11. fejezet – Hibakezelés, hibajavítás

Hibakezelés

Az adatfolyam eddig megismert keretekre bontása hasznos és szükséges, de nem elégséges feltétele az adatok hibamentes és megfelelő sorrendű átvitelének. Az adatfolyam keretekre (azaz kisebb egységekre) bontása leginkább abban segítség, hogy az átvitel bármely hibája esetén ne az egész adatfolyamot kellejen megismételni, hanem csak a hibás kereteket (azaz kisebb egységeket). A keretre bontás különböző módszereinek legfontosabb jellemzője, a kerethatárok azonosítása. Az adatkapcsolati rétegnek a keretekre bontáson (illetve vevő oldalon egyesítésen) túl meg kell tudnia győződni a keretek tartalmi megfelelősségéről és a keretek megfelelő sorrendiségéről is. A kapcsolat módja (nyugtázott/nyugtázatlan, összeköttetés alapú/összeköttetés nélküli) szerint ezek a feladatok különböző féleképpen oldhatók meg.

Nyugtázatlan összeköttetés esetén az adó folyamatosan küldi a kereteket, és nincs is lehetősége arra, hogy azok megérkezéséről, tartalmáról, sorrendjéről a vevő oldaltól tájékoztatást kapjon.

A nyugtázott összeköttetés az a kapcsolati mód, amikor „nyugtázás” azaz visszajelzés, visszacsatolás érkezik a vevő oldaltól minden egyes keret megérkezése után. A visszajelzés tartalma jelentheti a keret tartalmi és sorrendi megfelelősségét, illetve ezek hiányát, ami az adott keret ismételt elküldését fogja eredményezni az adó oldalon. Tekintettel arra, hogy az adatkapcsolati réteg alap adategysége a keret, maga a visszajelzés is gyakorlatilag egy keret, amely speciális vezérlőkaraktereket tartalmaz.

Természetesen az adatkapcsolati rétegnek arra az esetre is fel kell készülnie, amikor ez a nyugtaként szolgáló keret sérül meg, vagy veszik el. A gyakorlatban ez azt jelenti, hogy a nyugta beérkezésére az adó nem várhat az idők végezetéig (pláne ha a nyugta elveszett), hanem egy időzítő lejárta után – azt feltételezve, hogy ha a nyugta elveszett, akkor talán az a keret is elveszhetett vagy megsérülhetett, aminek a nyugtáját vártuk – az elveszett nyugtához tartozó keretet az adó újraküldi. Ebből a példából is nyilvánvalóan kiderül, hogy a kereteknek mindenképpen rendelkezniük kell egy sorszámmal. Ez a sorszám segít például abban is, hogy a vevő, a bármely okból is (például, ha csak a nyugta veszett el, pedig a küldött adatkeret megfelelően megérkezett) megismételt keretet a helyén tudja kezelni. Így el lehet azt kerülni, hogy a vevő oldalon visszaállított adatfolyamban egy (vagy több) keret duplán szerepeljen.

A fentiek alapján látszik, hogy az adatkapcsolati réteg a keretezési funkció kiegészítéseként időzítőket és számlálókat is kezel.

A hibakezelés szükségképpen redundanciát tartalmazó többlet adatátvitellel jár, így terheli az átviteli csatornát. Célszerű az átviteli csatorna hibaarányát és átviteli kapacitását is figyelembe venni az optimális hibakezelési algoritmus kiválasztásakor, hogy ne legyen feleslegesen sok az adatátviteli többlet.
A hibakezelő megoldásokat két részre oszthatjuk. Megkülönböztetjük a hibajelző kódokat (Error-detecting Codes) és a hibajavító (Error-correcting Codes) kódokat. Ez utóbbiakra egyes helyeken a FEC (Forward Error Correction / Előre Irányuló Hibajavítás) elnevezést is használják.
(Nem szabad arról sem megfeledkezni, hogy a hibajavítás szinte minden rétegben létezik, az adott réteg adategységének megfelelő módon – de legjellemzőbb előfordulása az adatkapcsolati réteg.)

Első ránézésre nyilván a hibajavító sokkal csábítóbbnak és hasznosabbnak tűnő megoldás, de a megfelelő megoldás kiválasztása ez esetben is főleg az átviteli csatorna megbízhatóságán múlik. Arról sem szabad elfeledkezni, hogy a hibajavítás ez esetben nem egy univerzális megoldás – ami minden potenciálisan bekövetkező hibát képes kijavítani, a javítás csak a lehetséges hibák egy részére korlátozódik. Azt se felejtsük el, hogy a hibajavítási algoritmus erőforrást von el a rendszertől.
Megbízható és gyors csatornák esetében, ahol hiba viszonylag ritkán fordul elő praktikusabb a hibás keretet újra küldeni, azaz elegendő a hibát jelezni a javítás helyett. Megbízhatatlan csatornák (pl. WLAN) estében célravezetőbb a hibajavító kódok használata. Könnyen elképzelhető ugyanis, hogy ha csak jelezzük a hibát és ismételjük a keretet, az újra és újra csak hibásan fog megérkezni – a csatorna jellegéből fakadóan és például egy időszakos külső zavarás miatt. Megfelelő hibajavító kód esetén a vevő képes a keret valós tartalmát helyreállítani.

Hibajavító kódok

A legszélesebb körben használt négy hibajavító kód a következő:

- Hamming-kódok
- Bináris konvolúciós kódok
- Reed-Solomon-kódok
- Alacsony sűrűségű paritásellenőrző kódok

A fenti négy hibajavító kód mindegyikére igaz, hogy az adó felsőbb rétegei által küldött adatokhoz további redundáns adatokat csatolnak, azaz a keret méretét megnövelik. Ha egy keretet a felépítése szerint vizsgálunk, akkor megállapíthatjuk, hogy „m" darab üzenetbitből (Message bit) és „r" darab redundáns, vagy ellenőrző bitből (Redundant/Check bit) állnak. Az pedig kódolás kérdése, hogy az „m" adatbithez melyik esetben mennyi „r" ellenőrző bitre van szükség.
Az adatátvitel szempontjából az „m" bitek és a kódolási algoritmussal előállított „r" bitek a közvetlenül, vagyis kódolás nélkül is belekerülhetnek a keretbe – ez esetben beszélünk szisztematikus kódról (Systematic Code). Amennyiben azonban a keretben már eleve az „m" és az „r" bitek valamely lineáris függvény szerinti megoldásában kerülnek bele – például az XOR (más néven: Kizáró Vagy; Antivalencia; Modulo2 összeadás) – ez esetben beszélünk lineáris kódolásról.

A hibakezelés első lépése az, hogy definiáljuk azt, hogy mit értünk hiba alatt. Hiba alatt például az adó oldali keretben található bitekhez képest a vevő oldali keretben található eltérő biteket értjük. Azaz a hiba legkisebb egysége a bit, viszont egyetlen bit egy egész bájtot (esetenként egy egész adatfolyamot) képes elrontani.
Hasonlítsuk össze a következő két nyolc bites adatot: 10001001 és 10110001
Meghatározhatjuk, hogy a két bájt hány egymásnak megfelelő bitje különbözik, tér el. Ebben az esetben 3 bit tér el (balról a harmadik, a negyedik és az ötödik). Ezt persze ránézésre meg tudtuk mondani, de hosszabb adatok esetében (illetve eleve gépi intelligencia esetében) célszerű matematikai módszerrel megkeresni a hibákat.

Adó oldal	10001001
Vevő oldal	10110001
XOR	00111000

Azon bitek számát, amiben két bitsorozat (esetünkben két bájt) eltér egymástól Hamming-távolságnak nevezzük. (Richard Hamming, 1950)

Hamming kódok

A Hamming-kódolás első lépése azon kódszavak definiálása, amelyek az adatátvitelben szerepelni fognak. A nullákból és egyesekből általunk kiválasztott bitsorozatokat (pl. 4 bites, 8 bites, 10 bites, stb.) nevezzük a kódszavaknak. Egy kódolást jellemző Hamming távolság az összes kódszó Hamming távolságai közül a legkisebb érték.

Vegyük észre: Ha olyan kódsorozat érkezik a vevő oldalra, ami eltér az előre definiált kódszavaktól, akkor hibás átvitel történt.

Például legyen adott egy nyolcbites kódrendszer, mely négy kódszóból áll. Keressük meg a minimális Hamming távolság „d" értékét, állapítsuk meg, hány hibát tud jelezni illetve hány hiba javítását teszi lehetővé a kódrendszer.

A kódszavak:

a) 00011111
b) 00101101
c) 01001011
d) 10000111

A Hamming távolságok:

H a/b= 3 H a/c= 3 H a/d= 3
H b/c= 4 H b/d= 4
H c/d= 4

A minimális Hamming távolság tehát „d=3". A felfedezhető hibák száma „d-1", azaz most 2 (kettő). A javítható hibák száma „d/2"-nél kisebb egész szám, mivel most „d/2=1.5", a javítható hibák száma tehát 1 (egy).

Ahhoz hogy „d" hibát ki tudjunk javítani „2d+1" Hamming-távolságú kódot kell alkalmazni, mert így az érvényes kódszavak olyan „távolságban" vannak egymástól, még „d" bit megváltozásakor is közelebb lesz az eredeti kódszó a hibáshoz, mint bármely másik érvényes kódszóhoz. Így egyértelműen el lehet dönteni, hogy mi lehetett az eredeti, a hibátlan kód – feltételezve, hogy nagyobb számú bithibának kisebb a valószínűsége.

A kódszavak kiválasztásakor figyelembe kell venni, hogy hiba, hibák esetén minél gyorsabban meg lehessen találni az eredeti kódszót. A Hamming-eljárás segítségével azt például a következő képpen tehetjük meg. A Hamming-kódokban a kódszavak bitjeit balról jobbra 1-el kezdve számozzuk meg. Azok a bitek, amelyek sorszáma 2 egészszámú hatványa (2^0=1, 2^1=2, 2^2=4, 2^3=8, stb.) lesznek a paritás bitek (ellenőrző bitek). A maradék helyeket pedig az üzenet bitjei foglalják el.

Az egyes paritásbitek nem az összes bitet ellenőrzik. Azon pozícióban lévő paritásbitek ellenőrzik a teljes kódszó „x”-edik bitjét, amely helyiértékeken az „x” kettes számrendszerbeli alakjában 1-es áll. Így például a kódszó 6. bitjét ellenőrzi a 4-es és a 2-es paritásbit, a többi viszont nem. Mert: $6 = 4 + 2 = 110_2$ (A négyes [2^2], valamint a kettes [2^1] helyiértéken áll 1-es, az egyes [2^0] helyiértéken 0-ás áll.)

Bináris konvolúciós kódok

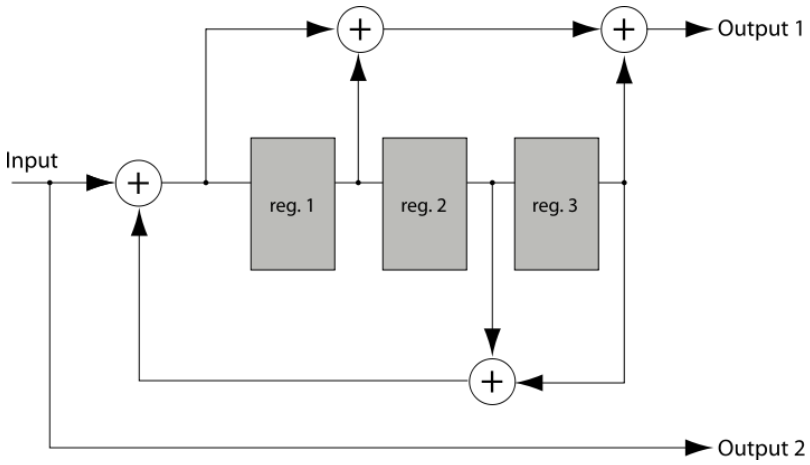
A bináris konvolúciós kódok jellegüket tekintve nem blokk-kódok, mivel a bemeneti (azaz kódolás előtti) bitsorozatból generálódik a kimeneti (azaz a kódolás utáni) kódsorozat, aminek esetileg változó a mérete, így üzeneti határokról sincs értelme beszélni. Az aktuálisan előállított kimeneti bit egyaránt függ a bemeneti bittől, és az előző kimeneti bitektől is – azaz összetett, kódoló memóriát kötelezően tartalmazó eljárásról van szó.

A blokk jellegű kódok előnye az egyszerűbb algebrai kezelhetőség, hátrányuk viszont a kód korlátozott hossza. A kódsebesség viszont a kódszavak hosszával egyre növekszik, így a konvolúciós kódok sebességelőnyre tesznek szert. Nem lehetne végtelen hosszú kódszavakat használni? A kérdésre a konvolúciós kód ad pozitív választ abban az értelemben, hogy elkezdhetünk kialakítani egy kódszót, amikor meg akarjuk kezdeni az információtovábbítást, és folyamatosan tarthat (növekedhet) a kódszó, ameddig van továbbítandó információnk. Ezáltal persze nem lesz végtelen, hiszen az adás egyszer véget ér, de nagyon nagy méretre tehet szert.

[Kódsebesség alatt a kódszó azon részét értjük, amely a nem redundáns információt tartalmazza. Azaz minél több a redundáns információ a keretben, annál lassabb lesz a kódsebesség.]

A konvolúciós kód klasszikus megvalósítása az, hogy vesszük a továbbítandó szimbólumsorozat „K” hosszú szegmensét, képezünk hozzá egy „N > K” hosszú sorozatot, mint a kódszó egy részét, majd vesszük a következő „K” adatszimbólumot, de ebből oly módon képezzük a kódszó következő „N” elemét, hogy tekintetbe vesszük a korábbi üzenetszimbólumokat is, vagy pontosabban azok véges és kötött hosszúságú szakaszát. Tehát miként egy lineáris invariáns transzformáció az időtartományban konvolúciót képez a bemenő jel és a súlyfüggvény között, létrehozva így a kimenő jelet, akként a konvolúciós kódoló is végigcsúsztatja egy speciális digitális szűrő súlyfüggvényét az adatszimbólumokon és előállítja a kódszimbólumokat. A specialitása abban rejlik, hogy több szimbólumot generál, mint amennyit a bemenetén feldolgoz.

[A konvolúció (ebben a megközelítésben) azt jelenti, hogy az egyes kimeneteink értékei több független (ez esetben nem közvetlenül függő) változón, több lépésben végrehajtott valamely matematikai logikai művelet eredményeképpen keletkeznek.]



A fenti ábrán a bináris konvolúciós kódolás egy lehetséges elvi megvalósítása látható. A modell 3db regiszterből, és 3db XOR művelet szerint működő összegzőből áll. Induláskor az összes regiszter tartalma 0, és két kimenetet sorrendben (Output1 majd Output2) kiolvasva 00 értéket kapunk. A regiszterek tartalma minden egyes bemeneti bit hatására egy lépéssel jobbra mozdul. Modellezzük le lépésenként, hogy mi történik, ha a bemenetre például a következő bitsorozat érkezik: 11100. A kimenetek értékei a következő képpen alakulnak: 11, 01, 11, 10, 00.
[Konvolúció.xls]
A konvolúciós kódolási rendszert először az 1977-ben indult Voyager küldetés során használta a NASA (ami a fenti ábrától eltérően nem 3db, hanem 6db regisztert tartalmazott, illetve egyik kimenet sem volt közvetlen fizikai kapcsolatban a bemenettel).
[Konvolúció.exe]
A konvolúciós kódolás ma már a mindennapos használatú, hiszen pl. a GSM szabványnak is része.

Reed-Solomon kódok

A Reed-Solomon-kódok, a Hamming-kódokhoz hasonlóan lineáris blokk kódok. A fő különbség az, hogy míg a Hamming-kódok különálló biteken működnek, a Reed-Solomon-kódok „m” bites szimbólumokon.
A Reed-Solomon-kódok arra a matematikai összefüggésre épülnek, hogy minden „n”-ed fokú polinomot egyértelműen meghatároz „n+1” darab pont.
Például egy egyenest (ami ax+b formában írható fel) 2 pont határoz meg. Ha például elküldjük üzenetben az egyenes 2 pontját, majd ez után az egyenes újabb két pontját, és az átvitel közben egy (1) hiba keletkezik, akkor van 3 olyan pontunk, ami biztosan az adott (akár az első két pontból meghatározható) egyenesen van. Ha ismertük (márpedig ismerjük) az egyenes egyenletét, akkor a hibás pont kiválasztható, sőt a hiba javítható – a pont ráhelyezhető az egyenesre.

A gyakorlati életben használt Reed-Solomon-kódok bonyolultabb polinomokkal, de hasonlóképpen működnek. Ha szimbólumok „m” bitesek, akkor a kódszavak 2^m-1 szimbólum hosszúak. Tehát „m=8” esetében a kódszavak 255 bájt nagyságúak. A 255 bájt (255,233) rendszerben kerül felosztásra, azaz 233 adatszimbólumból és 32 redundáns szimbólumból áll. Ez azt eredményezi, hogy akár 16 szimbólumnyi hibát is képes a rendszer kijavítani.
A Reed-Solomon-kódolás használatos a CD/DVD lemezek esetében is, ezért vagyunk képesek az alaposan összekarcolt lemezt is a legtöbb esetben hibátlanul lejátszani.

Alacsony Sűrűségű Paritásellenőrző kódok

Az LDPC (Low-Density Parity Check / Alacsony Sűrűségű Paritásellenőrző) kódok szintén a lineáris blokk kódok közé sorolandó. 1962-ben alkotta meg (doktori értekezésében) Robert Gallagher, de csak 1995-ben vált széles körben ismertté, amikor a technika lehetővé tette gyakorlati alkalmazását.
Az LDPC kód esetében minden egyes kimeneti bitet a bemeneti biteknek csak egy részéből képzik. A kód így egy mártix-szal jellemezhető, amiben viszonylag kevés 1-es és sok 0 található (innen származik az elnevezés is). A kódolás alacsony erőforrás igényű, nem túl komplex feladat, de a kód visszaállítása, a dekódolás roppant magas erőforrás igényű, iterációt igénylő komplex művelet. A kódolt üzenet minden esetben hosszabb az eredeti üzenetnél.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Generátormátrix

$$\underset{\text{Üzenet}}{\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}} \quad \underset{\text{Kódolt üzenet}}{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}}$$

Az LDPC kódolás főleg nagy blokkméret esetén hasznos, hiszen hibajavító képessége a többi kódolási módszert messze meghaladja. Erőforrás igényét a folyamatosan fejlődő hardware-ek egyre szélesebb körben képesek kiszolgálni – azaz mondhatjuk, hogy jelen ismereteink szerint ez a jövő kódolási technikája.
LDPC kódolás használatos például a DVB-T2 és a DVB-S2 műholdas adatátviteli rendszerekben.