Kódolás és IT biztonság
(VIHIBB01)
Laboratóriumi gyakorlat

Hibajavító kódolás Python nyelven

Szerző:
Holczer Tamás

2020. december 3.

Tartalomjegyzék

1. Mérés célja **2**

2. Háttér **2**
2.1. CRC 2
2.2. Reed-Solomon kódok 3

3. PyCRC csomag **4**

4. unireedsolomon csomag **4**

5. Topológia és kommunikáció **4**

6. Feladatok **6**
6.1. feladat (vezetett) 6
6.2. feladat 7
6.3. feladat 7
6.4. feladat 8

1. Mérés célja

A mérés folyamán széles körben használt hibadetektáló és hibajavító megoldásokkal fog egy zajos csatornát kezelni Python nyelven. A hibajavításhoz Cyclic Redundancy Check, röviden CRC kódokat kell felhasználni, amit a PyCRC csomag valósít meg Python nyelven. A hibajavítás Reed-Solomon kódokkal történik, amit a unireedsolomon csomagban érhető el. A mérés folyamán a zajos csatornát egy UDP szerver valósítja meg, ami tetszőleges hibavalószínűséggel tud hibákat okozni a rajta átfolyó kommunikációban. A mérés sikeres teljesítésével a hallgatók képesek lesznek hibadetektáló és javító kódokat alkalmazni, illetve a hálózatkezelési képességeik is fejlődnek.

2. Háttér

Hálózaton keresztül távoli felek is tudnak hatékonyan információt cserélni. A résztvevő eszközök és megoldások soha nem teljesen tökéletesek, kommunikációs hibák léphetnek fel a használatuk közben. Ezeket a hibákat a fogadó oldalon ki kell javítani, de legalábbis észre kell venni, hogy a kommunikáció megbízható legyen. Semelyik hibajavító vagy detektáló megoldás sem tökéletes, ezek egy adott mennyiségű hibát tudnak kezelni. A mérés folyamán CRC kódokat használunk a hibák detektálására és Reed-Solomon kódokat a hibák javítására.

Ebben a gyakorlatban használt megoldások nem támadások ellen lettek kifejlesztve, hanem véletlen csatornahibák kezelésére. Egy tudatos támadó minden további nélkül kiszámíthatja egy támadás után az új CRC-t vagy Reed-Solomon kódot, mivel ezek számításához semmilyen titkos kulcsra nincs szükség.

2.1. CRC

CRC (Cyclic Redundancy Check) kódok segítségével hibákat lehet detektálni például adatátvitel során. A CRC-vel számolt ellenőrzőösszeg az átvitt adat mellé van csatolva, amit a vevő oldal fel tud használni az ellenőrzés folyamán: A fogadó oldal kiszámolja ugyanazzal az algoritmussal az ellenőrzőösszeget, és ha az nem egyezik meg a kapott összeggel, akkor hiba történt az átvitel folyamán.

A CRC összeg kiszámítása osztáson alapul. A küldendő adatot egy hosszú

bináris vektorként ábrázolva azt el kell osztani egy másik bináris vektorral, ahol az osztó a CRC algoritmusban van meghatározva. Az osztás maradéka az átküldendő ellenőrzőösszeg.

A CRC kiszámítása során a bináris vektorok polinomokat ábrázolnak, ahol a bitek az együtthatók, és az osztás polinomosztást jelent.

Például az elterjedt CRC16 algoritmus a következő polinomot használja osztóként:

$$x^{16} + x^{12} + x^5 + 1$$

A polinom osztás utáni maradék együtthatói a CRC bitjei.

Az érdeklődő hallgatók a következő linkeken nézhetnek utána részletesebben a témának (nem feltétele a mérésnek): `http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html` vagy `https://en.wikipedia.org/wiki/Cyclic_redundancy_check`

2.2. Reed-Solomon kódok

A Reed-Solomon kódok adatblokkokon dolgoznak, amiket egy véges test elemiként (szimbólum) ábrázolnak. Például egy 4096 bájtos blokk (32.768 bit) 2731 darab 12 bites szimbólumként is kezelhető, ahol minden szimbólum egy tagja a $GF(2^{12})$ véges testnek (az utolsó 8 bitet ki kell egészíteni négy 0-ás bittel, hogy az is 12 bit hosszú legyen).

A Reed-Solomon kódok segítségével több hibát is lehet detektálni vagy akár javítani. t darab extra szimbólum hozzáadása árán legfeljebb t darab hibás szimbólum érzékelhető, vagy $\lfloor t/2 \rfloor$ szimbólum javítható. Ezen kívül t nem olvasható szimbólumot is ki tud javítani, illetve hibás és olvashatatlan szimbólumok keverékét is tudja kezelni. A kód legfontosabb paramétere az n maximális kódszóhossz és k üzenet hossz ($k + t = n$). A hibajavító szimbólumok olyan távol helyezik el a szimbólumokat a térben, hogy $\lfloor t/2 \rfloor$ hiba után is megkülönböztethetőek legyenek. A javítás a vett szimbólumhoz legközelebbi érvényes szimbólum megkeresésével valósul meg.

Az érdeklődő hallgatók a következő linkeken nézhetnek utána részletesebben a témának (nem feltétele a mérésnek): `https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html` vagy `https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction`.

3. PyCRC csomag

A PyCRC egy Python nyelven írt CRC implementáció, ami több szabványos megoldást is megvalósít, mint például CRC16 vagy CRC32. A telepítése pip segítségével történik.

```
pip install pythoncrc
```

A mérés előtt a csomag dokumentációját át kell tekinteni. A dokumentáció itt található:

`https://pycrc.readthedocs.io/en/latest/`

4. unireedsolomon csomag

Az inireedsolomon csomag egy Reed Solomon kódokon alapuló hibajavító csomag, ami a kódolást és dekódolást is teljes egészében Python nyelven valósítja meg. A telepítése pip segítségével történik.

```
pip install unireedsolomon
```

A mérés előtt a csomag dokumentációját át kell tekinteni. A dokumentáció itt található:

`https://pypi.org/project/unireedsolomon/`

Megjegyzés: a dokumentáció nem teljes, mivel a rendkívül hasznos `return_string` argumentum csak a kód megjegyzésében kerül ismertetésre.

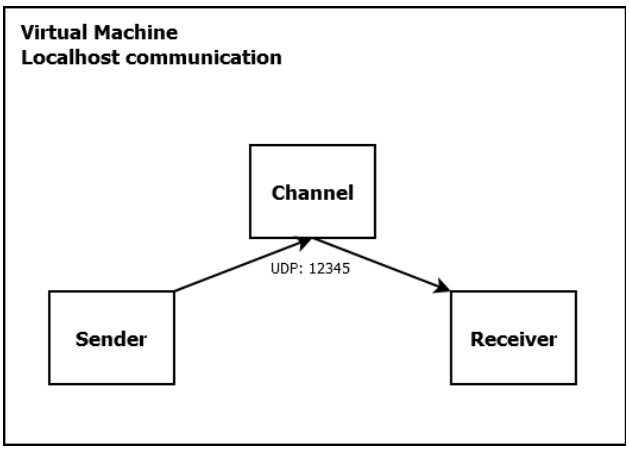
5. Topológia és kommunikáció

A mérés folyamán szabvány Python UDP kommunikáció segítségével küldik a csomagokat a felek. A kommunikáló felek lehetnek egy gépen, vagy hálózattal összekötött különböző gépeken is (a mérés folyamán egy gépen lesz mindenki a könnyebb lebonyolítás érdekében). A kommunikációban kétféle szerep fordul elő:

1. Csatorna: proxiként üzemel, üzeneteket továbbít a felek között. Továbbítás közben véletlen hibákat is tud okozni megfelelő beállítások esetén. A csatorna 2 vagy több klienst is össze tud kötni, de a mérés folyamán csak két kliens között fog kapcsolatot biztosítani.

2. Kliens: fogadni és küldeni tud csomagokat a csatornán keresztül.

A mérésen felhasznált topológia a 1. ábárn látható.



1. ábra. Mérés folyamán használt topológia

A mérés folyamán a következő üzenetformátumot használják a felek (az üzenetek feletti számok a mezők hosszát jelentik):

```
| ------ 8 ------ | ------- 8 ------- | - Var - | ---- 8 ---- | ----- 8 ----- | --------- Var ---------- |
| Type of message | Length of message | Message | Type of ECC | Length of ECC | Error correction code    |
| Üzenet típusa   | Üzenet hossza     | Üzenet  | ECC típusa  | ECC hossza    | Hibajavító/detektáló kód |
| --------------- | ----------------- | ------- | ----------- | ------------- | ------------------------ |
```

Az üzenet típusa a következő lehet:

```
TYPE_CONNECT = 0 (kapcsolódás)
TYPE_DISCONNECT = 1 (kapcsolat bontás)
TYPE_TRANSFER = 2 (átvitel)
```

A hibajavító/detektáló kód típusa a következő lehet:

```
ECC_TYPE_NONE = 0 (nincs)
ECC_TYPE_CRC16 = 1 (ECC16)
ECC_TYPE_RS5 = 2 (Reed-Solomon 5)
```

A többi konstans a `message_format.py` és `server_address.py` fájlokban megnézhető. A mérés előtt ezen fájlok áttekintése erősen javasolt.

6. Feladatok

A feladatok folyamán azt vizsgáljuk meg, hogy mi történik egy üzenettel, ha az egy zajos csatornán kerül továbbításra, és (i) nincs védve (ii) hiba detektálást alkalmazunk (iii) hiba javítást alkalmazunk.

6.1. feladat (vezetett)

A feladat folyamán megismerkedünk a környezettel, és megvizsgáljuk, hogy mi történhet egy zajos csatornán védelem nélkül átküldött üzenettel:

- készítsünk egy másolatot a `client_skeleton.py`-ról `client.py` néven
- indítsuk el a csatornát a következő paranccsal: `python channel.py 0.01` (a bithibavalószínűség legyen 0.01)
- a kliens ne használjon semmilyen védelmet még: `DEFAULT_ECC_TYPE = message_format.ECC_TYPE_NONE`
- indítsunk el egy klienst vevő módban: `python client.py` mód: `r`
- indítsunk el egy klienst adó módban: `python client.py` mód: `s`
- küldjünk pár üzenetet az adóból a vevőnek
- nézzük meg, hogy az adott és vett üzenetek megegyeznek e

Ha sikerült mindent jól futtatni, akkor nem keletkeznek kivételek a futás közben.

Beadandó: a vevő által kiírt utolsó sor miután vett egy üzenetet. A sor eleje így kezdődik: `No ...`

6.2. feladat

Ebben a feladatban egy CRC16 alapú hiba detektáló kódot kell elkészíteni a PyCRC segítségével:

- valósítson meg CRC16 funkcionalitást a `client.py` programban a hiányzó részek kiegészítésével.
- a client.py elején állítsa be, hogy a CRC16 legyen az alapértelmezett ellenőrző kód
- indítsa el a 3 programot, úgy hogy a kommunikáló felek CRC16-ot használjanak, a bemenet 20 darab 'a' karakter (`TEST_MESSAGE = 20*'a'`), míg a bithibaarány 0 legyen. Jó működés esetén a vevő oldal nem jelezhet hibát.
- állítsa le a 3 programot
- indítsa el a 3 programot, úgy hogy a kommunikáló felek CRC16-ot használjanak, a bemenet 20 darab 'a' karakter (`TEST_MESSAGE = 20*'a'`), míg a bithibaarány 0.01 legyen mind az üzenet mind a kód esetén.

Beadandó: A 20 darab 'a' karakterhez tartozó CRC érték számként (eleje: `26...`)

6.3. feladat

Ebben a feladatban az unireedsolomon csomag segítségével kell Reed-Solomon kódokat előállítani.

- fejezze be a Reed-Solomon kódot elkészítő részletet a `client.py`-ban, úgy hogy 5 hibát tudjon javítani a megoldás.
- indítsa el a 3 programot, úgy hogy az legfeljebb 5 hibát tudjon javítani (RS5), a bemenet 20 darab 'a' karakter legyen (`TEST_MESSAGE = 20*'a'`), és a bithibaarány 0 legyen.

Beadandó: A 20 darab 'a' karakter kiegészítve a hibajavító kóddal RS5 esetén. A byte tömb így kezdődik: `b'aaa`, és néhány hexa érték van a végén.

6.4. feladat

Ebben a feladatban a Reed-Solomon kódok hibajavító képességét kell megvizsgálni. Az előző feladatban elkészített kód segítségével végezzen mérést:

- Keresse meg azt a legkisebb bithibaarányt, ami esetén előfordul, hogy 10-ből 1 elrontott üzenetet nem tud javítani már az RS5 kód.

Beadandó: Bithibaarány két tizedes pontossággal számként (pl: 0.17).