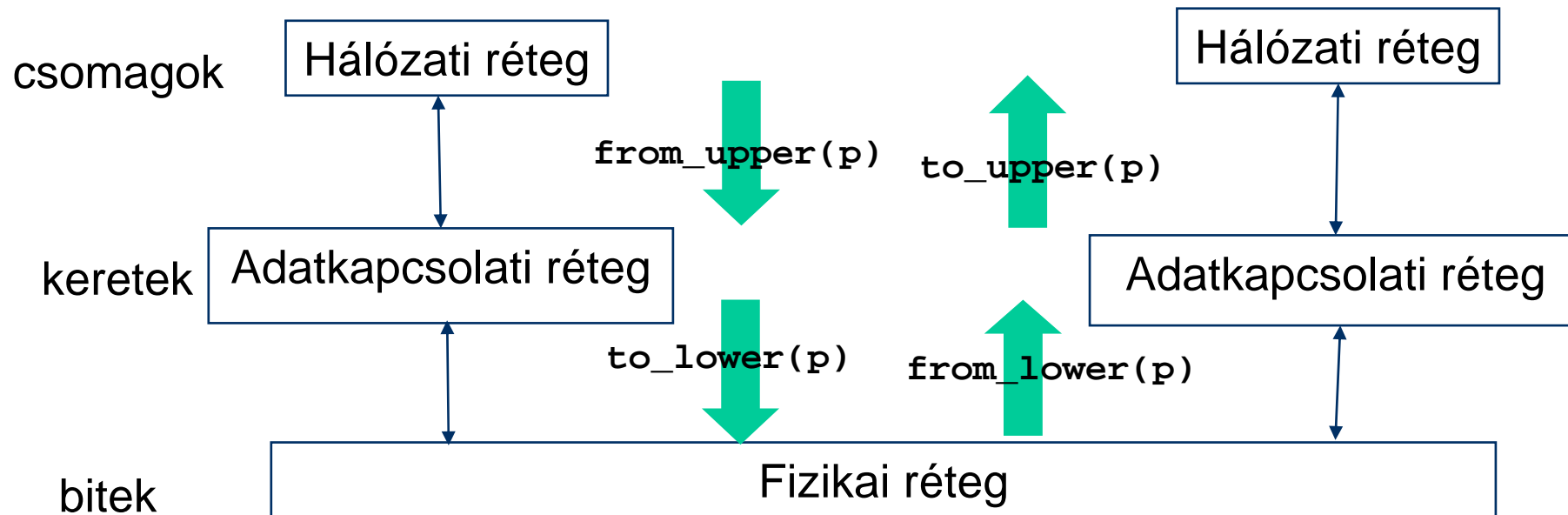
- Hiba még mindig lehetséges
 - Különösen, ha a bithibáknak megfelelő $E(x)$ többszöröse $G(x)$ -nek.

- Implementáció:

- Egyszerű XOR-operáció
 - HW implementáció: shift-register

Utólagos hibajavítás

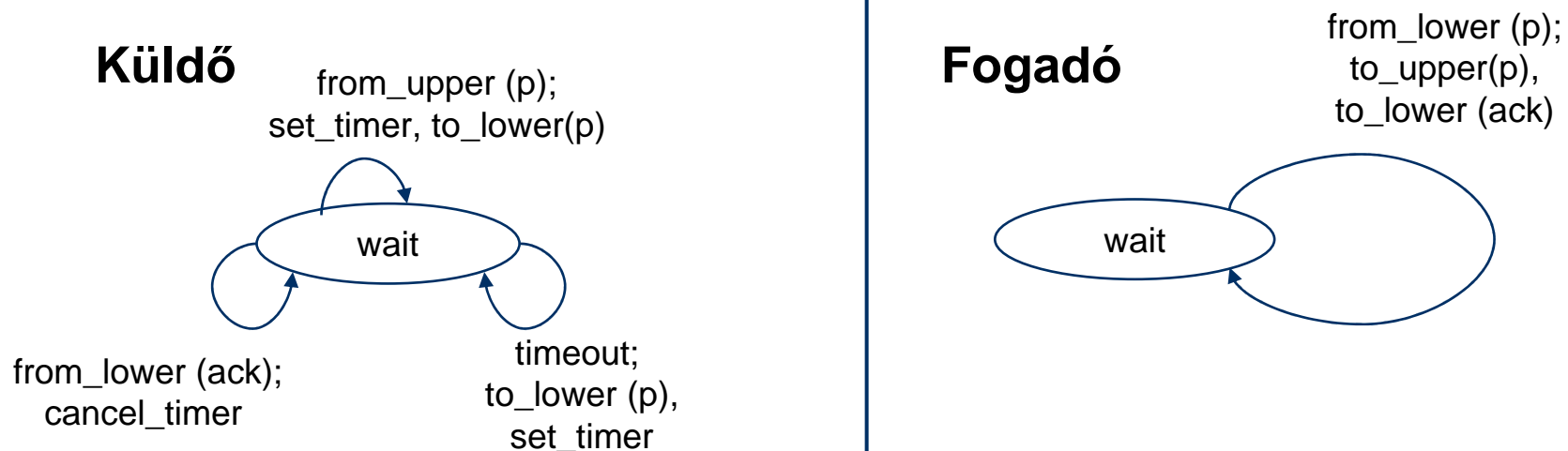
- A hiba felismerésekor a keretet újra kell küldeni
- Hogy néz ki a küldő és a fogadó összehangolt munkája?



`to_lower`, `from_lower` tartalmazzák a CRC-t
vagy (szükség esetén) utólagos hibajavítást

Egyszerű simplex protokoll nyugtákkal

- Simplex üzemmód: csomagok küldése csak egyirányú
- A fogadó nyugtázza a küldő csomagjait (ehhez fél-duplex fizikai csatorna elegendő)
 - A küldő vár egy bizonyos ideig a nyugtára (acknowledgment -- ACK)
 - Ha az idő lejárt, újraküldi a csomagot
- Első megoldási kísérlet:



Elemzés

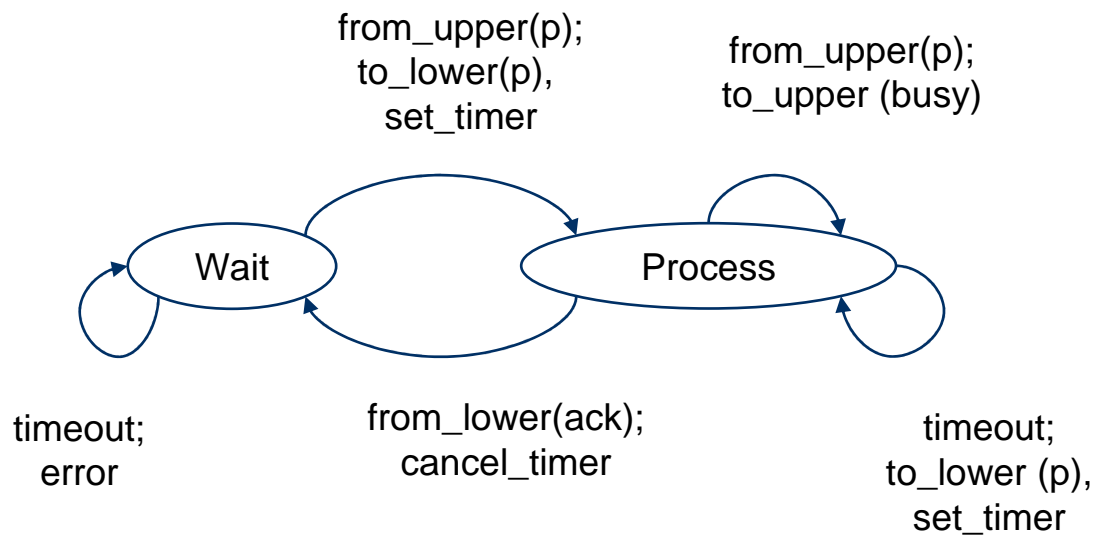
- Problémák
 - A felső réteg gyorsabban küldi a csomagokat, mint ahogy a nyugták megérkeznek

- Mi történik, ha nyugták elvesznek

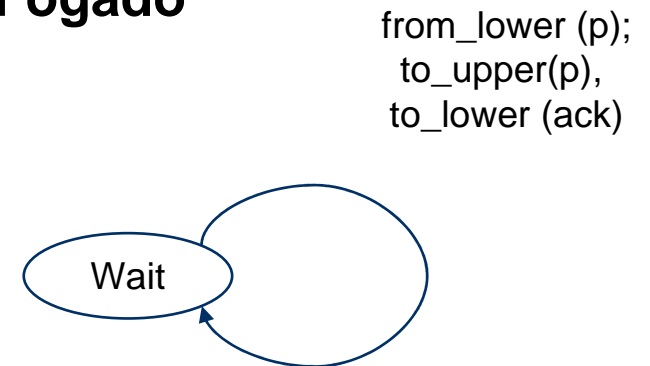
2. Kisérlet

- Az első probléma megoldása
 - Egy csomag a másik után

Küldő

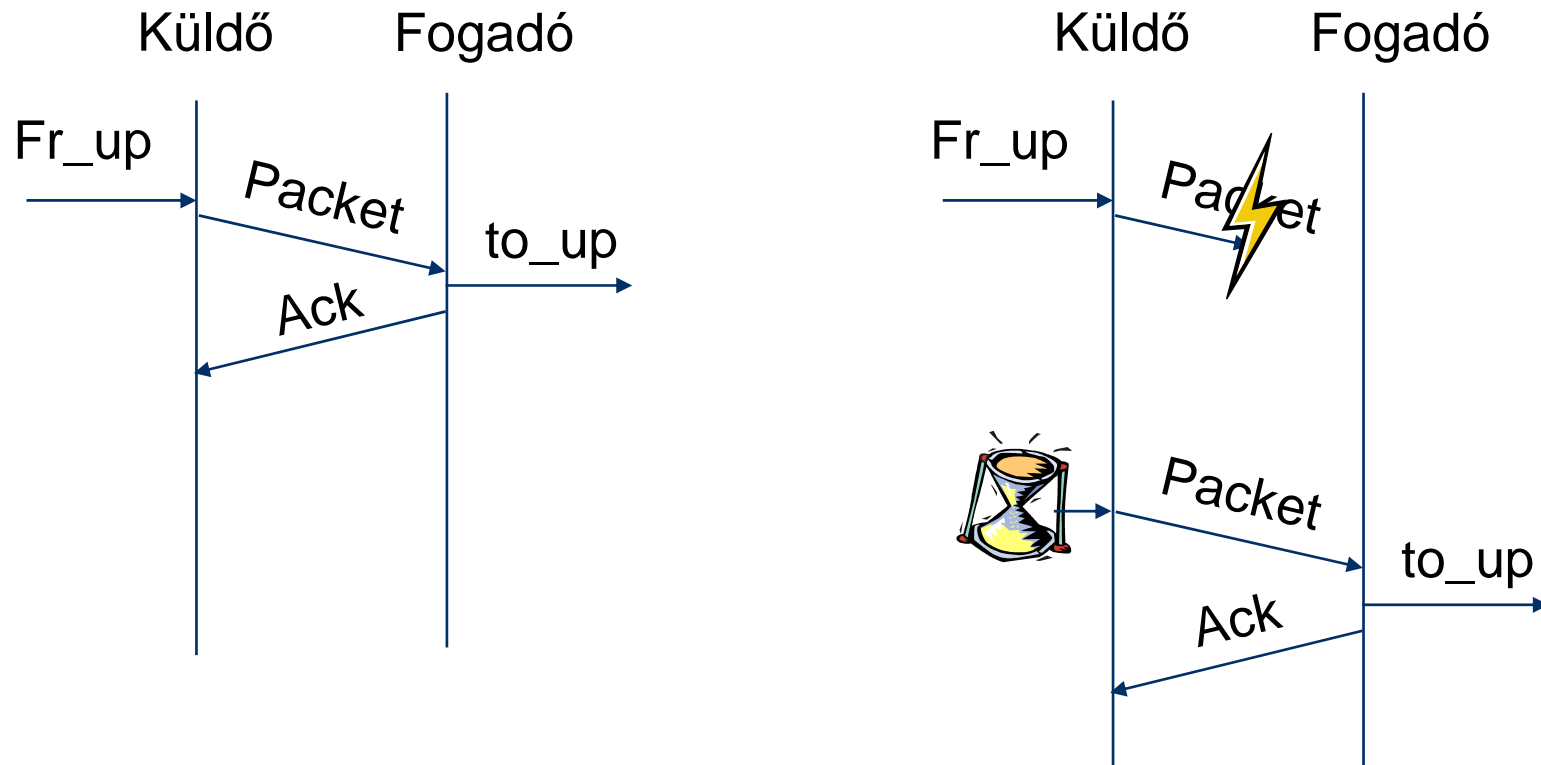


Fogadó



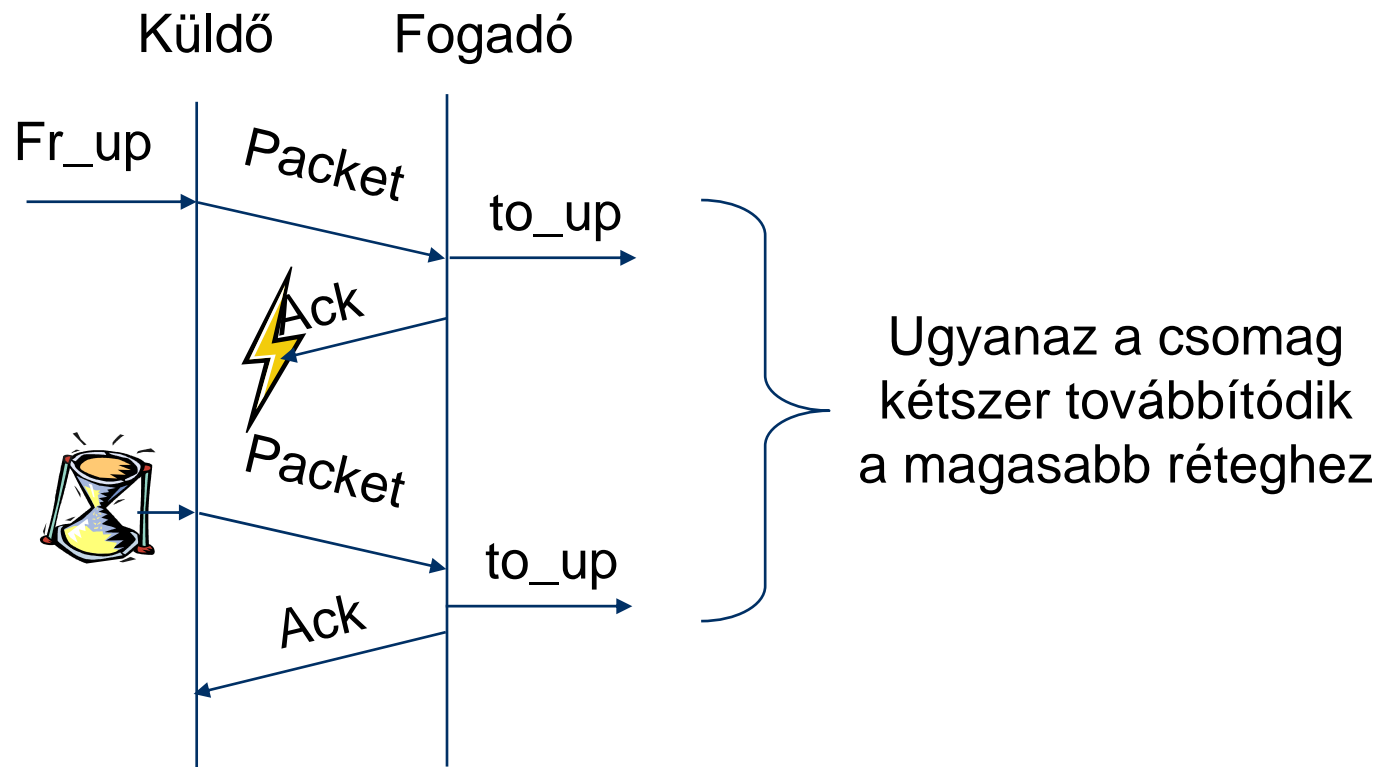
Elemzés

- A protokoll megvalósít egy elemi folyamfelügyeletet



Elemzés

- 2. probléma: elveszik a nyugta

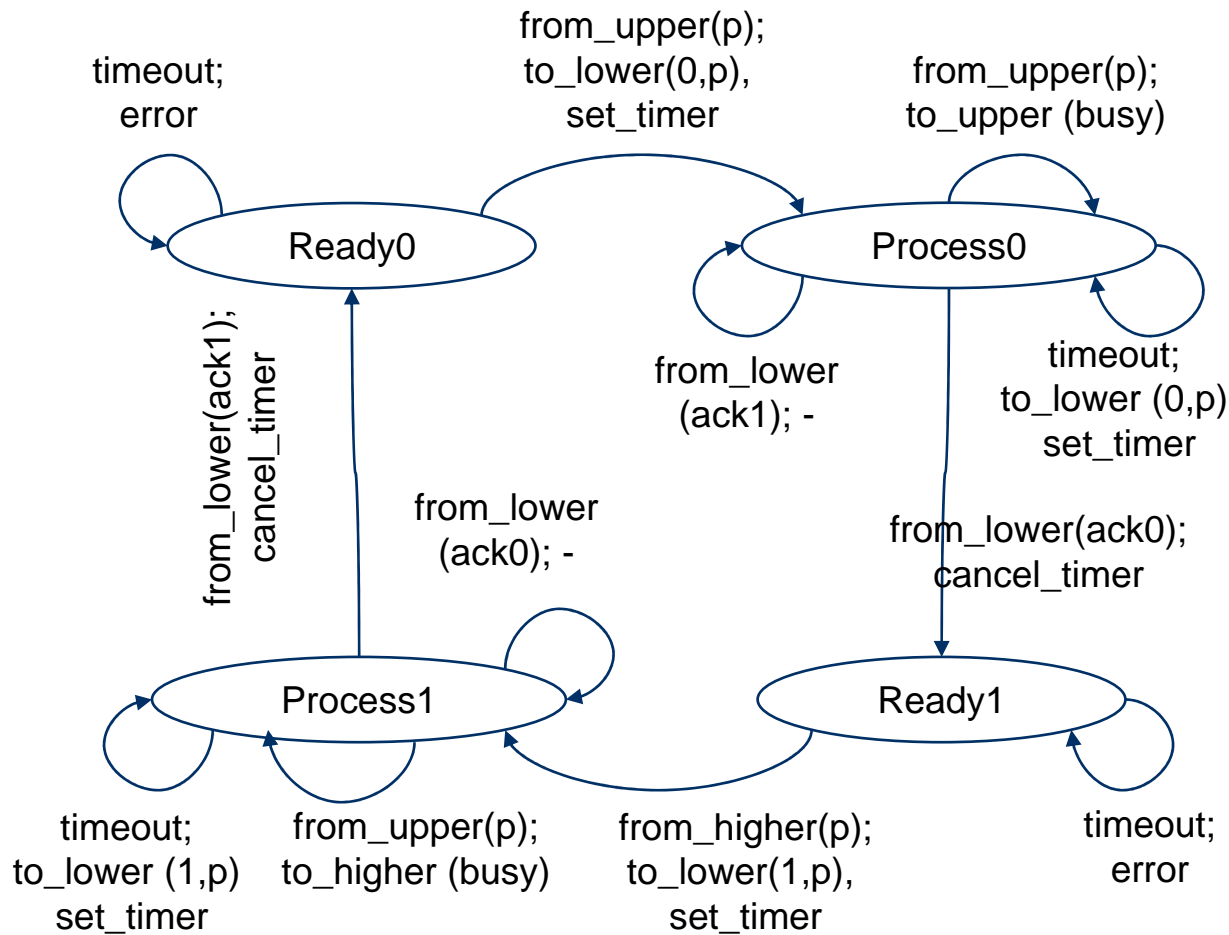


A 2. probléma (duplikátumok)

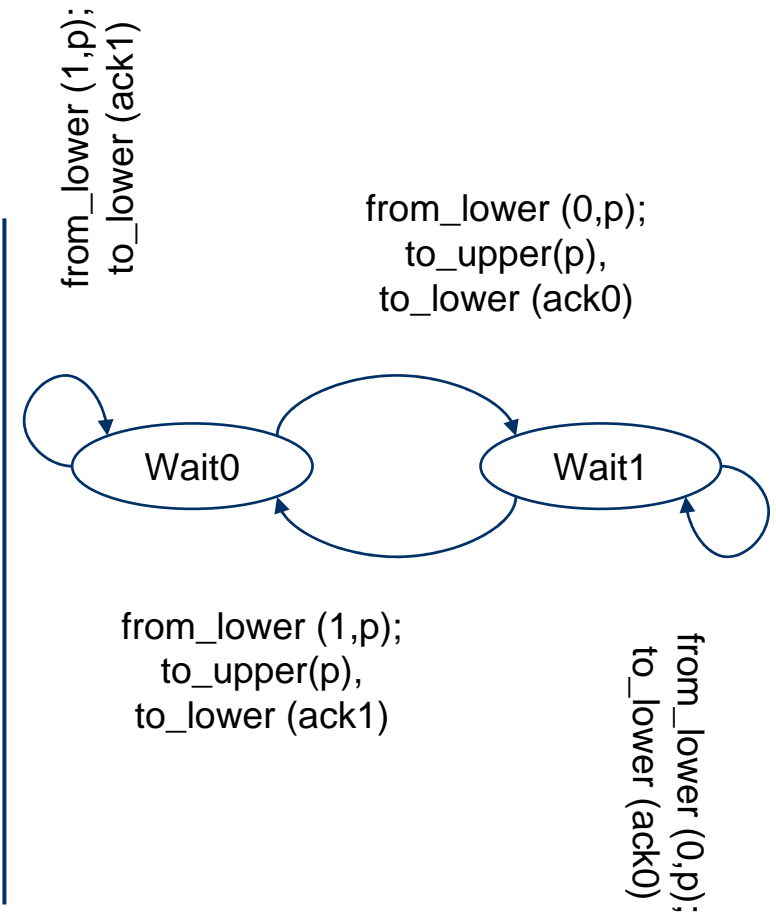
- A küldő nem tud különbséget tenni elveszett csomag és elveszett nyugta között
 - Újra kell küldeni a csomagot
- A fogadó nem tud különbséget tenni egy csomag és egy régi csomag redundáns másolata között
 - További információ szükséges
- Ötlet:
 - Minden csomagot ellátunk egy **sorszámmal** (**sequence number**), hogy a fogadónál az azonosítás lehetséges legyen
 - Minden csomag fejléce tartalmaz sorszámot
 - Itt: csak 0 vagy 1
- Szükséges a csomagban és a nyugtában
 - A nyugta az utolsó hibátlanul fogadott csomag sorszámát tartalmazza (tisztán konvenció)

3. kísérlet: nyugta és sorszám

Küldő



Fogadó

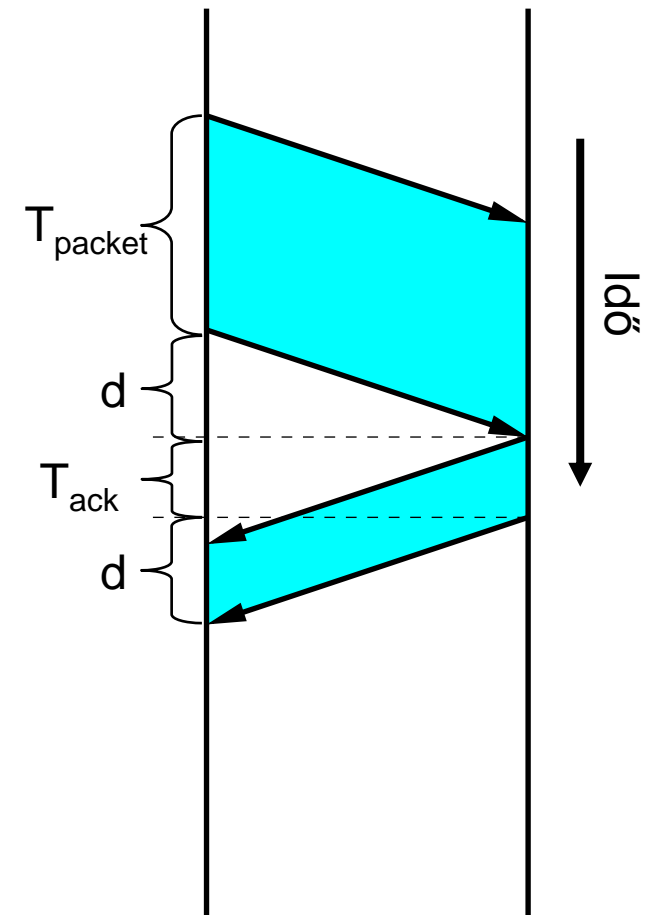


3. kísérlet: alternáló bit protokoll (Alternating Bit Protocol)

- A 3. kísérlet egy zajos csatorna fölötti megbízható protokoll korrekt implementációja
 - Alternating Bit Protokoll
 - Az „Automatic Repeat reQuest (ARQ)” protokollok közé tartozik
 - Folyamfelügyelet egy egyszerű formáját is tartalmazza
- Egy nyugta két feladata
 - nyugtázni, hogy egy csomag megérkezett
 - engedélyezni egy új csomag küldését

Alteráló bit protokoll -- hatékonyság

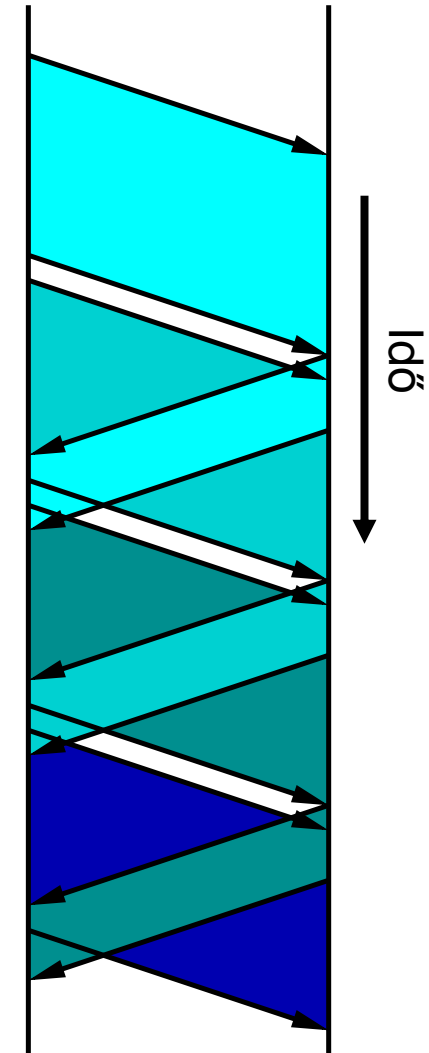
- Hatékonyság η a következő két érték arányaként definiált:
 - az idő, amely a küldéshez szükséges és
 - az idő, amely szükséges, amíg újra lehet küldeni
 - (hibamentes csatornán)
- $\eta = T_{\text{packet}} / (T_{\text{packet}} + d + T_{\text{ack}} + d)$
- Nagy delay esetén az alteráló bit protokoll nem hatékony



A hatékonyság javítása

- A csomagok folyamatos küldése növeli a hatékonyságot
 - több „outstanding” csomag (elküldött, de még nem nyugtázott) növeli a hatékonyságot
 - csomag „pipeline”
- Nem csak 1-bit-sorozatszámmal lehetséges

A küldő folyamatosan küld – nő a hatékonyság

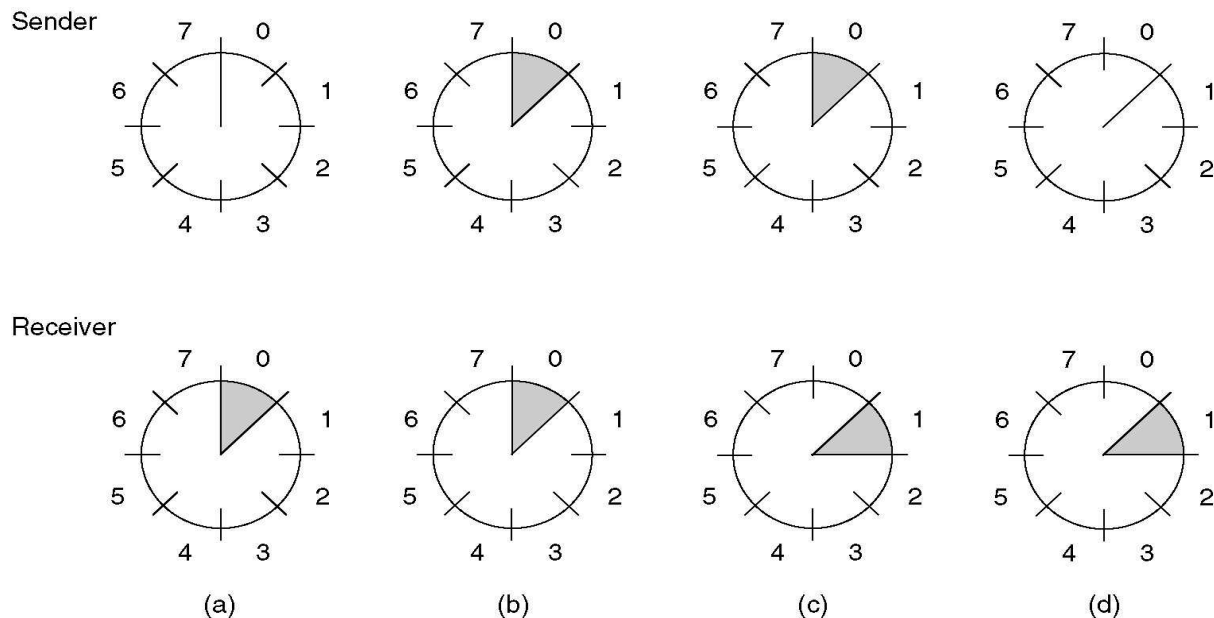


Csúszó ablak (sliding window)

- A sorozatszámok terét megnöveljük n bitre, azaz 2^n sorozatszámra
- Nem mind használható fel ugyanabban az időben
 - az Alternating Bit Protocol-ban sem lehetséges
- “**Csúszó ablakok**” (**sliding windows**) a küldőnél és a fogadónál kezelik ezt a problémát
 - Küldő: küldő-ablak
 - Sorozatszámok olyan sorozata, amelyek egy adott időben elküldhetők
 - Fogadó: fogadó-ablak
 - Sorozatszámok olyan sorozata, melyet a fogadó egy adott időpillanatban hajlandó elfogadni
 - Az ablakok mérete lehet fix vagy időben dinamikusan változtatható
 - Az ablakméret folyamfelügyeletet tesz lehetővé

Példa

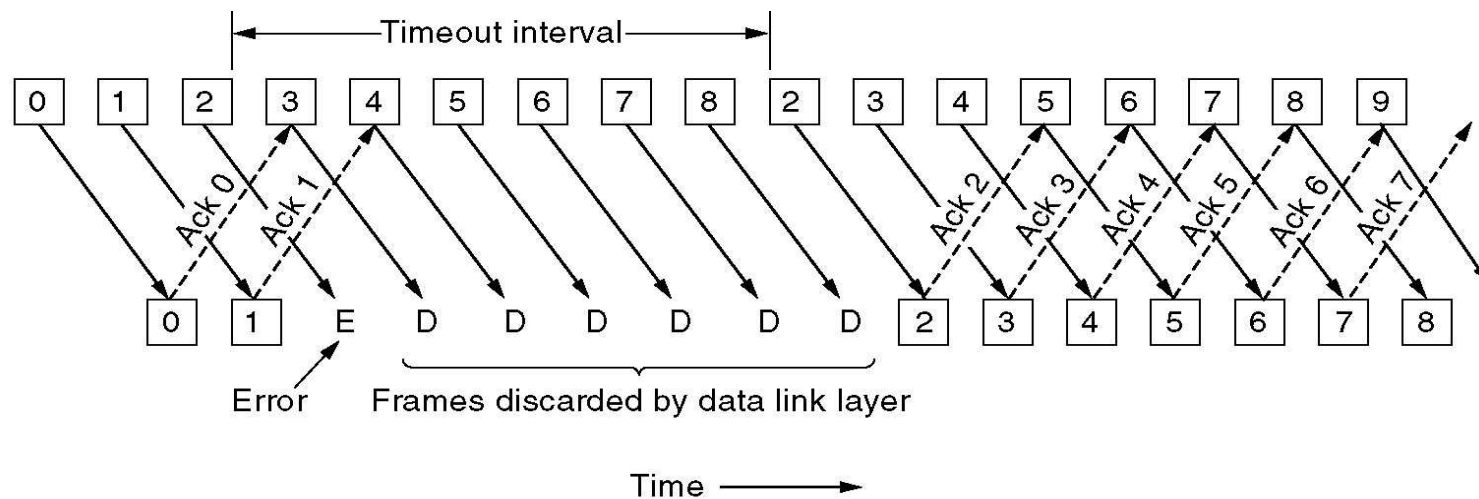
- “Sliding Window” példa $n=3$ és fix ablakméret = 1 esetén
- A küldő itt mutatja a még nem nyugtázott sorozatszámokat
 - Ha a még nem nyugtázott keretek (frame) száma ismert, akkor ez ekvivalens az előző fólián definiált a küldő-ablakkal



- Kezdetben: mielőtt bármit küldenénk
- Az első frame küldése után 0 sorozatszámmal
- Az első frame fogadása után
- Az első nyugta fogadása után

Átviteli hiba és a fogadó-ablak

- Feltételeink:
 - Az adatátviteli rétegnek minden frame-et helyesen és **helyes sorrendben** kell átvinni
 - A küldő hatékonyság növeléséhez pipeline technikát használva küldi a csomagokat
- Csomagvesztés esetén: Ha a fogadó-ablakméret = 1, a következő csomagokat mind eldobja a fogadó

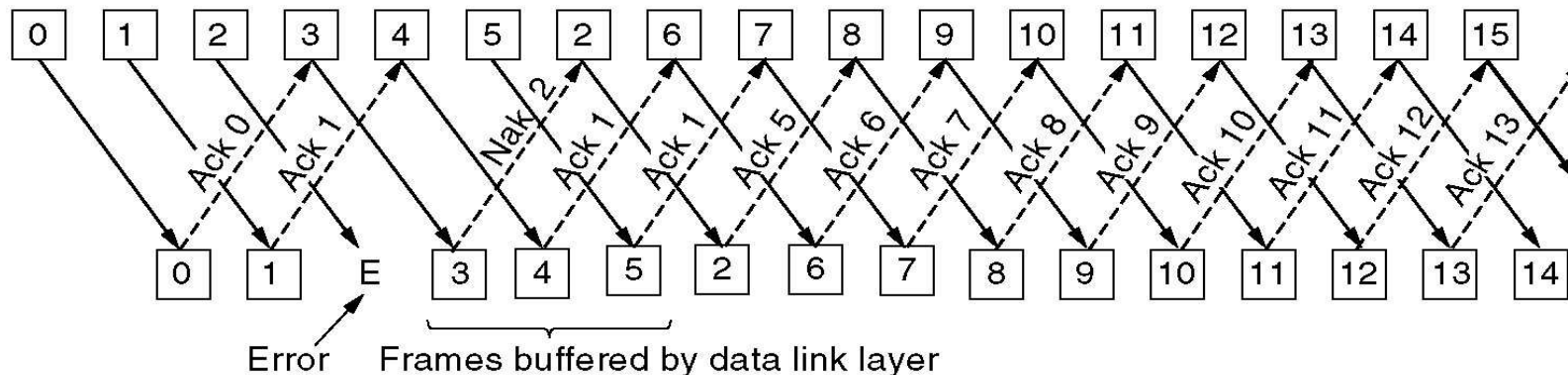


Go-back-N

- Ha a fogadó-ablakméret = 1, akkor a fogadó nem tudja feldolgozni azokat a frame-eket, melyek egy elveszett (vagy hibás) frame-et követnek
 - Nem tudja azokat nyugtázni, mert csak egy nyugtát küld az utolsó helyesen fogadott csomagról
- A küldőnél lejár a várakozási idő a nyugtára: “Timeout”
 - Minden frame-et, amit az utolsó nyugtázott frame után küldött, újra kell küldeni
 - **“Go-back-N”** Frames!
- Kritika
 - Az átviteli médium pazarlása
 - A fogadónál viszont nagyon egyszerű a feldolgozás

Szelektív ismétlés (Selective Repeat)

- Tegyük fel, hogy a fogadó tudja pufferelni a csomagokat, amelyek a közbeneső időben érkeztek
 - azaz a fogadó-ablakméret > 1
- Példa



- A fogadó értesíti a küldőt a hiányzó csomagról negatív nyugtával
- A küldő elküldi a hiányzó frame-eket szelektíven (**selective repeat**)
- Amikor a hiányzó frame megérkezik, minden frame-et (a helyes sorrendben) átad a fogadó a hálózati rétegnek

Duplex-operáció és „hátizsák” technika (piggybacking)

- Simplex
 - Információ küldés egy irányba
- Duplex
 - Információ küldés mindkét irányba
- Eddig:
 - Simplex interfész a magasabb réteghez (hálózati réteghez)
 - (Fél-)Duplex interfész az alacsonyabb réteghez (fizikai réteghez)
- Mi kell akkor, ha az interfész a magasabb réteghez duplex
 - Nyugta és adatcsomagok elkülönítve mindkét irányban
 - Vagy: **hátizsák technika** (általánosan használt)
 - A nyugtát az ellentétes irányba küldött adat-frame fejlécébe tesszük (**piggybacking**)