Számítógép-hálózatok
Az adatkapcsolati réteg

2025/2026. tanév, I. félév

Dr. Kovács Szilveszter

E-mail: szilveszter.kovacs@uni-miskolc.hu

www.iit.uni-miskolc.hu/~szkovacs

Informatika Intézet 107/a.

Tel: (46) 565-111 / 21-07

Tartalom

- Általános megállapítások
- Keretképzés - felismerés
- Hibakezelés, lehet
  - közvetlen, ill. közvetett.
- Adatfolyam vezérlés
- Elemi adatkapcsolati protokollok

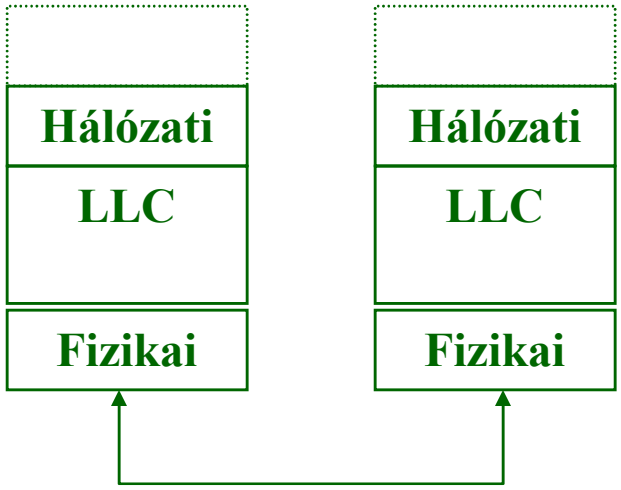
Az adatkapcsolati réteg

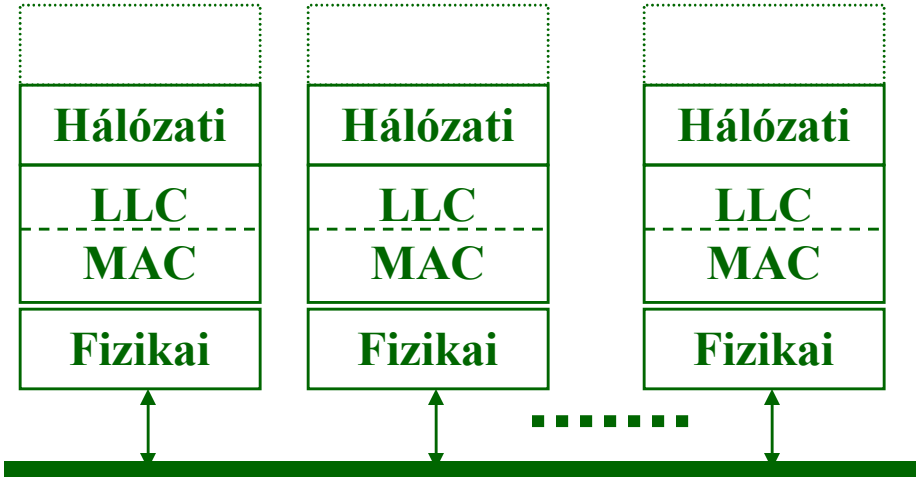
- **A második réteg**
- **Feladata (általánosan)**
  - **Jól meghatározott interfész és szolgálatok biztosítása a hálózati réteg számára**
  - **Két, szomszédos "huzalszerűen"** működő csatornával összekötött állomás között
    megbízható,
    hatékony
    kommunikáció biztosítása.

“Huzalszerű” csatorna

- **A bitek pontosan ugyanolyan sorrendben érkeznek, mint ahogy feladták őket.**
- **Két állomás között “huzalszerű” csatornát**
  - **pont-pont** kapcsolat esetén a **fizikai réteg,**
  - **üzenetszórásos** csatorna esetén a **fizikai + közeghozzáférési alréteg biztosít.**

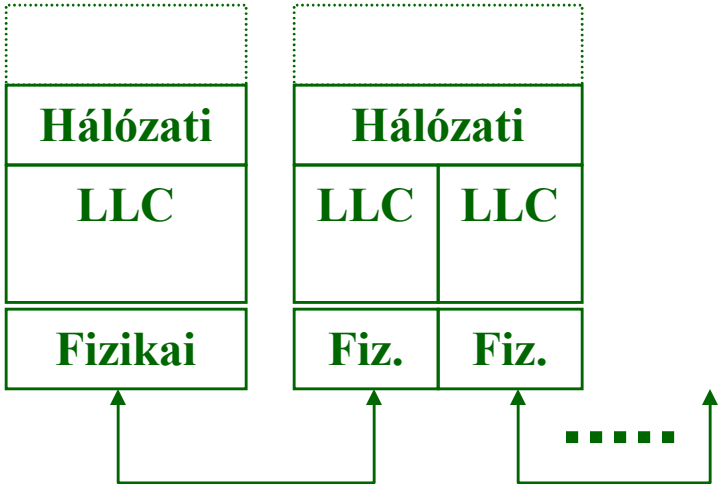
Logical Link Control
Medium Access Control

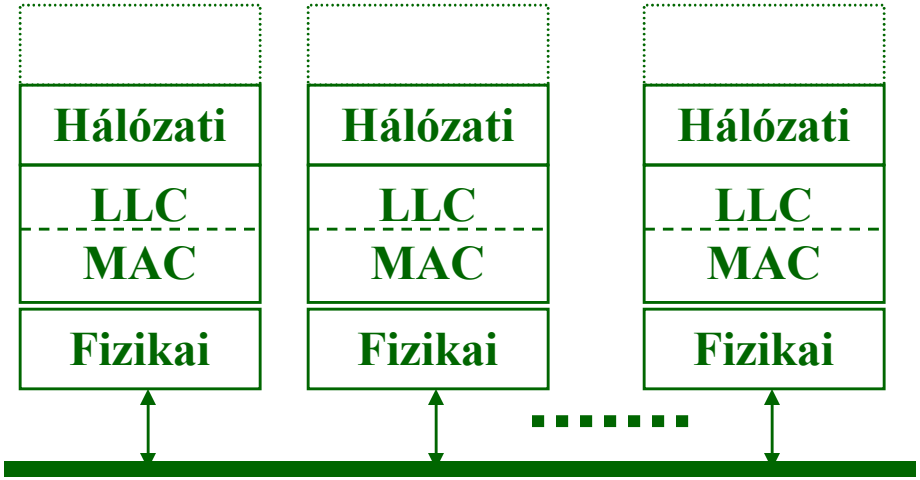




Az adatkapcsolati réteg

- **Pont-pont** kapcsolat esetén **csak LLC**
  - **L**ogical **L**ink **C**ontrol
- **Üzenetszórásos** csatorna esetén **MAC** + **LLC**
  - **M**edium **A**ccess **C**ontrol + **L**ogical **L**ink **C**ontrol





Az adatkapcsolati réteg által megoldandó problémák

- Az átviteli hibák kiküszöbölése
- A véges (esetleg változó) adatátviteli sebességből adódó késleltetés kezelése
- A gépek véges feldolgozási képességeinek kezelése (esetleg változó)
  - véges feldolgozási idő,
  - véges fogadókapacitás
    - gyors adó eláraszthatja a lassú vevőt (flooding) → forgalomszabályozás

A hálózati rétegnek nyújtott szolgálatok

- **Emlékeztető:**
  az **OSI szabványos szolgálat-primitívjei**:
  kérés (request); **bejelentés** (indication);
  válasz (respond); **megerősítés** (confirm).
- **A lehetséges szolgálat osztályok:**
  - **Nyugtázás nélküli összeköttetés-mentes szolgálat**
  - **Nyugtázott összeköttetés-mentes szolgálat**
  - **Nyugtázott összeköttetés-alapú szolgálat**
- **Az adatkapcsolati réteg funkcióinak ezeket kell biztosítania.**

Nyugtázás nélküli összeköttetés-mentes szolgálat

- **Összeköttetés-mentes**
  - **Az adategységek (keretek) egymástól függetlenül kerülnek feladásra**
    (a keretek önállóan vannak megcímezve)
    (az üzenetszórásos csatornák (MAC) általában összeköttetés-mentesek - LAN)
  - **Gyorsabb** (nincs kapcsolat-felépítés, bontás)
- **Visszajelzés nélküli**

  ⇒ nem biztos, hogy hibamentes
  - **alacsony hibaarány esetén lehet jó**
    (kisebb lehet az „overhead” nem kell hibakezelés)
  - **a hibákat (valamelyik) felsőbb réteg kezeli**
    (az veszi észre, az ismétel stb.)

Nyugtázott összeköttetés-mentes szolgálat

- **Összeköttetés-mentes**
  - **Az adategységek (keretek) egymástól függetlenül kerülnek feladásra** (az üzenetszórásos csatornák (MAC) általában összeköttetés-mentesek - LAN)
  - **Gyorsabb** (nincs kapcsolat-felépítés, bontás)
- **Hibamentes átvitelt valósít meg**
  - **nyugták jelzik a keretek hibátlan megérkezését.**
  - Ha tipikusan több hiba van,
    jobb a hibakezelést alacsony szinten intézni
    (kevesebb adatot kell ismételni)
    (bár ott a hibaszűrés bevezetése több overhead-et (veszteséget) jelent, mintha nem lenne)

Az átviteli hibák kezelése

- **Gyakori, esetleg magából a MAC protokollból** (pl. CSMA versengés) **adódó hibákat**
  ⇒ magában a MAC protokollban kell kezelni
- **Azonban ez a hibáknak csak egy része** (pl. ha a címzett nem létezik, akkor sem kapja meg, ha nincs ütközés).
- **A tipikus (gyakori) hibákat célszerű azon a legalacsonyabb szinten kezelni, ahol azok először detektálhatóak** (Minél magasabban szinten történik a hibakezelés, annál több a hibákból adódó járulékos veszteség)
- **De az alacsonyabb szintek valamilyen szintű hibakezelése nem mentesít a felsőbb szintek hibakezelésének szükségességétől! (Csak a hatékonyságot javítják.)**

Nyugtázott összeköttetés-alapú szolgálat

- **A társelemek „párbeszédet” folytathatnak.**
- **Az adategységek** (keretek) **ellenőrzötten és sorrendhelyesen pontosan csak egyszer érkeznek meg**
  - **Megbízható, a küldési sorrenddel azonos vételi sorrendű keretfolyam**
- **Fázisai:**
  - **az összeköttetés létrehozása** (hívásfelépítés);
  - **a keretek átvitele** (érdemi kommunikáció);
  - **az összeköttetés lebontása** (erőforrás felszabadítás).

Az adatkapcsolati réteg funkciói

- **Keretképzés és behatárolás**
  - **A hálózati réteg adataiból keretek kialakítása és**
  - **a fizikai bitfolyamból a keretek behatárolása.**
- **Hibavédelem**
  - **Az adatok „alkalmas” kódolása, hibafelismerés,**
  - **nyugták küldése és fogadása.**
- **Adatfolyam vezérlés (forgalomszabályozás, flow control)**
  - **az adó adássebességének a vevő fogadóképességéhez igazítása (visszacsatolás → fölöslegesen ne terhelje az adó a csatornát)**
- **Kapcsolatmenedzselés**
  - **Összeköttetés-alapú szolgálat esetén lényeges**
    - **összeköttetés létesítése, bontása, erőforrások lefoglalása, felszabadítása.**

A keretképzés és behatárolás

- **Az adatkapcsolati réteg szokásos működése:**
  - A hálózati rétegből kapott bitfolyamot **diszkrét keretekké alakítja**, melyeket **ellenőrző összeggel lát el**;
  - majd továbbítja a kereteket → bitfolyamból jelfolyam;
  - a kapott jelfolyam bitfolyammá alakítása:
    a **kereteket behatárolása**,
    az **ellenőrző összeg visszaellenőrzése**;
  - a bitfolyamot továbbítja a hálózati rétegnek.
- **Keretképzés és behatárolás ≡**
  kerethatárok jelezése és ezen jelzések felismerése
  (a fölöttes hálózati réteg számára transzparensen)

A keretképzés indokai

- Egyes MAC protokollok korlátos méretű (min., max.), esetleg fix méretű kerteket igényelnek
- A fizikai átvitel késleltetése csökkenthető, ha a keretek rövidebbek
- Egy keret meghibásodási valószínűsége csökkenthető, ha a keretek rövidebbek
- A folytonos bitfolyamon végzett hibafelismeréshez, behatároláshoz elengedhetetlenül szükséges

Keretképzési módszerek

- A négy vizsgált módszer
  - A karakter-számlálásos keretezési módszer
  - A karakter-beszúrásos keretezési módszer
  - A bitbeszúrásos keretezési módszer
  - Az érvénytelen kódmintás keretezési módszer

A karakter-számlálásos keretezés

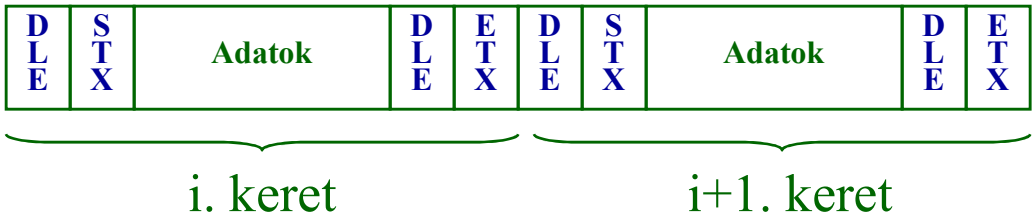
- **A keret-fejrész egy mezőjében megadjuk a keret karaktereinek számát (a keret hosszát)**
- **Feltétel: léteznie kell karakterszinkronnak, fel kell tudni ismerni az egyes karaktereket**
- **Gond: nagyon sérülékeny:**
  ha kiesik a szinkronból, félreértelmezi a kereteket
  - **ha bit elcsúszás, esetleg karaktereket is félreértelmezhet**
- **Hiba esetén nincs újbóli szinkronizálási lehetőség**

A karakter-beszúrásos keretezés

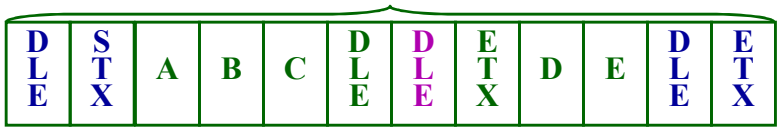
(Character stuffing)

- **A keretek elejét és végét speciális karakter-szekvenciák jelzik.** (Ez is csak karakterorientált módszer)
- **Hiba esetén a keretszinkron könnyen helyreállítható** (csak a keret eleje szekvenciát kell keresni).
- **Keret eleje: DLE, STX (ASCII v EBCDIC karakter párok)**
- **Keret vége: DLE, ETX**
  - **DLE: Data Link Escape**
  - **STX: Start of Text**
  - **ETX: End of Text**
- **Általában terminál kapcsolatoknál alkalmazzák**
  Pl: az IBM BSC (Binary Synchronous Communication protokoll)

Character stuffing



- **Gond a transzparens átvitel:**
  az adatok között ne legyen DLE, STX, vagy DLE, ETX
- **Megoldás: a kerethatárok elhelyezése előtt az adatok közötti valamennyi DLE-t meg kell kettőzni:**
  - **DLE+STX v. ETX párok a kerethatárokat jelezik,**
  - **a DLE+DLE pedig DLE a szövegben.**
- **Pl. a szöveg: A+B+C+DLE+ETX+D+E; akkor a keret**
- **Gond: Karakter-orientált**



A bitbeszúrásos keretezés

(Bit stuffing)

- **Bit-orientált, nem kötődik karakterekhez**
- **Minden keret speciális bitmintával (flag) kezdődik és végződik**

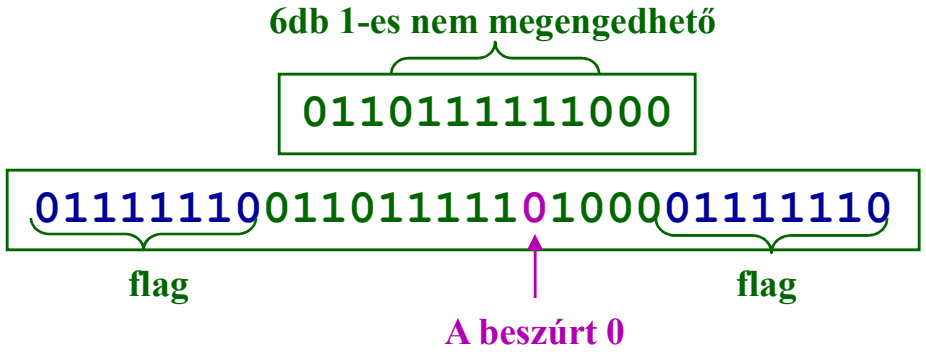
`01111110` **(6 db 1-es bit)**

- **Pl. az IBM SDLC** (Synchronous Data Link Control Protocol) **az ISO HDLC (High Level DLC, az X.25 is ezt használja)**

- **Gond: a bitfolyamban nem lehet ilyen bitminta**

Bit stuffing

- **„Transzparenssé” alakítás:**
  - **A kerethatárok elhelyezése előtt az adó az adat bitfolyamba minden egymás utáni 5 db 1-es bit után beszúr 1db. 0-át;**
  - **a vevő a kerethatárok felismerése és levétele után minden egymás utáni 5 db 1-es bit után kivesz 1db. 0-át.**
  - **Így a 6 bites 1-es sorozat biztosan csak a kertet eleje, vagy vége lehet.**
- Pl. az adatok:
- Ekkor a keret:



- Előny: csak 1 bitet kell beszúrni, legföljebb 5 bitenként

Az érvénytelen kódmintás keretezés

- **A keretek elejét és végét az adatbitektől eltérő módon kódolt jelekkel jelzik.**
  „Kódolás sértéssel”.
- **Pl. Manchester kódolás esetén a „szokásos”**
  - **HL** ⌐↓ **ill. LH** ⌐↑ **helyett**
  - **LL** _ _ **ill. HH** ¯ ¯ **;**
  - Pl. a 802.5 vezérjeles gyűrű ilyen.
- **Előnye: Transzparens, hatékony**
  A keret elejét és végét jelző kódok nem fordulhatnak elő az adatok között (nem kell még biteket sem beszúrni)
- **Hátránya: Nem lehet mindig megcsinálni** (kódolás)

Kombinált

- **Sokszor a karakter-számlálásos módszert kombinálják valamilyen más módszerrel.**
  - **A kerethossz megadás esetleg gyorsíthatja a feldolgozást;**
  - **Egy másik módszer pedig megoldhatja a keretszinkron helyreállítását.**

A hibavédelem

- **A hibavédelem lehet közvetlen**
  - **hibajavító kódolás esetén a vett szóból.**
- **és lehet közvetett**
  - **hibajelző kódolás esetén nyugtázás útján.**

- **Pl: A vevő közvetlenül győződik meg a keret épségéről:**
  - **a keret a formai szabályainak ellenőrzése;**
  - **a kódolás és az ellenőrző összeg ellenőrzése.**
- **Pl: Az adó a hibáról közvetett módon nyugta útján értesülhet**
  - **A pozitív nyugta hiánya jelzi a hibát**
    (Küldött negatív nyugta általában nincs!)

Közvetlenség - közvetettség

- **A vevő**
  - **hibajavító kódolás esetén közvetlenül javíthatja is a hibát,**
  - **hibajelző kódolás esetén csak érzékeli, de saját maga nem javíthatja ⇒ közvetettség (az adó segítsége)**
- **Az adó**
  - **Nem javítható hiba esetén** (visszacsatolás a nyugtázási módszeren keresztül) **újra kell adnia a keretet.**

Közvetlen hibakezelés

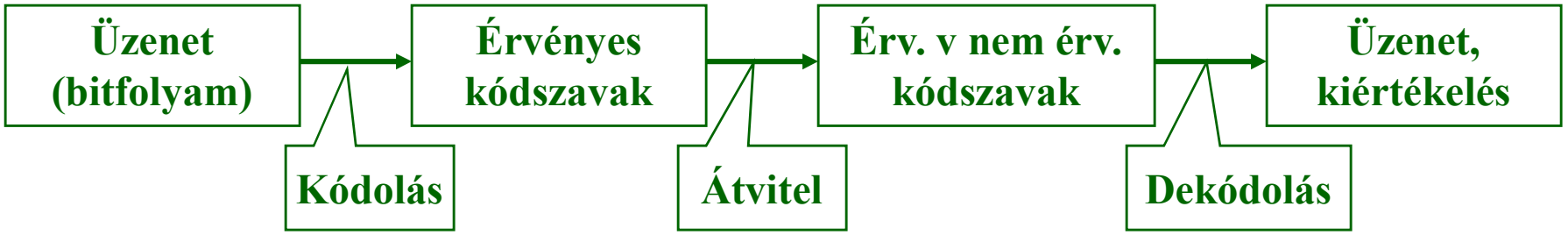
- **Hibajelzés, hibajavítás**
- **Alapvetően a redundanciára épül:**
  a tényleges (hasznos) információknál több információt kell átvinni
- **Mindig véges adategységekre vonatkozik**
  (pl. ezért is kellenek a keretek)

Modell a kódolásra

- **Üzenetfolyam:** **m db bit**
- **Redundáns bitek:** **r db bit**
- **Egy kódszó (code word):** **n = m + r bit**
- **Kódolás: m + r → n (kódszavak képézése)**
- **Kód: az érvényes kódszavak összessége**
- **Lehetséges üzenetszám:** 2^m
- **Lehetséges kódszó szám:** 2^n **(az érvénytelenekkel együtt)**
- **Érvényes kódszavak:** 2^m **darab a** 2^n **-ből.**

Modell a kódolásra

- **Az adatátvitel**



- **A dekódolás eredményeként érvényes vagy nem érvényes kódszavakat kaphatunk**
  - **érvényes kód esetén – dekódolás – üzenet;**
  - **érvénytelen kód esetén esetén kiértékelés:**
    - **hibajelzés, vagy**
    - **hibajavítás.**

Hamming távolság

- **Kódszavak Hamming távolsága** (Hamming 1950):
  - **Két kódszóban az egymástól eltérő bitek száma** (két kódszó kizáró-vagy kapcsolatában az 1-ek száma).
  - **Ha két kódszó H távolsága d, akkor d db 1 bites hibával konvertálhatók át egymásba, vagy**
  - **érvényes kódszóból d számú 1 bites hibával érvényes kódszó adódhat.**
- **Kód Hamming távolsága:**
  - **A kód kódszavai közötti legkisebb Hamming távolság. (A kód az érvényes kódszavak összessége.)**

Tételek

- **Tétel 1: Valamely kód alkalmas d darab hiba felismerésére ⇔ a kód Hamming távolsága d + 1**
  - **Tetszőleges d darab egyedi hiba csak érvénytelen kódszót eredményezhet.**
- **Tétel 2: Egy kód alkalmas d darab hiba javítására ⇔ a kód Hamming távolsága $2 \cdot d + 1$**
  - **Tetszőleges d hiba esetén is „közelebb állunk” az eredeti kódszóhoz, mint bármi más kódszóhoz.**
- **Tétel 3: A hibajavító kód egyben hibafelismerő kód is. A d hibát javítani képes kód 2d hibát képes felismerni.**

Tételek

- **Tétel 4: Tökéletes (közvetlen) hibavédelem, hibajavító kód valamennyi bitre nem létezik.**
  - **Legyen n bites kódszó, valamennyi bitre hibajavító.**
  - **Ekkor a kód H-távolsága $2 \cdot n + 1$ kellene legyen.**
  - **Ez lehetetlen, hiszen $2 \cdot n + 1 > n$**
- **Megjegyzés**
  - **Belátható, hogy a hibajavító kódok alkalmazása nagy redundanciát igényel.**
    Ez csökkenti a hatékonyságot (növeli az overhead-et).
  - **Adatátvitel esetén** (ahol viszonylag alacsony a hibaarány) **hatékonyabb a kevésbé redundáns hibajelző kódot alkalmazni, és újraadni a hibás kódokat (adategységeket)!**

Hibajelző kódok

- **Paritásos kód**
- **Blokkonként paritás**
- **Ciklikus redundancia kód (CRC)**

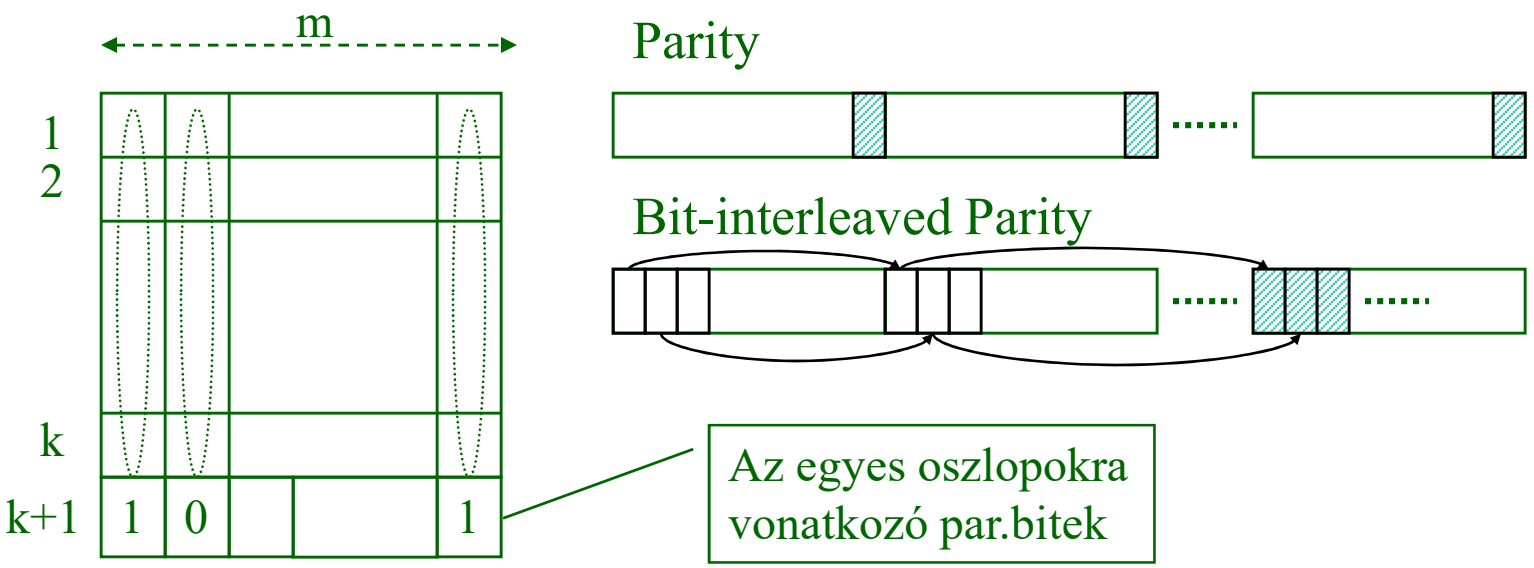
Paritásos kód

- **Az adatok végére egyetlen paritásbitet (parity bit) illeszt. Ennek értékét úgy kell választani, hogy**
  - **Páros paritás (even parity) esetén a kódszóban az 1-esek száma páros (even) legyen (párosra egészít ki), vagy**
  - **Páratlan paritás (odd parity) esetén a kódszóban az 1-esek száma páratlan (odd) legyen.**
    - Pl. Páros paritásbit: az adatbitek kizáró-vagy (XOR) kapcsolata
- **A paritásos kód a kód H-távolságát 1-ről 2-re növeli,**
  - **ezzel 1 egyedi hibát észlelhetünk.**
- **Gond: csoportos hibák esetén 0,5 a hibajelzés valószínűsége**
  (csoportos hiba: több egymásmelletti bit sérül)

Bit-interleaved Parity (Blokkonkénti paritásbit)

- **Egymástól távoli biteket vizsgál és azokból képez paritásbitet** (csoportos hibák észlelése)
- **Képzési módszer:**
  - **az adatbitekből „mátrixot” alkotunk;**
  - **és a paritást oszloponként képzi;**
  - **majd a kapott paritásbiteket a mátrix utolsó sorának teszi;**
  - **a biteket soronként küldjük el.**

Bit-interleaved Parity



- **Belátható: egyetlen, max. m bites csoporthibát képes észlelni;**
- **az m+1 hosszú csoporthiba már észrevétlen maradhat;**
- **Pl: észrevétlen maradhat az i. és i+m. bit együttes hibája.**

Ciklikus redundancia kód

- Cyclic **R**edundancy **C**heck **(CRC)**
- **Más néven: polinom-kód (polinomial code)**
- **Elgondolás:**
  - **tekintsük a bitfüzéreket 0 és 1 együtthatójú polinomoknak,**
  - **azaz egy m bites keret m-1 fokú polinom:**
  - $M(x)=a_{m-1}\, x^{m-1} + \ldots a_1\, x + a_0$

Példa: $\overset{4\ \ 3\ \ 2\ \ 1\ \ 0}{1\ 0\ 0\ 1\ 1} \rightarrow x^4 + x + 1$

Ciklikus redundancia kód

- **Elgondolás** (folytatás)**:**
  - **Legyen M(x) a védendő keret**
  - **Legyen G(x) az r+1 bites, r-ed fokú generátor polinom,**
    - **úgy, hogy m > r+1**
      (hosszabb a keret, mint a generátor polinom),
    - **és a G(x) generátor polinom mind az adóban, mind a vevőben egyforma, előre adott.**
- **Alapelv:**
  - **Legyen R(x) az ellenőrző összeg (checksum) az az r-1-ed fokú** (r bites) **polinom, melyet a védendő keret M(x) végéhez fűzve, azzal együtt G(x)-szel maradék nélkül osztható polinomot alkot.**
    Ahol az osztás a moduló 2 aritmetikán alapul.

0 + 0 = 0	0 - 0 = 0
0 + 1 = 1	0 - 1 = 1
1 + 0 = 1	1 - 0 = 1
1 + 1 = 0	1 - 1 = 0

Az ellenőrző összeg számítása

- Legyen G(x) r-ed fokú
- Illesszünk az M(x) védendő keret mögé r db 0-át (hogy ez m+r bites legyen): $M(x)*x^r$
- Osszuk ezt el (mod-2 algebra szerint) a G(x) generátor polinommal
- Az osztás maradéka lesz az R(x) keresett ellenőrző összeg
- A keresett kódszó: $T(x) = M(x)*x^r + R(x)$

- Állítás:
  T(x) maradék nélkül osztható a G(x)-szel.

Az állítás bizonyítása

- **A moduló-2 aritmetika esetén:**
  (Kizáró vagy - XOR, nincs átvitel)

0 + 0 = 0	**0 - 0 = 0**
0 + 1 = 1	**0 - 1 = 1**
1 + 0 = 1	**1 - 0 = 1**
1 + 1 = 0	**1 - 1 = 0**

- **Bizonyítás:** $\frac{M(x)\cdot x^r}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$ és

$$\frac{T(x)}{G(x)} = \frac{M(x)\cdot x^r + R(x)}{G(x)} = Q(x) + \underbrace{\frac{R(x)}{G(x)} + \frac{R(x)}{G(x)}}$$

Ez mod-2 aritmetikában 0 (A + A = 0)

Hiba esetén:

- **T(x) → T'(x) = T(x) + E(x)**
  - **ahol E(x)-ben 1 van, ott a T(x) megváltozott!**
- **Ugyanis**

$$\frac{T'(x)}{G(x)} = \frac{T(x)+E(x)}{G(x)} = \underbrace{\frac{T(x)}{G(x)}}_{\text{Itt nincs maradék}} + \underbrace{\frac{E(x)}{G(x)}}_{\text{Ennek maradéka jelzi a hibát}}$$

- **Ebből tétel: Minden E(x)-re hibát jelez, ami nem osztható G(x)-szel**
  - **Pl. egyetlen bithiba esetén $E(x) = x^i$; ahol az i-ik bit hibás**
- **⇒ Ha G(x) két, vagy többtagú, mindig jelzi az egyes bithibát!**

Szabványos generátor polinomok

- **CRC-12 = $x^{12} + x^{11} + x^3 + x^2 + x + 1$**
- **CRC-16 = $x^{16} + x^{15} + x^2 + 1$**
- **CRC-CCITT = $x^{16} + x^{12} + x^5 + 1$**
- **Tulajdonságaik (CRC-16; CRC-CCITT)**
  - **észlelik az összes 1-es és 2-es hibát;**
  - **az összes páratlan hibás bit hibát;**
  - **a 16-os, v rövidebb csoportos hibát;**
  - **a 17-es csoportos hibák 99,997%-át.**

- **A CRC algoritmusok számításához egyszerű, léptető-regiszteres áramkör építhető, Ez HW, ezt használják.**

Elemi adatkapcsolati protokollok

- **Adatkapcsolati rétegek közötti protokoll**
- **Az adatkapcsolati réteg funkciói:**
  - **Keretképzés és behatárolás**
  - **Hibavédelem**
    - **Közvetlen (hiba észlelő, javító kódolás)**
    - **Közvetett (nyugta)**
  - **Adatfolyam vezérlés**
  - **Kapcsolat menedzselés**

  } **Elemi adatkapcsolati protokollok**

- **Közvetett hibavédelmet és adatfolyam vezérlést (forrás-vevő szinkronizálást) valósítanak meg.**
- **Építenek a keretképzésre és a közvetlen hibaészlelésre.**

Elemi adatkapcsolati protokollok

- **A vizsgálatok előfeltevései**
  - **A fizikai, az adatkapcsolati és a hálózati réteg**
    - **független folyamatok, melyek üzeneteken keresztül kommunikálnak ⇒ nincs egymásrahatás „átlátszóak”**
  - **Két állomást vizsgálunk, melyek a hálózati rétegtől kapott végtelen hosszú üzeneteket akarnak küldeni egymásnak**
    - **az adónak nem kell várnia a küldendő adatra, az mindig rendelkezésre áll.**

Általános előfeltételek

- **Az adatkapcsolati keretek általános felépítése:**

Fejrész	Információs rész	Farokrész
Header	Data	Trailer

A fejrészben lehet:

keret típus:
- információs keret (adatok)
- vezérlő keret
sorszám,
nyugta információ

A farokrészben:
hibavédelmi információk lehetnek

Az adatrész:
a hálózati réteg adatai

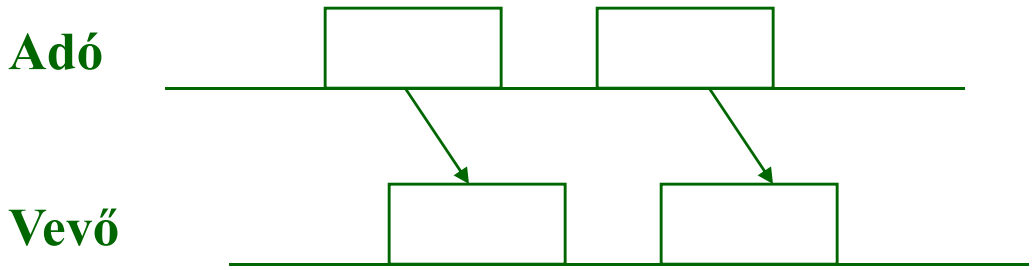
- **Követelmény: hibamentes, sorrendhelyes átvitel**

A vizsgált protokollok

- **Korlátozás nélküli szimplex (utópia) protokoll**
- **Szimplex megáll és vár protokoll**
- **Szimplex protokollok zajos csatornára**
- **Forgóablakos protokollok**

Korlátozás nélküli szimplex (utópia) protokoll

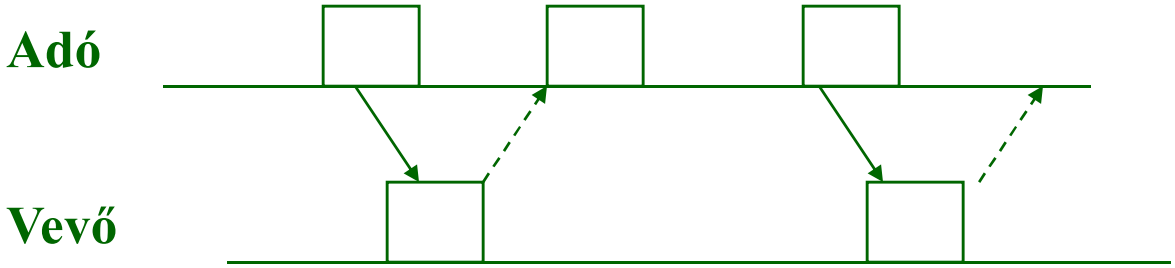
- **További előfeltevések**
  - **végtelenek a bufferek (a vevő sohasem telik be)**
  - **hibamentes a csatorna**
- **Működés:**
  - **Az adó folyamatosan ad**
  - **a vevő folyamatosan vesz**
- **Egyirányú, nincs adó-vevő szinkronizálás**



(Utópisztikus nincs szükség sem hibajavításra, sem adatfolyam vezérlésre)

Szimplex megáll és vár protokoll

- **További előfeltevések:**
  - **Véges bufferek (a vevő betelhet, szükséges a forgalomszabályozás);**
  - **Hibamentes a csatorna.**
- **Működés:**
  - **(1) adó ad egy keretet, majd várakozik;**
  - **vevő veszi a keretet, majd nyugtát küld;**
  - **adó veszi a nyugtát, majd folytatja az (1) ponttól**



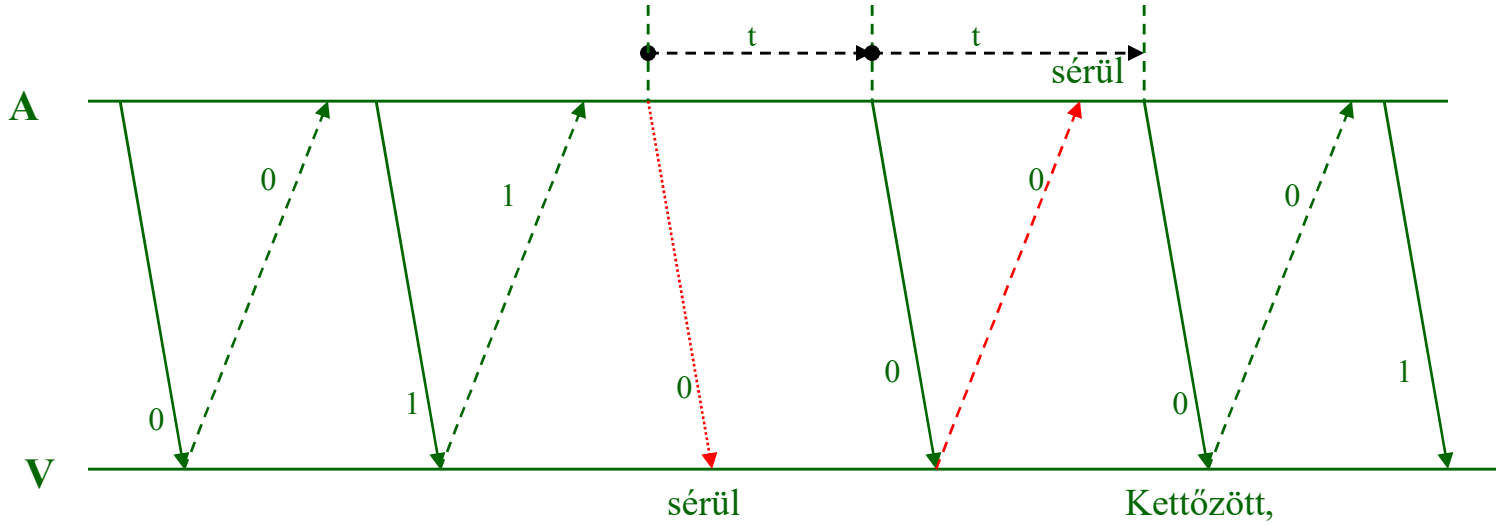
Az adónak meg kell várnia a vevő nyugtáját ⇒ visszacsatolás
⇒ szinkronizálja az adó sebességét a vevő sebességéhez ⇒ forgalomszabályozás
Gond: Nem jó zajos csatornára (a nyugtavesztésen elakad) előfeltevés a hibamentesség

Szimplex protokoll zajos csatornára

- PAR: Positive Acknowledgement with Retransmission
- További előfeltevések:
  - véges bufferek,
  - zajos csatorna.
- Működés
  - (A1) az adó ad egy keretet, majd
  - (A2) beállít egy időzítőt és várakozik
  - (A3) az adó az időzítő lejárta esetén újraadja a keretet, majd megy (A2)-re
  - (A4) az adó nyugta vétele esetén megy az (A1)-re
  - (V1) a vevő érvényes keret vétele esetén továbbítja az a felettes rétegnek, majd nyugtát küld.
  - (V2) a vevő érvénytelen keret vétele esetén vár
- A PAR protokoll
  - Forgalomszabályozást és
  - hibavédelmet biztosít (hiba esetén újraküldéssel)

Szimplex protokoll zajos csatornára

- **Gond:**
  - **nyugta sérülés esetén ugyanazt a keretet az adó többször is elküldi**
  - **Ezért az egymás utáni kereteket meg kell tudni különböztetni**
  - **Elég 1 bit az adott keretek „megszámozására”**



A nyugtának is kell sorszám (elkéső ismételt nyugta)!

Forgóablakos protokoll (Sliding window)

- **További előfeltevések:**
  - **végesek a bufferek,**
  - **zajos a csatorna.**
- **Megoldandó probléma még:**
  - **Nagy átviteli késleltetés esetén túl hosszú kivárni a „körülfordulási időt” (amíg a nyugta visszaér az újabb keret elküldése előtt).**
- **Megoldás:**
  több nyugtázatlan keret elküldését kell lehetővé tenni, ugyanakkor sorrendhelyes keretátvitelt kell biztosítani.

Forgóablakos

Forgóablakos protokoll

- **Az elküldött kereteket „ciklikusan” sorszámozzuk**
  - **Pl. n bites sorszám esetén 0, 1, …. (2^n-1), 0, 1, ... (2^n-1), 0, ...**
- **Az adó „ablakot” képez, (max. $w_{adó}$ szélességűt), melybe az elküldött, de még nyugtázatlan keretek sorszámai kerülnek.**
- **A vevő „ablakot” képez, (állandó $w_{vevő}$ szélességűt), mely azokat a sorszámokat tartalmazza, mely sorszámú kereteket hajlandó fogadni.**

Forgóablakos

A forgóablakos protokoll működése

(A1) Adó a hálózati rétegtől kap egy keretet, tárolja pufferében, elküldi, ugyanekkor beállít egy ehhez tartozó időzítőt, valamint
→ növeli 1-gyel az adóablak felső határát $\mathbf{w_{Adó}}$,
Mindezt ismétli, míg el nem éri a max. ablakszélességet (ha egyáltalán eléri).

(A2) Az adó, ha valamely kerethez tartozó időzítő lejárt,
→ azt a keretet a pufferéből újraadja.

(A3) Az adó, ha nyugtát vesz, az adó ablak alsó határát
→ a nyugta sorszámát követő értékre állítja.

Vevő vesz egy keretet.

(V1) Ha a vett keret hibás, → nem csinál semmit (nem nyugtáz),

Ha a vett keret ép, akkor

(V2) Ha kívül van a vevőablakon: nem csinál semmit (eldobja és nem küld nyugtát),

Ha beleesik a vevőablakba, akkor nézi, hogy a vett keret sorszáma azonos-e a vevőablak alsó határával.

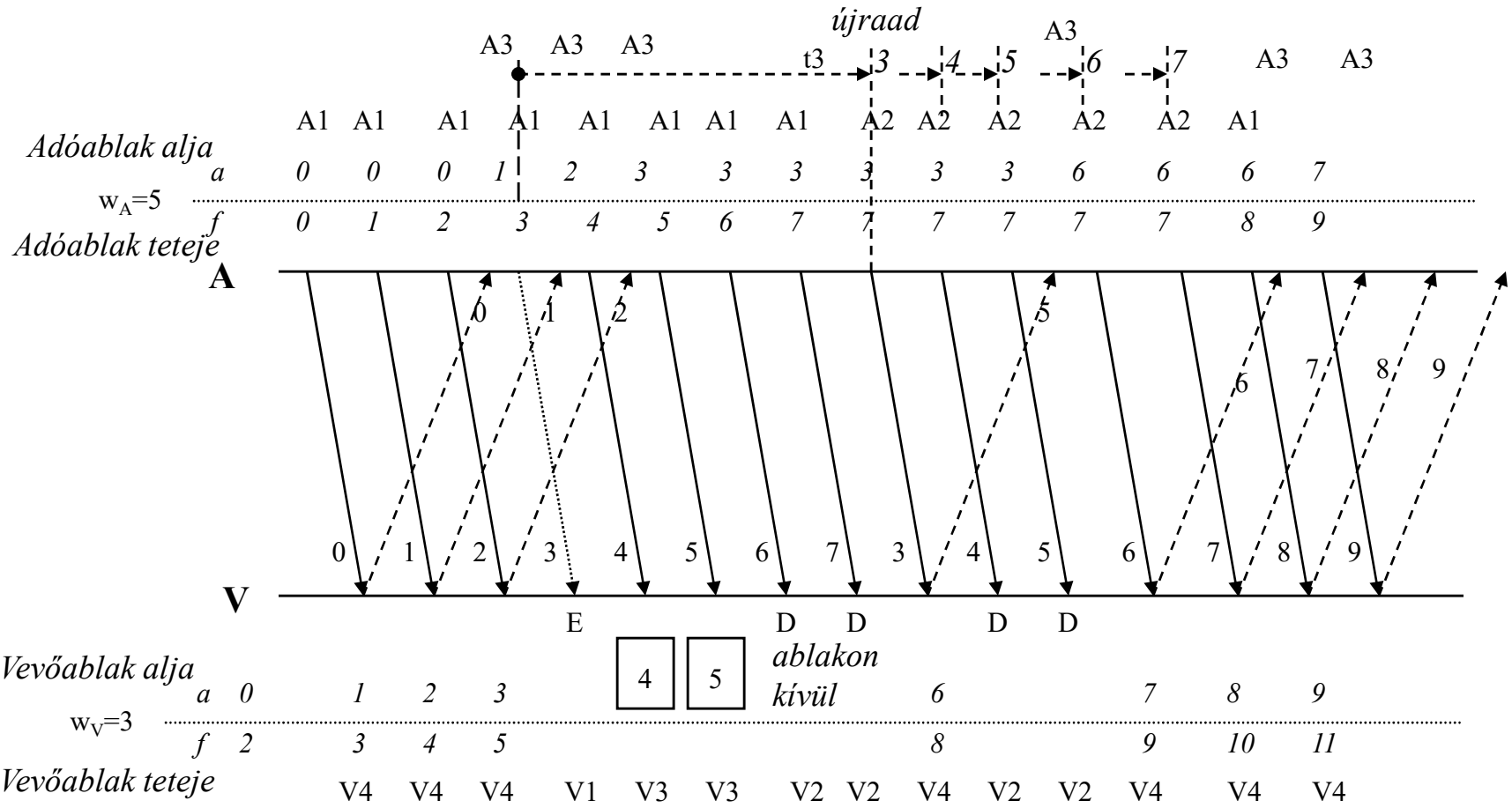
(V3) Nem azonos: veszi a keretet és tárolja a pufferében.

(V4) Azonos: veszi a keretet (továbbadja a hálózati rétegnek, ha pufferében vannak még ezen kívül soron következők, akkor azokat is továbbítja).
→ Ugyanekkor nyugtázza a folytonos sorrendbeli legnagyobb sorszámú keretet, továbbá az ezt követő sorszámra állítja a vevőablak alsó határát. A vevőablak fix méretű $\mathbf{w_{Vevő}}$ → a felső határ is elmozdul ennyivel.

Kiegészítés: *A legutolsó nyugtázott keret újraküldése esetén a vevő a nyugtát megismétli!*

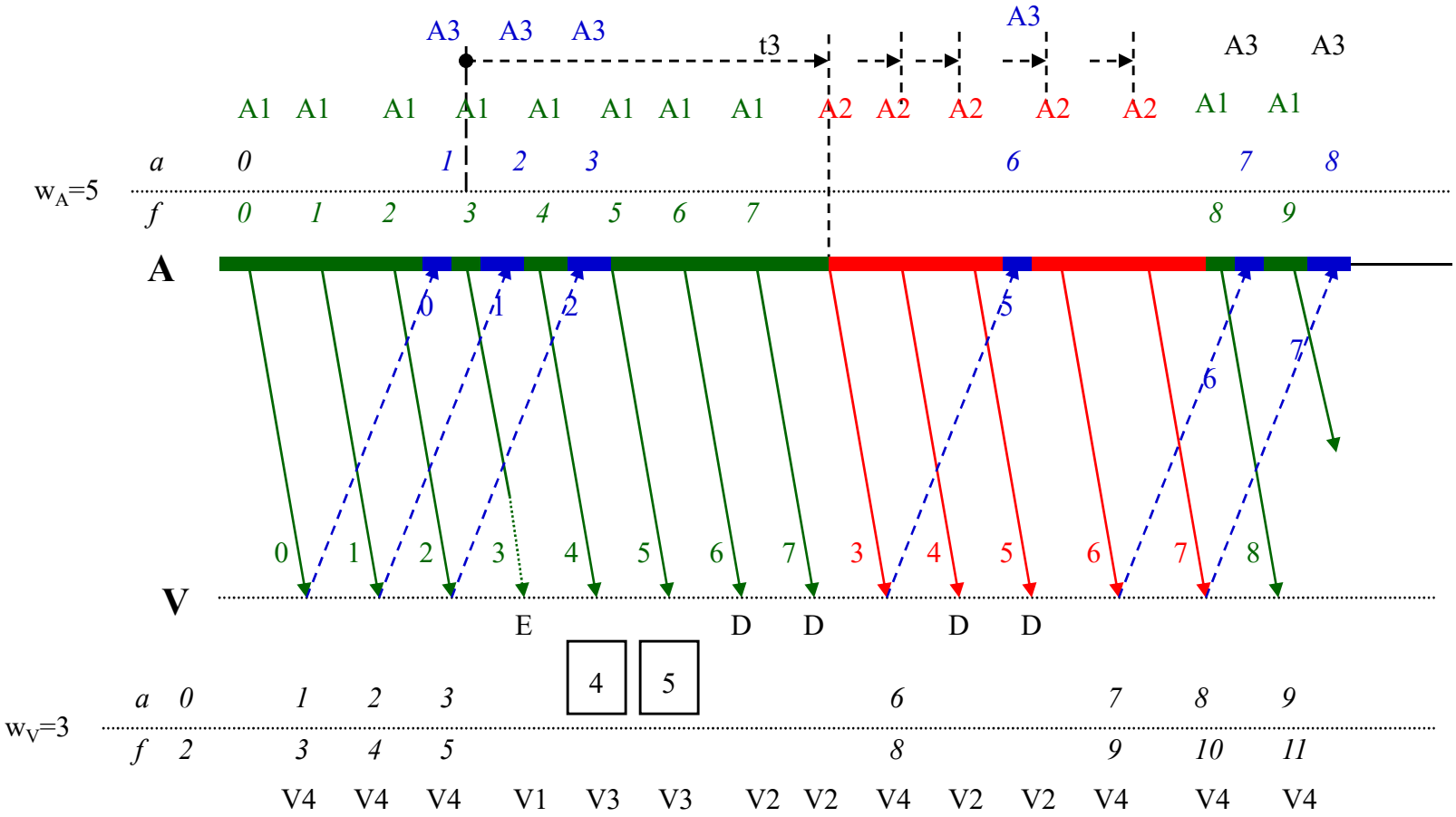
Pl. a forgóablakos protokoll működésére

- **Legyen $w_{Adó} = 5$; $w_{Vevő} = 3$;**
- **A 3. sorszámú keret hibásan érkezik**

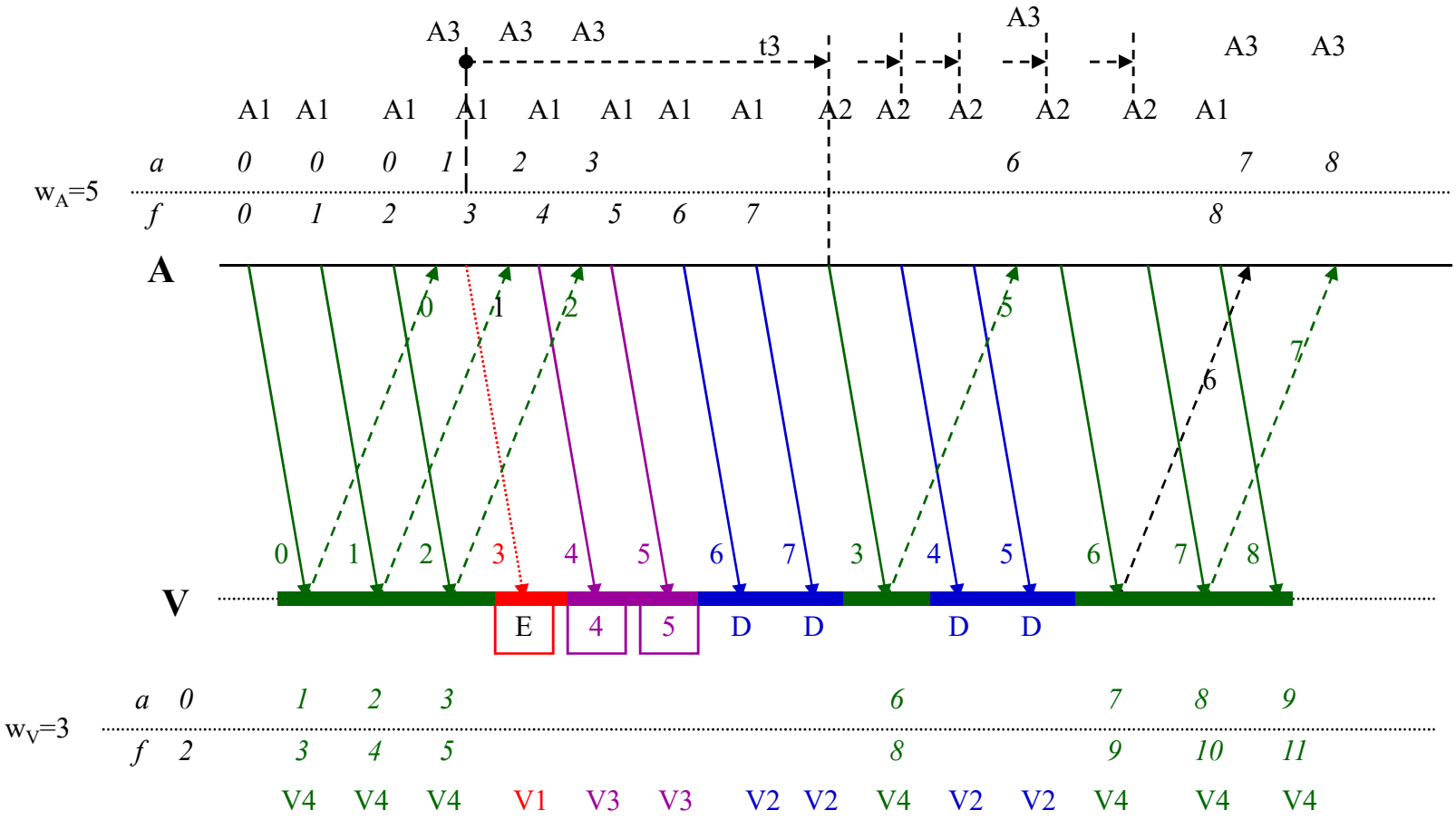


Az adó szemszögéből nézve

(A1) Adó ad, beállít egy időzítőt, növeli az ablak felső határát (időzítő beállítás nem mindig látszik)
(A2) Ha az időzítő lejárt, keretet a pufferéből újraadja.
(A3) Az adó, ha nyugtát vesz, ablak alsó határát a nyugta sorszámát követő értékre



A vevő szemszögéből nézve



(V1) Ha a vett keret hibás, nem csinál semmit (nem nyugtáz),

(V2) Ha kívül van a vevőablakon: nem csinál semmit (eldobja),

(V3) Nem azonos az alsó határral: veszi a keretet és tárolja a pufferében.

Forgóablakos

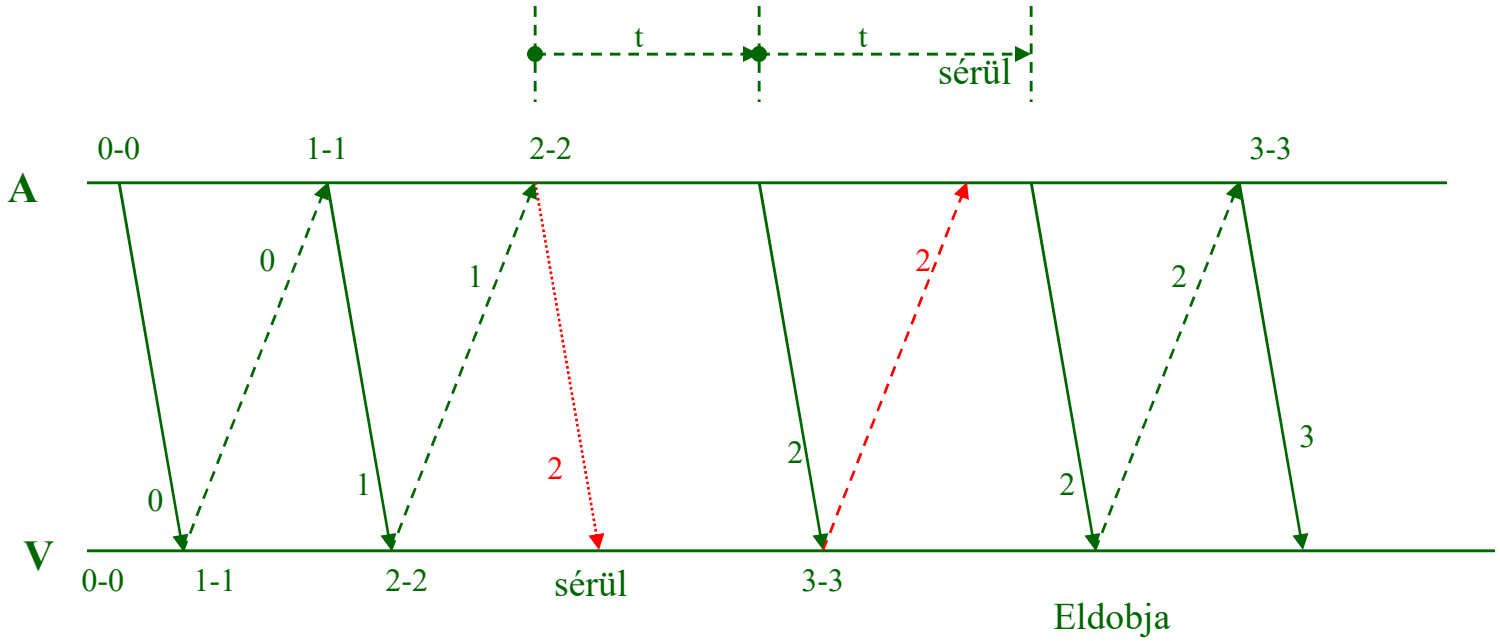
(V4) Keretet továbbítja, ha vannak pufferében soron következők, azokat is továbbítja. Nyugtázza a legnagyobb sorszámú keretet, az ezt követőre állítja az ablak alsó határát, a felső határt is elmozdítja ennyivel.

Az ablakméret megválasztása

- A választást a
  - terjedési késleltetés (adóablak)
  - és a hibaarány határozhatja meg (vevőablak).
- Minél nagyobb késleltetés, annál szélesebb az adóablak:
  - Ne kelljen megállnia az adónak míg a nyugta visszaér (az ablak legalább a körbejárási ideig kitartson)
- Minél nagyobb a vevőablak, annál kevesebbet kell megismételni hiba esetén:
  - A hibás keret újbóli vétele után a vevő a korábban vett rákövetkező jókat is nyugtázza
  - A vevőablak mérete max. az adóablak

A PAR protokoll

- **$w_a = 1$; $w_v = 1$ ablakméretű forgóablakos protokoll**



Megjegyzés: az „eredeti” PAR-nál a csomagszámozás 0, 1 volt.
Itt a számlálómező szélesebb

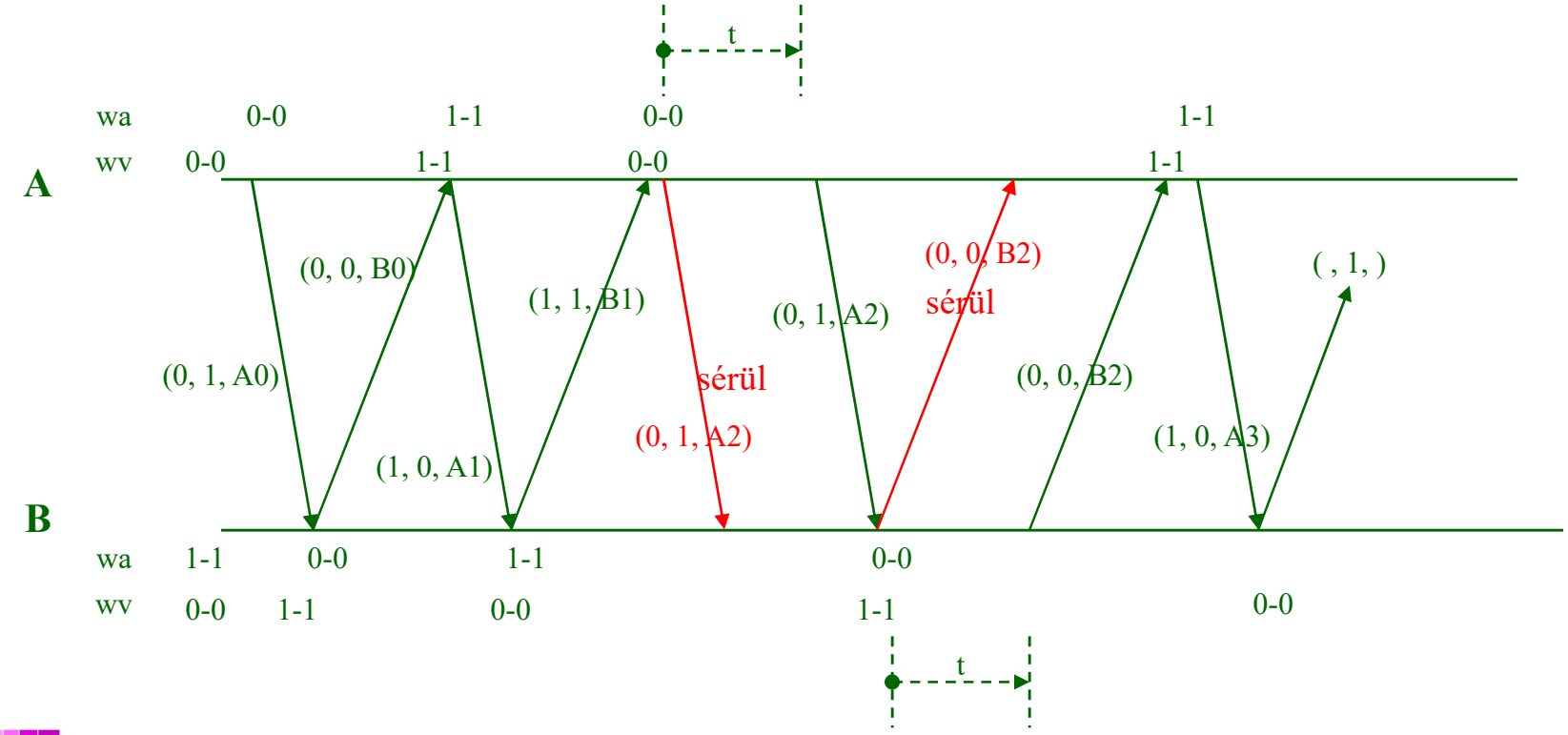
Ráültetéses nyugtázás (piggybacking)

- **duplex (kétirányú) a csatorna esetén:**
  - **⇒ ráültetéses (piggybacking) nyugtázás lehetséges:**
  - **a vevő a nyugtákat az ellenirányú adatkeretekre ültetve küldi el**
    (az adatkeretben néhány bit fenntartva a nyugta számára)
  - **Gond: ha sokáig nincs ellenirányú forgalom ⇒ késik a nyugta**
  - **Megoldás: a vevő, ha nyugtát küldene, időzítést állít be és ha ez letelik míg nincs adnivalója (amire ráültethetné a nyugtát),**
    ⇒ akkor önálló nyugtakeretet küld.

piggyback = (US) pick-a-back = *biz gyerm* vknek a hátán/vállán ülve

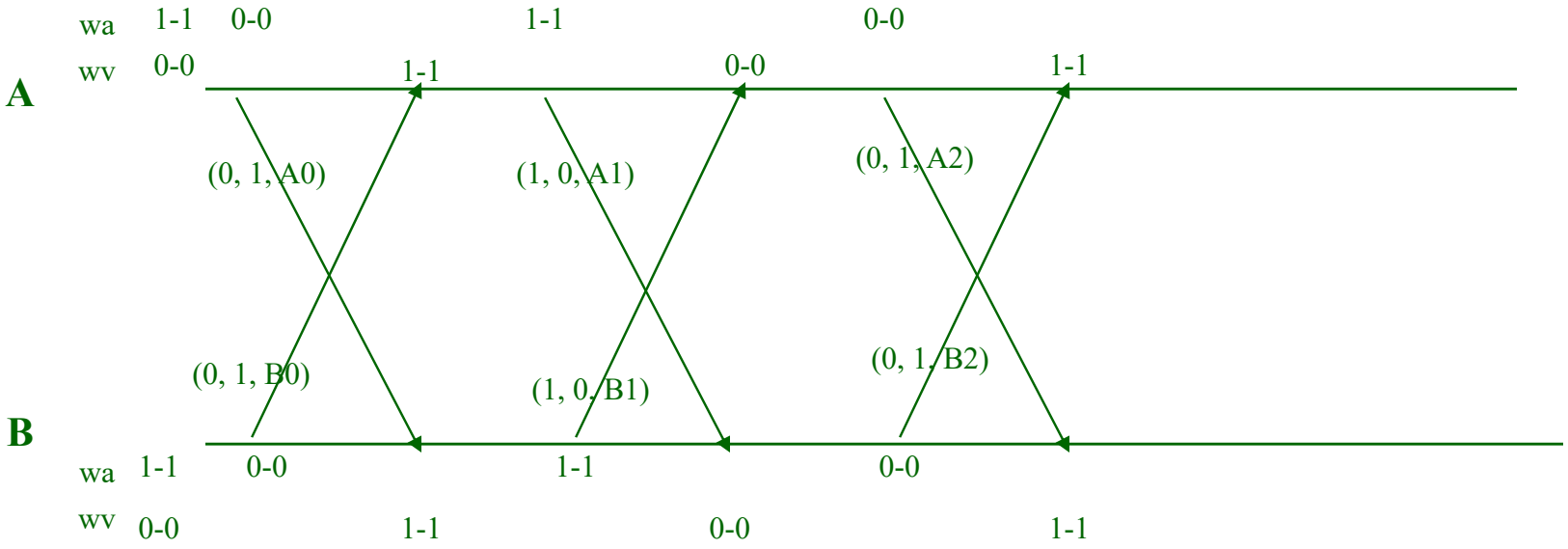
PAR ráültetéses nyugtázással

- **A és B állomások, mindkettő adó és vevő (kétirányú adatforgalom)**
- **Jelölés: keretsorsz, nyugtasorsz, keret (sorszám 1 bites)**
  - **Pl. 0, 0, Ai; vagy 1, 1, Bj**

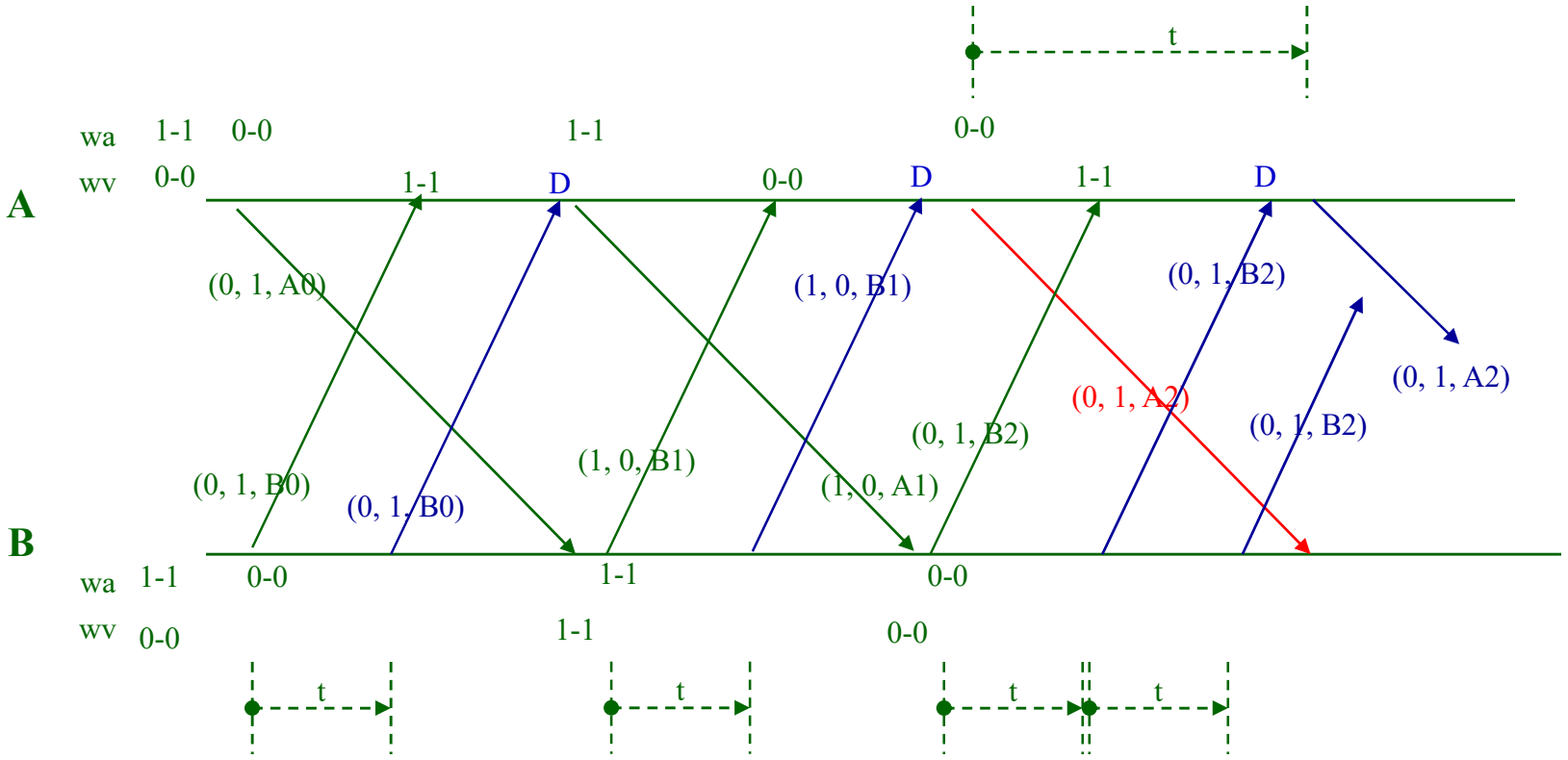


PAR ráültetéses nyugtázással

- **A és B állomások, mindkettő adó és vevő (kétirányú adatforgalom)**
- **Jelölés: keretsorszám, nyugtasorszám, keret (a sorszám 1 bites)**
  - **Pl. 0, 0, Ai; vagy 1, 1, Bj**



Ha túl rövid az időzítés B-nél

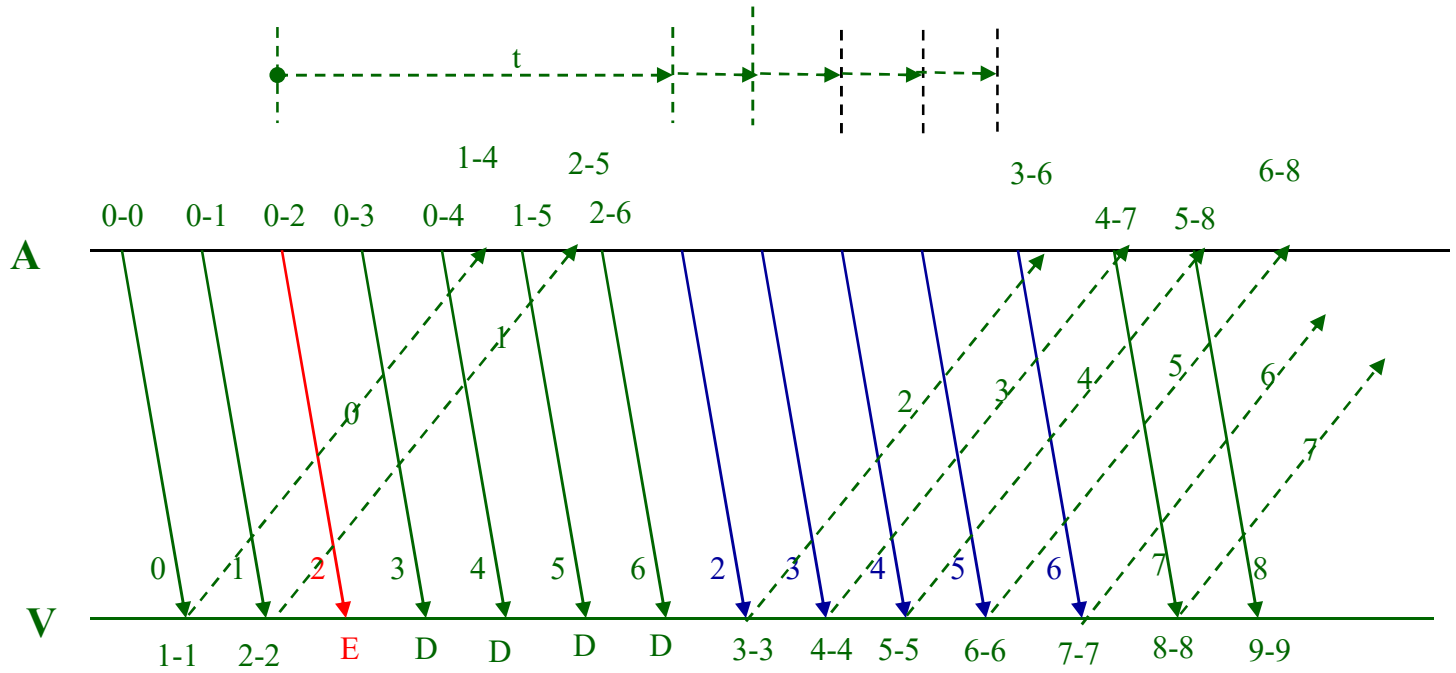


Gond: Sok a felesleges ismétlés, de még így is (sorrend)helyesen működik (a hatékonyságot rontja)

Visszalépés n-nel (go back n) hibakezeléses protokoll

- **Legyen**
  - **w_a = n (nagy adóablak),**
  - **w_v = 1 (vevő egyesével nyugtáz).**
- **Hiba esetén a vevő az összes rákövetkező keretet eldobja, míg az adó meg nem ismétli a hibás keretet** (ezért hívják „visszalépésesnek”).

Go back n protokoll



Gyakori hibák esetén nem hatékony.
Nagyobb vevőablak esetén a vevő a jól beérkezett kereteket az ablakméret erejéig eltárolja, így amikor az adó megismétli a hibásat, a jókat is nyugtázza (kevesebbet kell újraadni).

Ezért nevezik a teljes forgóablakos protokollt *szelektív ismétlés*nek is.

A ciklikus-számlálómező mérete

- **Az ablakméretek határozzák meg**
- **n bites ciklikus számlálómező esetén a sorszámok 0 - (2^n - 1) közöttiek lehetnek**

a) Ekkor az adó max ablakmérete 2^n - 1 lehet.

b) Ekkor a vevőablak max mérete 2^{n-1} lehet.

Tegyük fel, nem tartjuk be az a)-t

- Legyen n = 3 (0 - 7 sorszámok), $w_a = 8$
- Adó elküldi a 0-7 sorszámú kereteket (8 db)
- Az adó 7-es sorszámú nyugtát kap, ezért
- újabb 8 keretet küld, (0-7 sorszámokkal most is).
- Az adó újból 7-es sorszámú nyugtát kap:
  - nem lehet eldönteni, hogy ez az újabb 8 keret nyugtája,
  - vagy az előző 8 nyugtájának az ismétlése!

max. $2^n - 1$

Tegyük fel, nem tartjuk be a b)-t

- Legyen n = 3 (0 - 7 sorszámok), $w_a = 7$, $w_v = 5$
- Adó elküldi a 0-6 sorszámú kereteket (7 db)
- A vevő elfogadja mindet, nyugtázza, ablakát 7 - 3 -ra állítja, de a nyugták elvésznek!
- Adó újraadja az első 7-et (0-6 sorszámokkal most is, mert az időzítése lejárt).
- Ezekből a vevő elfogadja és puffereli a 0-3 számú 4 db-ot, azokat a következő sorozatnak gondolva …
- A bajt az „átlapolódás” okozza. A vevőnél nem szabad átlapolódnia a sorszámoknak. A vevőnél az ablakméret max = (adó max sorszám + 1) / 2 lehet! Az i. sorszámú keret az i mod w_v pufferbe kerül, nem lehet átlapolódás!

max. 2^{n-1}

Összefoglalás

- Általános megállapítások
- Keretképzés - felismerés
- Hibakezelés, ami lehet
  - közvetlen, ill. közvetett.
- Adatfolyam vezérlés
- Elemi adatkapcsolati protokollok

Gyakorlat

- **1. feladat: CRC számítása (adás)**
  - **Legyen az üzenet: 1 1 0 1 0 1** $M(x) = x^5 + x^4 + x^2 + 1$ **(m = 5)**
  - **Legyen a gen. polinom:1 1 0 1** $G(x) = x^3 + x^2 + 1$ **(r = 3)**
- **Adás, R(x) képzése:**

  $M(x)*x^r = x^8 + x^7 + x^5 + x^3$

  $(M(x)*x^r)/G(x) = (x^8 + x^7 + x^5 + x^3) : (x^3 + x^2 + 1) = x^5 + 1$

  $- (x^8 + x^7 + x^5)$

  ———————

  x^3

  $- (x^3 + x^2 + 1)$

  ———————

  $x^2 + 1 \quad = R(x) \rightarrow$ **R: 1 0 1**
- **Adás, T(x) összeállítása:**

  $T(x) = \underbrace{1\ 1\ 0\ 1\ 0\ 1}_{m}\ \underbrace{1\ 0\ 1}_{r}$ $\qquad T(x) = x^8 + x^7 + x^5 + x^3 + x^2 + 1$

Gyakorlat

- **1. feladat: CRC ellenőrzése (vétel)**
- **Vétel, a kódszó ellenőrzése:**

$(x^8 + x^7 + x^5 + x^3 + x^2 + 1) : (x^3 + x^2 + 1) = x^5 + 1$

$-(x^8 + x^7 + x^5)$

$x^3 + x^2 + 1$

$-(x^3 + x^2 + 1)$

0; rendben, nincs maradék.

- **Ugyanez bináris számokkal:**

$M(x)*x^3 =$ 1 1 0 1 0 1 | 0 0 0

G(x) 1 1 0 1 jobbra tolni az első 1-ig, majd kivonni

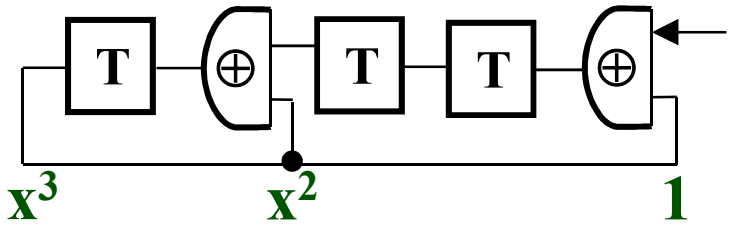
0 0 0 0 0 1 | 0 0 0

1 1 0 1 jobbra tolni az első 1-ig, majd kivonni

1 0 1 = R (nem osztható továbbb)

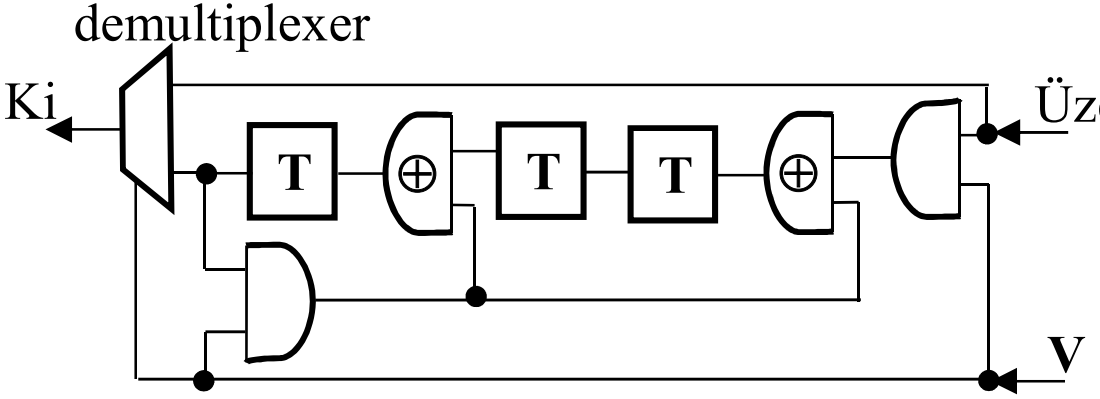
Gyakorlat

- **CRC képzés HW megvalósítás – tárolók és antvalencia (XOR)**
- **Pl: G(x): 1 1 0 1 ($x^3 + x^2 + 1$), r = 3**
- **Kezdetben ∀ tároló ≡ 0**



Ott van átkötés és antvalencia, ahol a G(x)-ben 1-van

- **Ha legelől 1-van ⇒ kivonja G(x)-et, majd léptet**
- **Ha az összes bit belépett ⇒ a tárolókban előáll a maradék**
- **A teljes áramkör (adó):**
  (a vevő mint a fenti ák., csak m+r lép be és a tárolókat nézi, hogy ∀ 0-e (van-e maradéka))



Vezérlés:

m lépésig: V = 1

(kilép az üzenet + maradék a regiszterekben)

r lépésig: V = 0

(kilép az ellenőrző összeg)