Telekommunikációs Hálózatok

6. gyakorlat

ZH időpontok!!!

- **Okt 28.** zárthelyi
- **Nov. 4.** ha szükséges, pótzárthelyi, egyébként továbbhaladás

Feladat 1

- Készítsünk egy ***kliens***-***proxy***-***szerver*** alkalmazást, ahol:
  - a ***szerver*** egy TCP szerver,
  - a ***proxy*** a ***szerver*** irányába egy TCP kliens, a ***kliens*** irányába egy TCP szerver,
  - a ***kliens*** egy TCP kliens a ***proxy*** irányába
- Folyamat:
  - a ***kliens*** küldje a ‚Hello Server' üzenetet a ***proxy***nak,
  - amely küldje tovább azt a ***szerver***nek,
  - amely válaszolja vissza a ‚Hello Kliens' üzenetet a ***proxy***nak,
  - amely küldje tovább azt a ***kliens***nek

Feladat 2: Egyszerű TCP proxy

Készítsünk egy egyszerű TCP alapú proxyt (átjátszó). A proxy a kliensek felé szerverként látszik, azaz a kliensek csatlakozhatnak hozzá. A proxy a csatlakozás után kapcsolatot nyit egy szerver felé (parancssori argumentum), majd minden a klienstől jövő kérést továbbítja a szerver felé és a szervertől jövő válaszokat pedig a kliens felé.

Pl: python netProxy_gyak6_f2.py szalaigj.web.elte.hu 9000

Webböngészőbe írjuk be: localhost:9000

Feladat 3A

- Készítsünk a számológéphez egy proxy-t (select-tel, több kliens is lehet), ami a klienstől kapott TCP kéréseket az UDP serverhez küldi, majd az eredmény a proxyn keresztül vissza a kliensnek.

Feladat 3B

- Egy számológép kliens az UDP szervertől kérje el a TCP-s szerver elérhetőségét!
  - Küldjön egy ‚GET' üzenetet
- A kliens küldjön egy ‚Hello Server' üzenetet a UDP szervernek, aki visszaküldi a TCP szerver elérését, ahova a számokat és az operátort fogja elküldeni.
- A TCP szerver legyen a korábbi számológép szerver

- Hasznos lehet a null ('\x00') karakter eltűntetése:

```
hostname = b'localhost\x00\x00\x00\x00\x00\x00'
hostname = hostname.replace(b'\x00', b'')
```

Hiba felügyelet Hamming távolsággal – emlékeztető

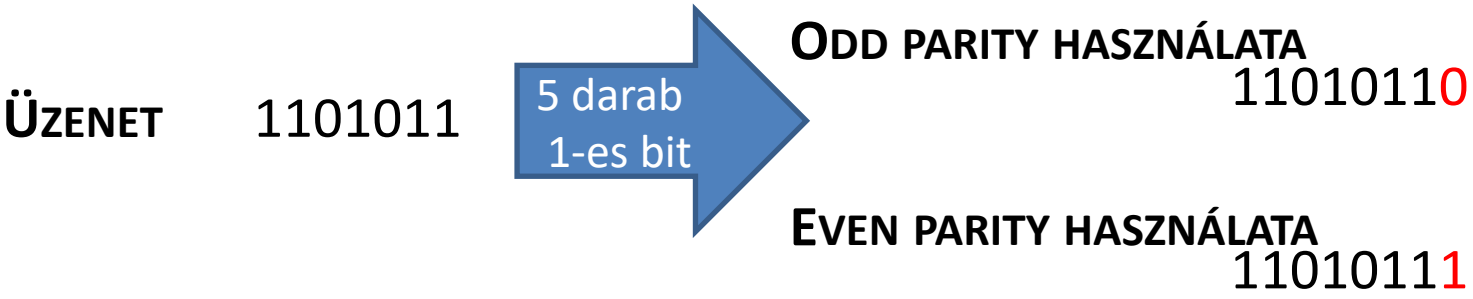
- Hamming távolság: két azonos hosszúságú bitszóban a különböző bitek száma.
- Kiterjesztése azonos hosszúságú bitszavak S halmazára:

$$d(S) \coloneqq \min_{x,y \in S \wedge x \neq y} d(x,y)$$

  - (S halmazt hívják kódkönyvnek vagy egyszerűen kódnak is.)
- *d* bit **hiba felismeréséhez** a megengedett (helyes) keretek halmazában legalább *d+1* Hamming távolság szükséges.
- *d* bit **hiba javításához** a megengedett (helyes) keretek halmazában legalább *2d+1* Hamming távolság szükséges
- Egy $S \subseteq \{0,1\}^n$ **kód rátája** $R_S = \frac{\log_2|S|}{n}$.
  - (a hatékonyságot karakterizálja)
- Egy $S \subseteq \{0,1\}^n$ **kód távolsága** $\delta_S = \frac{d(S)}{n}$.
  - (a hibakezelési lehetőségeket karakterizálja)

Paritás bit használata – emlékeztető

- A paritásbitet úgy választjuk meg, hogy ha a kódszóban levő 1-ek száma
  - **Odd parity** – páratlan, akkor 0 befűzése; egyébként 1-es befűzése
  - **Even parity** – páros, akkor 0 befűzése; egyébként 1-es befűzése



Feladat 4

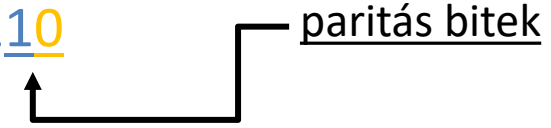
Egyetlen paritásbit által nyújtottnál nagyobb biztonságot akarunk elérni, így olyan hibaészlelő sémát alkalmazunk, amelyben két paritásbit van: az egyik a páros, a másik a páratlan bitek ellenőrzésére.

- Mekkora e kód Hamming-távolsága?
- Mennyi egyszerű és milyen hosszú löketszerű (burst-ös) bit-hibát képes kezelni? (Löketszerű: egymás utáni bitek hibásan jönnek át)

Feladat 4 megoldás

- A kód Hamming-távolsága 2, mivel a páros pozíciókban lévő paritás bit független a páratlan pozíciókban levőtől, külön-külön pedig könnyen látszik, hogy a H-táv 2 → 1 hibát tudunk jelezni (pozíciók paritása alapján egyet-egyet).
- A burst-ös hibánál 3 hosszúságúnál még épp tudjuk jelezni, ha baj van, mivel így vagy a páros vagy a páratlan pozíciókra csak 1 hiba fog esni, azt pedig jelezni fogja a megfelelő paritás bit.

PL. ODD PARITY HASZNÁLATA 110101110



010101100
000101101
001101111
001001110

Feladat 5

- Tekintsük a következő paritás-technikát. Tekintsük az *n* küldendő adatbitet, mint egy *k* × *l* bit mátrixot. Minden oszlophoz számoljunk ki egy paritás-bitet (odd parity) és egészítsük ki a mátrixot egy új sorral, mely ezeket a paritás-biteket tartalmazza. Küldjük el az adatokat soronként.
- Példa *k* = 2, *l* = 3 esetén:

1 0 1

0 1 1

0 0 1

- Hogy viselkedik ez a módszer egyszerű bit-hibák és löketszerű (burst) bit-hibák esetén, ha *k* = 3, *l* = 4? Milyen hosszú lehet egy bitsorozat, melynek minden bitje hibás, hogy a hibázást meg tudjuk állapítani? (Löketszerű: egymás utáni bitek hibásan jönnek át)
- Egészítsük ki a mátrixot egy új oszloppal is, amely minden sorhoz paritás-bitet tartalmaz (két dimenziós paritás technika). Hogyan használható ez a módszer 1-bithiba javítására? Mi a helyzet több bithibával és löketszerű-hibákkal?

Feladat 5 megoldás I.

- Azt kell észre venni, hogy az új sorban számolt paritás-bitek oszloponként függetlenek egymástól. Egy oszlopra számolva a Hamming távolság 2 lesz, mert ha az első *k* = 3 értéke közül egy megváltozik, akkor a paritás-bit is meg fog változni. Tehát oszloponként egy bithibát képes jelezni, és nem tudja javítani.
- Például:

1	1	0	0
0	0	1	1
1	1	0	1
1	**0**	**0**	**1**

Feladat 5 megoldás II.

- A löketszerű bit-hibák esetén az kell, - ahhoz, hogy a hiba tényét egyáltalán meg tudjuk állapítani, - hogy legyen az oszlopok között legalább egy olyan, ahol maximum egy hiba van (az előző feladat alapján). Tehát max. (2*l*-1 =) 7 bit lehet hibás löketszerűen, mert különben két bithiba jutna mindegyik oszlopra.
- Például:

1	1	1	1
1	1	0	0
0	0	0	1
1	**1**	**0**	**1**

Feladat 5 megoldás III.

- Az új sorban levő paritás-bitekkel a hibás oszlopot lehet beazonosítani, míg az új oszlopban levővel a hibás sort. Ezek metszete egyértelműen meghatároz egy bithibát, amit így javítani tudunk.
- Az új oszlopban számolt paritás-bitek sorokra számolva függetlenek egymástól. Itt is igaz, - az új sorra történő érveléshez hasonlóan, - hogy sorokra számolva a Hamming távolság 2 lesz. Tehát soronként egy bithibát képes jelezni.
- Az kell a ***biztos***[1] hibajelzéshez, hogy legyen legalább egy olyan sor vagy oszlop, ahol csak egy bithiba van. Ez azt jelenti, hogy max. három bithibát mindig jelezni fog tudni akárhogyan is oszlanak el a hibák[2], de négyet már nem (gondoljunk a „négyzetes” elhelyezkedésre).

[1]Elképzelhető olyan eset, amikor több hibánál is jelez „véletlenül”, de azzal nem foglalkozunk.

[2]Ehhez az kell, hogy a „sarok bit” is szerepeljen

Feladat 5 megoldás IV.

- Ha a „sarok bit” nincs, akkor már hármat sem tud jelezni mindig, pl. az alábbi eset:

1	1	0	0	**1**		0	1	0	0	**0**
0	0	1	1	**1**		0	0	1	1	**1**
1	0	0	1	**1**		1	0	0	1	**1**
1	**0**	**0**	**1**			**0**	**0**	**0**	**1**	

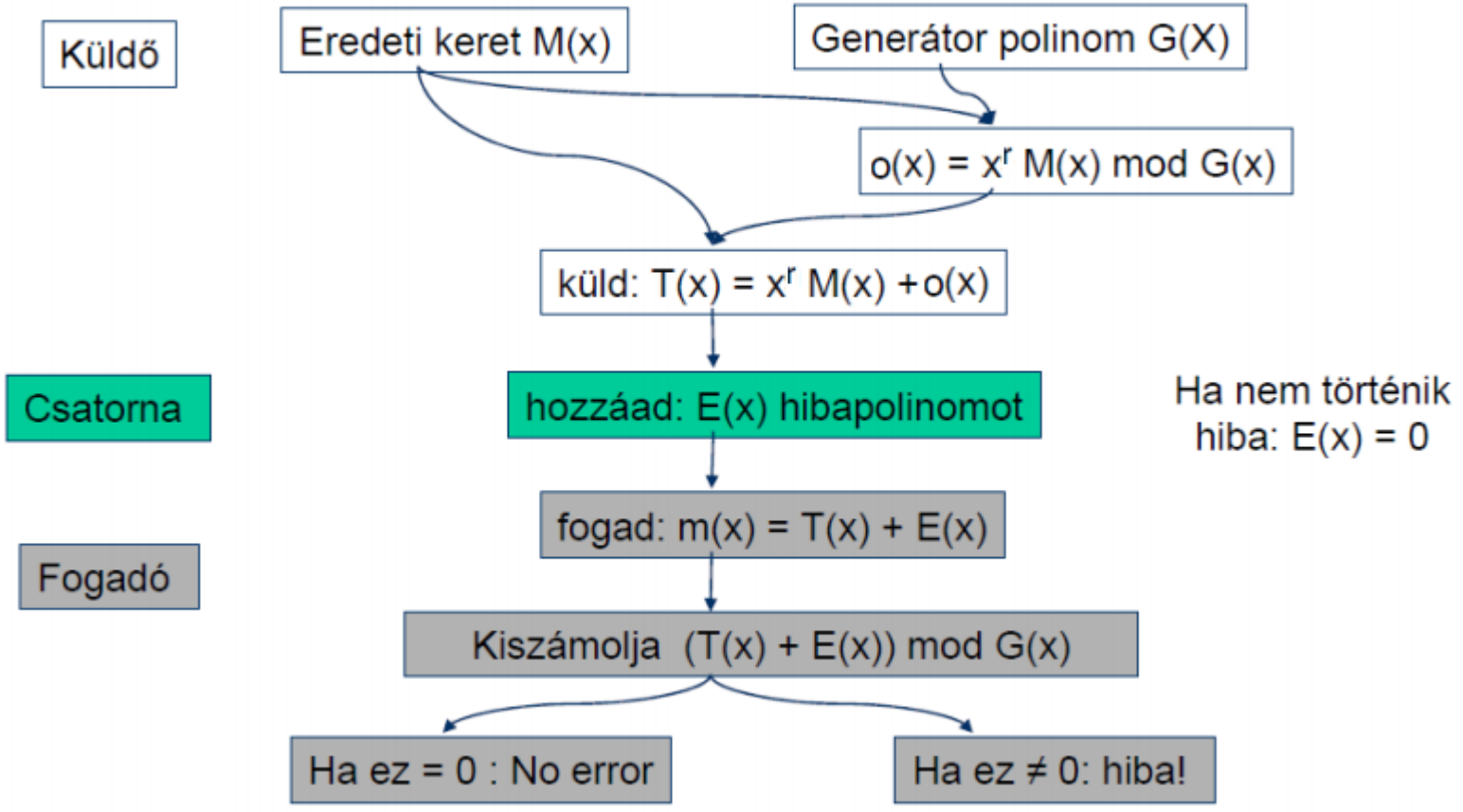
- „Négyzetes” elhelyezkedésre példa:

1	1	0	0	**1**		0	1	1	0	**1**
0	0	1	1	**1**		0	0	1	1	**1**
1	0	0	1	**1**		0	0	1	1	**1**
1	**0**	**0**	**1**	**1**		**1**	**0**	**0**	**1**	**1**

- Az új paritás-biteket tartalmazó oszloppal max. (2(*l+1*)-1 =) 9 hosszúságú löketszerű (burst-ös) hibát tudunk jelezni.

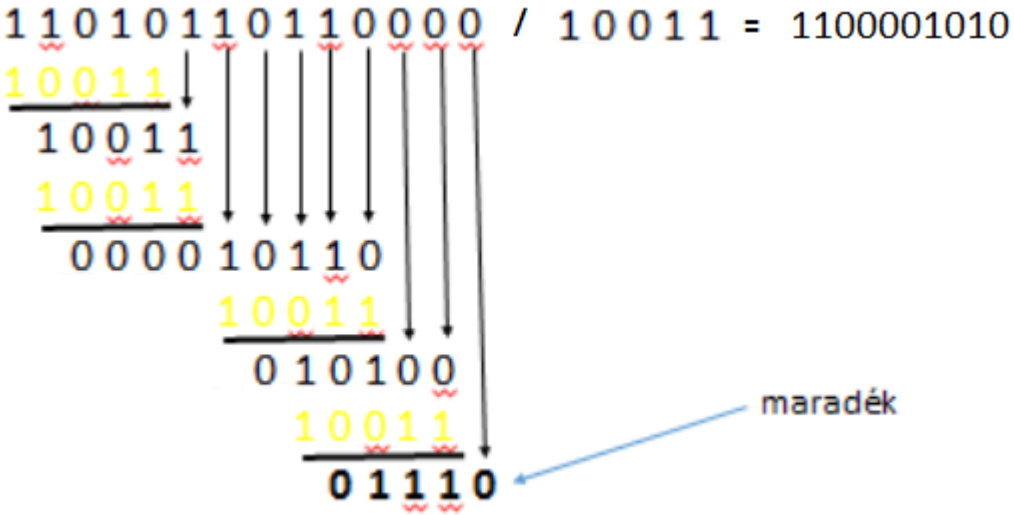
CRC hibajelző kód – emlékeztető

- Forrás: Dr. Lukovszki Tamás fóliái alapján



Példa CRC számításra – emlékeztető

- Keret ($M(x)$): 1101011011
- Generátor ($G(x)$): 10011
- Végezzük el a következő maradékos osztást: $\frac{11010110110000}{10011}$
- (A maradék lesz a CRC ellenőrzőösszeg)



Feladat 6

- Adva a $G(x) = x^4 + x^3 + x + 1$ generátor polinom.
- Számoljuk ki a $1100\ 1010\ 1110\ 1100$ bemenethez a 4-bit CRC ellenőrzőösszeget!
- A fenti üzenet az átvitel során sérül, a vevő adatkapcsolati rétege az $1100\ 1010\ 1101\ 1010\ 0100$ bitsorozatot kapja. Történt-e olyan hiba az átvitel során, amit a generátor polinommal fel lehet ismerni? Ha nem, akkor ennek mi lehet az oka?

Feladat 6 megoldása

- Mivel a generátor polinom foka 4, ezért négy 0-t írunk a bemenet végéhez. A $G(x)$ bináris alakban: 11011 lesz, tehát a $\frac{1100\ 1010\ 1110\ 1100\ 0000}{11011}$ maradékos osztást kell elvégeznünk:

```
11001010111011000000   /   11011
--------------------------------
11001010111011000000
11011
--------------------
 0010010111011000000
  11011
--------------------
 1001111011000000
 11011
--------------------
 100011011000000
 11011
--------------------
 10101011000000
 11011
--------------------
 1110011000000
 11011
--------------------
 011111000000
 11011
--------------------
 0100000000
 11011
--------------------
 10110000
 11011
--------------------
 1101000
 11011
--------------------
 000100
```

→ 0100 a CRC ellenőrzőösszeg

Feladat 6 megoldása

- Az előbbi számításnál az jött ki, hogy 1100 1010 1110 1100 0100 lenne az a bitsorozat, amelyet a vevő kapna. Ha ebből kivonjuk az alfeladatban megadott sorozatot, az alábbi eredmény jön ki:

$$\begin{array}{rr} & 11001010111011000100 \\ - & 11001010110110100100 \\ \hline & 00000000001101100000 \end{array}$$

- Tehát a két bitsorozat pontosan a generátor polinom többszörösével tér egymástól, amely tehát a hiba polinom. Ezt pedig nem lehet felismerni.

CRC, MD5, SHA1 pythonban

- CRC

```
import binascii, zlib
test_string= "Fekete retek rettenetes".encode('utf-8')
print(hex(binascii.crc32(bytearray(test_string))))
print(hex(zlib.crc32(test_string)))
```

- MD5

```
import hashlib
test_string= "Fekete retek rettenetes".encode('utf-8')
m = hashlib.md5()
m.update(test_string)
print(m.hexdigest())
```

- SHA1

```
import hashlib
test_string= "Fekete retek rettenetes".encode('utf-8')
m = hashlib.sha1()
m.update(test_string)
print(m.hexdigest())
```

Házi feladat
netcopy alkalmazás

Készítsen egy netcopy kliens/szerver alkalmazást, mely egy fájl átvitelét és az átvitt adat ellenőrzését teszi lehetővé CRC vagy MD5 ellenőrzőösszeg segítségével! A feladat során három komponenst/programot kell elkészíteni:

1. Checksum szerver: (fájl azonosító, checksum hossz, checksum, lejárat (mp-ben)) négyesek tárolását és lekérdezését teszi lehetővé. A protokoll részletei a következő oldalon.
2. Netcopy kliens: egy parancssori argumentumban megadott fájlt átküld a szervernek. Az átvitel során/végén kiszámol egy md5 checksumot a fájlra, majd ezt feltölti fájl azonosítóval együtt a Checksum szerverre. A lejárati idő 60 mp. A fájl azonosító egy egész szám, amit szintén parancssori argumentumban kell megadni.
3. Netcopy szerver: Vár, hogy egy kliens csatlakozzon. Csatlakozás után fogadja az átvitt bájtokat és azokat elhelyezi a parancssori argumentumban megadott fájlba. A végén lekéri a Checksum szervertől a fájl azonosítóhoz tartozó md5 checksumot és ellenőrzi az átvitt fájl helyességét, melynek eredményét stdoutputra is kiírja. A fájl azonosító itt is parancssori argumentum kell legyen.

Leadás: A program leadása a TMS rendszeren **.zip** formátumban, amiben egy **checksum_srv.py**, egy **netcopy_cli.py** és egy **netcopy_srv.py** szerepeljen!

Beadási határidő: **TMS rendszerben**

Checksum szerver - TCP

- Beszúr üzenet
  - Formátum: szöveges
  - Felépítése: BE|<fájl azon.>|<érvényesség másodpercben>|<checksum hossza bájtszámban>|<checksum bájtjai>
  - A „|” delimiter karakter
  - Példa: BE|1237671|60|12|abcdefabcdef
    - Ez esetben: a fájlazon: 1237671, 60mp az érvényességi idő, 12 bájt a checksum, abcdefabcdef maga a checksum
  - Válasz üzenet: OK
- Kivesz üzenet
  - Formátum: szöveges
  - Felépítése: KI|<fájl azon.>
  - A „|” delimiter karakter
  - Példa: KI|1237671
    - Azaz kérjük az 1237671 fájl azonosítóhoz tartozó checksum-ot
  - Válasz üzenet: <checksum hossza bájtszámban>|<checksum bájtjai>
    Péda: 12|abcdefabcdef
  - Ha nincs checksum, akkor ezt küldi: 0|
- Futtatás
  - python checksum_srv.py <ip> <port>
    - <ip> - pl. localhost a szerver címe bindolásnál
    - <port> - ezen a porton lesz elérhető
  - A szerver végtelen ciklusban fut és egyszerre több klienst is ki tud szolgálni. A kommunikáció TCP, csak a fenti üzeneteket kezeli.
  - Lejárat utáni checksumok törlődnek, de elég, ha csak a következő kérésnél ellenőrzi.

Netcopy kliens – **TCP alapú**

- Működés:
  - Csatlakozik a szerverhez, aminek a címét portját parancssori argumentumban kapja meg.
  - Fájl bájtjainak sorfolytonos átvitele a szervernek.
  - A Checksum szerverrel az ott leírt módon kommunikál.
  - A fájl átvitele és a checksum elhelyezése után bontja a kapcsolatot és terminál.
- Futtatás:
  - python netcopy_cli.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>
    - <fájl azon>: egész szám
    - <srv_ip> <srv_port>: a netcopy szerver elérhetősége
    - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége

Netcopy szerver – TCP alapú

- Működés:
  - Bindolja a socketet a parancssori argumentumban megadott címre.
  - Vár egy kliensre.
  - Ha acceptálta, akkor fogadja a fájl bájtjait sorfolytonosan és kiírja a parancssori argumentumban megadott fájlba.
  - Fájlvége jel olvasása esetén lezárja a kapcsolatot és utána ellenőrzi a fájlt a Checksum szerverrel.
  - A Checksum szerverrel az ott leírt módon kommunikál.
  - Hiba esetén a stdout-ra ki kell írni: CSUM CORRUPTED
  - Helyes átvitel esetén az stdout-ra ki kell írni: CSUM OK
  - Fájl fogadása és ellenőrzése után terminál a program.
- Futtatás:
  - python netcopy_srv.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>
    - <fájl azon>: egész szám ua. mint a kliensnél – ez alapján kéri le a szervertől a checksumot
    - <srv_ip> <srv_port>: a netcopy szerver elérhetősége – bindolásnál kell
    - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége
    - <fájlnév> : ide írja a kapott bájtokat

VÉGE
KÖSZÖNÖM A FIGYELMET!