Fakulteta za elektrotehniko
Univerza v Ljubljani

Matevž Kunaver

Linearne blokovne kode

seminarska naloga

Mentor: prof. dr. Sašo Tomažič

Ljubljana, maj 2005

Povzetek

V tej seminarski nalogi so opisane linearne blokovne kode, njihove lastnosti ter način uporabe.

Ker so linearne blokovne kode del kanalskega kodiranja, se v drugem poglavju najprej podrobneje opiše kanalsko kodiranje, postopki, ki se uporabljajo, ter pristopi k odpravljanju napak na tem nivoju.

V tretjem poglavju so opisani principi kodiranja z odpravljanjem napak ter opisane glavne prednosti in slabosti tega pristopa.

V četrtem poglavju se opisujejo linearne blokovne kode, njihove lastnosti, uporaba, realizacija ter prednosti in slabosti.

V petem poglavju je napisan kratek primer načrtovanja linearne blokovne kode.

V šestem poglavju pa so na hitro opisane linearne blokovne kode, ki so danes v rabi.

Kazalo

Kazalo slik

Kazalo tabel

1 Uvod

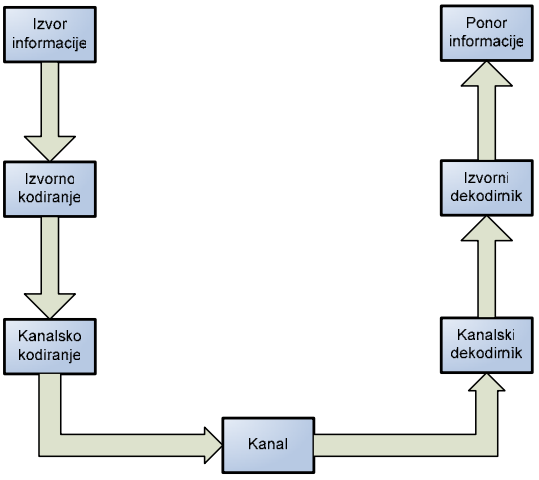
Pri prenašanju podatkov na daljavo preko različnih prenosnih poti se signal, ki te podatke prenaša, večkrat spremeni oziroma prekodira. Pri delu z digitalnimi podatki tako ni pomembna sama informacija, ki jo prenašamo, ampak sta sestavna dela digitalnega prenosa kodiranje in dekodiranje le-te. Za pripravo na prenos gre signal skozi vsaj dve fazi kodiranja.

V prvi fazi poteka izvorno kodiranje, s katerim iz analognih podatkov dobimo digitalne za nadaljnjo obdelavo. Izvorno kodiranje je lahko optimalno ali pa ne. V primeru optimalnega kodiranja nimamo opravka z izgubo informacije, vendar je zato stopnja kompresija manjša, kodirni postopek pa bolj zapleten. Če kodiranje ni optimalno, lahko dosežemo veliko večjo stopnjo kompresije, vendar imamo v zameno za to opravka z izgubo informacije. Le-ta se pokaže v obliki kvantnega šuma, ki se pojavi ko digitalni signal pretvorimo nazaj v analognega.

V drugi fazi poteka linijsko kodiranje, ki prilagodi frekvenčni spekter signala tako da ne vsebuje frekvenčnih komponent okoli 0 in da nastopajo stalne spremembe. Na ta način se prenos signala močno olajša, poleg tega pa se napake laže zaznajo.

V tretji fazi se z redundantnim kodiranjem signalu dodajo dodatni biti, s katerimi lahko sprejemnik zazna in odpravlja napake, do katerih je prišlo med prenosom zaradi šuma, interferenc in slabljenja. Redundantno in linijsko kodiranje se skupaj označuje tudi za kanalsko kodiranje, ker skupaj skrbita za čim boljšo prilagoditev signala na prenosni kanal, po katerem se bo prenašal. To tudi pomeni, da je potrebno za vsako prenosno pot signal na novo prekodirati, če želimo ohraniti zaželeno stopnjo kvalitete prenosa.

V zadnji fazi pa se lahko signal še zaščiti z uporabi kriptografskega kodiranje.



Slika 1: Kodiranje sporočila

Digitalni postopki postajajo dandanes vedno bolj priljubljeni, ker so integrirana vezja in procesorji digitalnih signalov vedno cenejša in zato bolj razširjena. Poleg tega se z uporabo primernih kodirnih postopkov lahko doseže do 10 dB izboljšanje pri razmerju med signalom in šumom. V tej seminarski nalogi se bomo osredotočili na del kanalskega kodiranja, ki se ukvarja z dodajanjem redundance osnovnemu signalu – linearnim blokovnim kodam.

2 Kanalsko kodiranje

Linearne blokovne kode spadajo med postopke, ki se uporabljajo kot del kanalskega kodiranja. Služijo torej zaznavanju in odpravljanju napak do katerih pride med prenosom. Pri samem kanalskem kodiranju prevladujeta dva različna principa oziroma dve kodiranji. Pri sami realizaciji prenosa se lahko uporabita obe, le eno ali nobeno od teh kodiranj. Poleg samega tipa kanalskega kodiranja pa je pomembna tudi metoda, s katero se odpravljajo napake, do katerih je prišlo med prenosom.

2.1 Tipi kanalskega kodiranja

2.1.1 Linijsko kodiranje

Linijsko kodiranje se ne ukvarja z detekcijo napake med prenosom, niti z odpravljanjem le-teh, temveč poizkuša optimizirati samo obliko prenašanega signala. S spreminjanjem oblike se tako poizkuša doseči neobčutljivost na zakasnitve in popačenja, do katerih lahko pride med prenosom. Tu se uporabljajo predvsem M-ary signalizacija, antipodalni in ortogonalni signali ter modulacija s Trellis-evim kodiranjem.

2.1.2 Strukturirane sekvence

V nasprotju z linijskim kodiranjem se pri strukturiranih sekvencah ne spreminja oblika signala, ampak spreminjamo samo kodiranje signala. Zaporedje bitov prenašanega signala se razdeli na posamezne bloke, nakar se ti bloki prilagodijo. Najbolj pogosto se posamezen blok spremeni v daljšo kodno besedo, s čimer se pravzaprav doda redundanca. V zameno za večje zahteve pri prenosu se pridobi sposobnost zaznavanja in odpravljanja napak na strani sprejemnika. Pomembno vlogo pri tem igra samo število dodanih bitov, vendar se pri tem naleti na realne meje, ker bi se drugače po kanalu prenašalo več bitov redundance kot sporočilnih bitov in izgubili bi veliko kapacitete prenosa.

2.2 Tehnike odpravljanja napak

Pri kanalskem kodiranju poznamo dve tehniki odpravljanja napak. To sta vnaprejšnje odpravljanje napak ter avtomatska zahteva po ponovitvi sporočilnega bloka.

2.2.1 Vnaprejšnje odpravljanje napak

Ta tehnika temelji na predpostavki, da sprejemna stran ne more komunicirati z oddajnikom, torej da samo sprejema prenesene podatke. Zaradi tega je potreben mehanizem, ki omogoča odpravljanje in zaznavanje napak kot samostojen proces na sprejemni strani.

Ta mehanizem se realizira z dodajanjem redundance, s pomočjo katere se lahko z računskimi postopki zazna, če je do napake prišlo in v določenih primerih to napako potem tudi odpravi. Vendar tehnika ne omogoča, da bi sprejemnik zaznal vse napake, do katerih pride. Sposobnost zaznavanja je močno omejena s količino dodatnih redundantnih bitov, ki jih lahko dodamo posameznemu bloku. Iz istega razloga je močno omejena tudi sposobnost samega odpravljanja nastalih napak. Vendar v zameno za te šibkosti tehnika še vedno omogoča zadosti zanesljiv prenos, ne postavlja nobenih dodatnih zahtev glede same strukture prenosnega kanala ter omogoča lažji hkratni prenos na več sprejemnikov.

Ker linearne blokovne kode uporabljajo to tehniko, bo več podrobnosti o njenih prednostih in slabostih napisanih v kasnejših poglavjih.

2.2.2 Avtomatična zahteva po ponovitvi sporočilnega bloka

Ta tehnika pri realizaciji zahteva, da ima sprejemnik sposobnost, da komunicira z oddajnikom. Zaradi tega oddajnik ne potrebuje sposobnosti odpravljanja napak, temveč napake samo zaznava. Ko napako zazna, to sporoči oddajniku, ki mu nato blok, v katerem je prišlo do napake, pošlje še enkrat. Glede na različne možnosti povezave med oddajnikom in sprejemnikom se je razvilo več različnih tehnik ponovnega pošiljanja sporočilnih blokov:

- **Stoj in čakaj** (Stop and wait) je pristop, pri katerem oddajnik po vsakem poslanem bloku čaka na potrditev sprejema na sprejemniku. Ta način sicer najslabše izkoristi prenosne zmogljivosti povezave, vendar se ga v zameno za to da uporabiti tudi v situaciji kjer ni možna hkratna obojestranska povezava med oddajnikom in sprejemnikom.
- **Neprekinjeno oddajanje s pomikom nazaj** (Continuous with pullback) je pristop, pri katerem oddajnik ves čas pošilja bloke sporočila. Na sprejemni strani jih sprejemnik prejema in preverja, če je prišlo do napake. Če napako zazna, pošlje oddajniku številko bloka, v katerem je prišlo do napake. Le-ta se nato pomakne nazaj do tega bloka in ponovno pošlje vse bloke od tam naprej. To je uporabno v primeru, ko sprejemnik ni sposoben nadomestiti samo pokvarjenega bloka, ampak mora celotno sekvenco še enkrat sestaviti. Ta pristop zahteva, da imata sprejemnik in oddajnik sposobnost hkratne komunikacije.
- **Neprekinjeno oddajanje s selektivnim ponavljanjem** (Continuous with selective repeat) je pristop podoben prejšnjemu. Tudi tu oddajnik ves čas oddaja bloke sporočila, sprejemnik pa jih sprejema in preverja napake. Vendar v primeru zaznane napake oddajnik ponovi samo tisti

blok, v katerem je prišlo do napake, nakar nadaljuje zaporedje od tam kjer je prej prekinil. To je še najhitrejši način odpravljanja napak, ker ne zahteva čakanja niti pretiranega ponavljanja sporočilnih blokov. Vendar v zameno za to zahteva bolj zmogljiv sprejemnik, ki mora imeti sposobnost nadomeščanja poljubnega sporočilnega bloka. Tudi ta pristop zahteva, da imata sprejemnik in oddajnik možnost hkratne komunikacije med seboj.

2.2.3 Primerjava tehnik za odpravljanje napak

Vsaka od obeh opisanih tehnik ima svoje prednosti in slabosti in se zato uporablja v različnih situacijah. Vnaprejšnje odpravljanje napak ima glavno prednost v tem, da ne postavlja dodatnih zahtev glede same povezave med oddajnikom in sprejemnikom, ker deluje vsak del popolnoma neodvisno od drugega. Vendar v zameno za to potrebuje zelo zmogljiv sprejemnik, ki ima sposobnost, da iz dodatnih bitov redundance ugotavlja, ali je med prenosom prišlo do napake, ter te napake potem računsko odpravi. Za zaznavo in odpravljanje napak je pri tej tehniki potrebno dodati veliko mero redundance, kar pomeni, da se zmanjša izkoristek zmogljivosti, ki jih nudi prenosna pot.

Pri tehniki avtomatičnega ponavljanja sporočilnih blokov je situacija ravno obratna. Tukaj ni zahtev po zelo zmogljivem sprejemniku, ker mora le-ta napake samo zaznavati ne pa tudi odpravljati. Sama prenosna pot je bolje izkoriščena, ker je za samo detekcijo napak potrebno dodati veliko manj redundantnih bitov na posamezen blok. Vendar v zameno za to tehnika postavlja določene zahteve glede samih lastnosti prenosne poti, ki mora zagotavljati obojestransko komunikacijo. Poleg tega v primeru zelo velikega števila napak sistem začne izgubljati svoje prednosti, ker vsaka napaka pomeni ponavljanje vsaj enega prenesenega bloka.

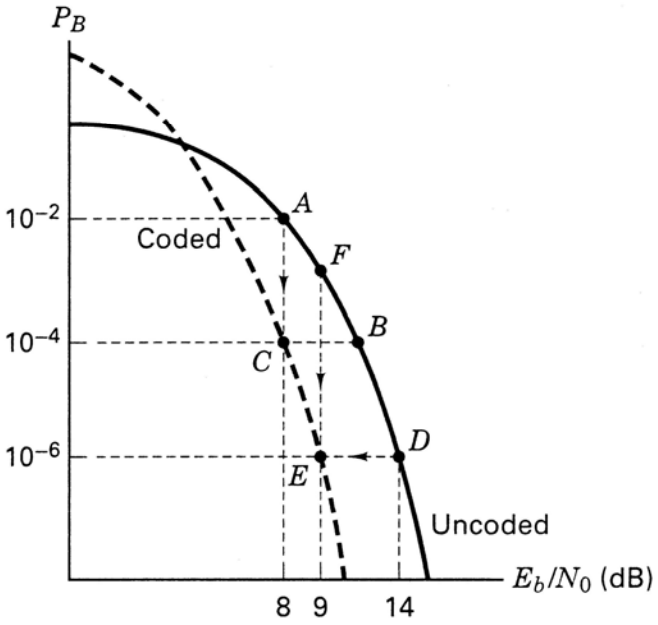
V praksi se izkaže, da sta oba pristopa močno razširjena, vendar se vnaprejšnje odpravljanje napak bolj uporablja v primerih, ko ni na voljo povratna povezava med sprejemnikom in oddajnikom ter ko pričakujemo večjo količino napak. Poleg tega v določenih situacijah ni zaželeno, da bi pretirano ponavljali posamezne bloke sporočila. Pomembno vprašanje je tudi ali gre za integracijo v že obstoječ sistem. V takem primeru je veliko lažje uporabiti vnaprejšnje odpravljanje napak, ker je ta rešitev bolj programerske narave. Po drugi strani bi uporaba ponavljanja lahko zahtevala, da se dogradijo dodatne prenosne poti in signalizacijski protokoli, kar ponavadi ni zaželeno.

3 Kodiranje z odpravljanjem napak

Pri kanalskem kodiranj želimo omogočiti zaznavanje napak ter odpravljanje leteh. Zato da bi to dosegli, lahko uporabljamo ponavljanje ali pa sporočilo razdelimo na posamezne bloke in jih prekodiramo. Toda tako kodiranje ima svoje prednosti in slabosti, zaradi česar je potrebno dobro premisliti kakšne parametre izbrati za sistem.

3.1 Prednosti in slabosti kodiranja

Kodiranje nam omogoči do 10dB izboljšanje pri razmerju signal / šum, vendar to za ceno določenih zmogljivosti prenosnih poti.



Slika 2: Izboljšanje razmer z uporabo kodiranja [1]

3.1.1 Odpravljanje napak

Pravilna blokovna koda nam omogoča, da na sprejemniku zaznamo napake, do katerih je prišlo med prenosom in jih nato popravimo. Če smo omejeni z razpoložljivim razmerjem med signalom in šumom, lahko z uvedbo kodiranja dosežemo do 100krat manjšo verjetnost napake. Vendar kodiranje zahteva, da v sporočilo dodamo redundanco, ki nam te funkcije sploh omogoči. Ker torej posamezen blok razširimo, to pomeni, da sicer ohranimo razmerje med signalom in šumom, vendar potrebujemo v zameno za to večjo pasovno širino, po kateri naše sporočilo prenašamo.

3.1.2 Moč signala

Druga možnost je, da želimo ohraniti trenutno verjetnost, da pride do napake med prenosom, vendar potrebujemo brez kodiranja zelo močno oddajno moč signala. Ker že vemo, da je s kodiranjem možno doseči manjšo verjetnost napake med prenosom, iz tega sledi da je možno tudi ohraniti enako verjetnost, vendar se zato potrebuje nižja moč signala. Cena za doseg tega pa ostaja enaka – zaradi dodajanja redundance potrebujemo večjo pasovno širino po kateri bomo naš signal prenašali.

3.1.3 Kodno ojačanje

Kot je bilo opisano v prejšnjih dveh podpoglavjih, nam kodiranje omogoča spreminjanje razmerja med signalom in šumom, oziroma izboljšanje razmer med prenosom, če tega razmerja ne spreminjamo. V primeru, ko razmerje prilagodimo, vendar ohranimo enako verjetnost napake med prenosom, lahko govorimo o kodnem ojačanju, ki nam predstavlja izboljšanje razmerja do katerega pride zaradi prekodiranja signala. To ojačanje izračunamo po enačbi (1).

$$G(dB) = \left(\frac{E_b}{N_0}\right)_u (dB) - \left(\frac{E_b}{N_0}\right)_c (dB) \qquad (1)$$

Pri tem nam $(E_b/N_0)_u$ predstavlja razmerje med signalom in šumom pred kodiranjem, $(E_b/N_0)_c$ pa razmerje po kodiranju.

3.1.4 Hitrost prenosa

V primeru ko pri nekodiranem signalu želimo doseči večjo hitrost prenosa podatkov, lahko le-to dosežemo samo z zmanjšanjem razmerja med signalom in šumom. To seveda pomeni slabšo kvaliteto prenosa in več napak. To je razvidno iz enačbe (2), kjer nam Pr/N0 predstavlja začetno razmerje med signalom in šumom, Eb/N0 končno razmerje, R pa nam predstavlja samo hitrost prenosa.

$$\frac{E_b}{N_0} = \frac{P_r}{N_0}\left(\frac{1}{R}\right) \qquad (2)$$

Ker kodiranje omogoča zmanjšanje verjetnosti napake to pomeni, da lahko v danem sistemu uvedemo kodiranje in zvečamo hitrost prenosa, ne da bi pri tem izgubili na sami kvaliteti signala. Kot v vseh prejšnjih primerih je potrebno za realizacijo takega sistema porabiti večjo pasovno širino pri oddajanju.

3.1.5 Kapaciteta prenosne poti

Tudi ta prednost temelji na dejstvu, da z uvedbo kodiranja potrebujemo manjše razmerje med signalom in šumom, da dosežemo želeno kakovost prenosa. Če pa potrebujemo manjše razmerje, to posledično pomeni, da potrebujemo za tak prenos tudi prenosno pot z manjšo kapaciteto. Če uporabljamo že obstoječe linije, nam to omogoči, da si več uporabnikov linijo deli brez opaznega poslabšanje kvalitete prenosa. Če povezavo šele načrtujemo, se lahko na ta način uporabi drugačne povezave, kot je bilo mišljeno na začetku.

Vendar to izboljšanje znova postavi zahtevo po večji pasovni širini, da do izboljšave pride.

3.1.6 Uporaba kodiranja pri slabem razmerju signal / šum

Obstaja spodnja meja razmerja med signalom in šumom, kjer kodiranje signala izgubi vse prednosti oziroma se celo začne obnašati slabše kot sam nekodiran signal. Do tega pride, ker je potrebno za vse zgoraj naštete izboljšave blokom signala dodajati dodatne bite, ki nam zagotovijo zadostno redundanco za zaznavanje napak.

Pri vedno slabšem razmerju med signalom in šumom bo na sprejemni strani sprejemnik zaznaval vedno več napak. Če se meja spusti še niže, bodo na sprejemniku samo še napake. Ker se je zaradi kodiranja samo število bitov povečalo, to pomeni, da bo v tem primeru pri slabem razmerju na sprejemniku število napak še dodatno naraslo. Torej bo vsak dodaten bit pomenil veliko večjo količino napak. Zaradi tega pojava ne moremo manjšati razmerja preko vseh meja, temveč je potrebno uporabiti razumno spodobno mejo, ki nam še vedno zagotavlja zadostno kvaliteto prenosa.

Če pa je res potrebno povezavo vzpostavljati pri zelo slabih razmerah, se za to uporabljajo posebne kode imenovano Turbo kode, ki so specializirane za take razmere.

3.1.7 Praktične omejitve

Kodiranje nam torej lahko pomaga v različnih situacijah. Lahko nam izboljša kvaliteto prenosa ali pa zagotovi enako kvaliteto v slabših razmerah. Omogoči nam da po istem kanalu komunicira več uporabnikov, ter da pri sami komunikaciji uporabljamo šibkejše signale. V splošnih primerih lahko tako z uvedbo kodiranja pričakujemo 50 do 100krat manjšo verjetnost napake med prenosom.

Vendar je potrebno za vse te izboljšave plačati z uporabo večje pasovne širine, kar v določenih situacijah ni zaželeno. Poleg tega se je izkazalo, da kode niso uporabne v vseh situacijah, predvsem v situacijah kjer imamo opravka z zelo slabim razmerjem med signalom in šumom.

Kljub vsemu so kodirne tehnike zelo močno orodje, ki nam omogoči večjo fleksibilnost pri delu s prenosnimi potmi. Zaradi tega postajajo tudi vedno bolj priljubljene in razširjene.

4 Linearne blokovne kode

Linearne blokovne kode torej temeljijo na principu dodajanje redundance za boljšo detekcijo in odpravljanje napak. Osnovna ideja tega kodiranja je, da sporočilo razdelimo na bloke velike k-bitov, ter te bloke nato prekodiramo v kodne besede sestavljene iz n-bitov. Na sprejemni strani te kodne besede nato preverimo, da ugotovimo ali smo sprejeli eno od 2^k možnih, ki nam predstavlja pravilno sprejet blok. Če pa je bila sprejeta katerakoli druga možna kodna beseda, to pomeni da je med prenosom prišlo do napake. V takem primeru mora sprejemnik nato ugotoviti pri katerem bitu je prišlo do napake, ter če mu to uspe, le-tega nadomestiti z njegovo pravilno vrednostjo.

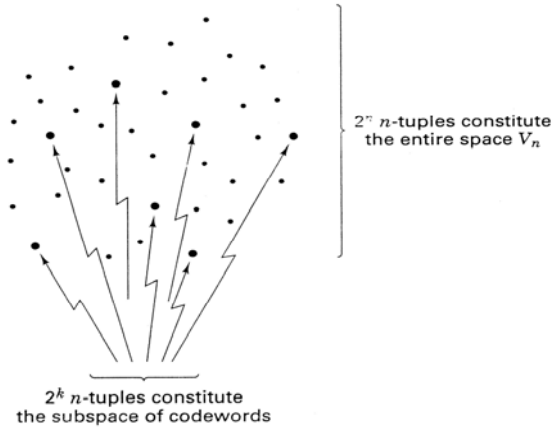
Samo pretvarjanje med posameznimi kodnimi besedami se izvršuje s pomočjo posebne tabele. Postopek pretvorbe ni kompleksen in je linearen, zato se tehnika tudi imenuje linearne blokovne kode.

4.1 Vektorski prostori in podprostori

Medtem ko imamo opravka z zaporedji bitov, ki nam predstavljajo posamezen blok sporočila, je bolj enostavno, če si ta zaporedja lahko predstavljamo kot vektorje v prostoru z k dimenzijami.

Pri linearnih kodah imamo tako opravka s preslikavami med prostori različnih dimenzij. Preslikavamo iz prostora z 2^k baznih vektorjev v prostor, ki ima 2^n baznih vektorjev. Ne glede na dimenzijo prostora v njem veljata dve osnovni operaciji – množenje in seštevanje po modulu 2. Ker v prostoru dimenzije n v katerega preslikavamo sporočilne bloke iz prvega prostora ne potrebujemo vseh možnih kod, je potrebno določiti pravila na podlagi katerih se nato izbere primerne vektorje.

Linearne kode so ponavadi podane kot **(n,k)**, kjer nam n predstavlja novo dimenzijo kodnih besed, k pa osnovno.



Slika 3: Vektorski prostor kodnih besed [1]

4.1.1 Določanje vektorjev podprostora

Ker preslikavamo iz manjšega prostora v večjega, obstaja $\binom{n}{k}$ različnih možnih načinov, katere vektorje izberemo, da nam predstavljajo osnovni nabor. Za linearne blokovne kode pri tej izbiri veljata dve pravili:

- Vektor samih ničel mora vedno biti vsebovan,
- Veljati mora, da je vsota dveh vektorjev enaka poljubnemu vektorju iz podprostora. Torej če vektorja $\mathbf{V_1}$ in $\mathbf{V_2}$ pripadata podprostoru $\mathbf{S_1}$, mora tudi njuna vsota $\mathbf{V_1 + V_2}$ pripadati temu prostoru.

Obe pravili nam predvsem pomagata pri zaznavanju napak. Vektor samih ničel se praviloma uporablja pri identifikaciji možnih skupin napak, do katetrih lahko pride med prenosom.

Pri sami izbiri sta pomembni še dve načeli, ki služita k čim boljšem izkoristku naše prenosne linije. Želimo namreč, da se večji prostor čim bolj zapolni ter nam na ta način ne postavlja pretirano velikih zahtev po dodatni pasovni širini. Poleg tega želimo doseči, da so si vektorji, ki nam predstavljajo naše bloke, med seboj čim bolj oddaljeni, ker so tako manj občutljivi na šum med prenosom. Če tako pride med prenosom do napake in sprejmemo popačen vektor, ga lahko sprejemnik še vedno pravilno dekodira, če le-ta ni preveč oddaljen od izvorne točke.

4.2 Računanje kodnih besed

Sedaj vemo, da iz nabora 2^k možni bitnih kombinacij naš sistem razširimo na 2^n kombinacij. To predstavlja tudi večjo računsko zahtevnost. Ker se sama pretvorba vrši z uporabo posebne tabele, se pojavi tudi težava s pomnilnikom v primeru ko je **n** veliko večji od **k**. Zato je bilo potrebno razviti metodo, s pomočjo katere se lahko z uporabo posebnega računa tvori potrebne kodne besede, ne da bi za to potrebovali celotno pretvorno tabelo.

Najprej definiramo podprostor **S**, ki vsebuje 2^k baznih vektorjev, od katerih je vsak dolg n bitov. Sedaj lahko vsakega od osnovnih blokov sporočila predstavimo kot vsoto baznih vektorjev po enačbi (3). U nam predstavlja ustrezno kodno besedo iz novega vektorskega podprostora. V nam predstavlja posamezen vektor tega prostora, m pa so biti bloka sporočila, ki ga kodiramo.

$$U = m_1\mathbf{V}_1 + m_2\mathbf{V}_2 + \cdots + m_k\mathbf{V}_k \qquad (3)$$

Sedaj lahko definiramo generatorsko matriko, ki sistemu omogoča da direktno izračuna pripadajočo kodno besedo. Zapišemo jo po enačbi (4).

$$G = \begin{bmatrix} V_1 \\ V_2 \\ : \\ V_k \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdot\cdot & v_{1n} \\ v_{21} & v_{22} & \cdot\cdot & v_{2n} \\ : & & & : \\ v_{k1} & v_{k2} & \cdot\cdot & v_{kn} \end{bmatrix} \quad (4)$$

Sedaj lahko posamezno kodno besedo direktno izračunamo kot **U = mG**. Tako je rešen problem kodiranja sporočilnih blokov na oddajni strani. Sporočilo sedaj samo razbijemo na ustrezno velike bloke in vsak blok posebej zmnožimo z generatorsko matriko. Rezultat te operacije je ustrezna kodna beseda, ki jo lahko sedaj pošljemo naprej.

4.2.1 Sistematske linearne blokovne kode

Z generatorsko matriko lahko dosežemo še dodatne prednosti. Lahko jo sestavimo na tak način, da izračuna tudi potrebne redundantne bite oziroma pariteten bite s katerimi bomo kasneje na sprejemni strani preverjali uspešnost prenosa. Kodam, ki uporabljajo tako matriko, rečemo sistematske linearne blokovne kode. Tu se generatorska matrika sestavi po enačbi (5). P nam predstavlja tisti del matrike, ki se uporablja za izračun paritete, I pa služi sami generaciji kodnih besed.

$$G = \begin{bmatrix} P & | & I_k \end{bmatrix} \quad (5)$$

Kodni vektor je sedaj izražen po enačbi (6).

$$U = p_1, p_2, ..., p_{n-k}, m_1, m_2, ..., m_k \quad (6)$$

Tu nam p predstavlja paritetne bite, m pa so biti originalnega sporočila. Število paritetnih bitov je povezano z dimenzijo začetnega in končnega prostora, torej za razliko med n in k.

4.3 Preverjanje paritete in odpravljanje napak

Potem ko smo na oddajni strani sporočilo razdelili na bloke, posamezne bloke prekodirali in jim dodali redundanco moramo sedaj postopek na sprejemni strani ponoviti v obratnem vrstnem redu. Tudi tu naletimo na težavo s pomnilnikom, zato je bilo potrebno razviti računsko metod za direktno dekodiranje, brez potrebe po hranjenju prevajalne tabele v spominu.

V prvi fazi je tako potrebno preveriti, če je med prenosom prišlo do napake in to napako, če se le da tudi popraviti.

4.3.1 Matrika za preverjanje paritete

Za preverjanje paritete se sestavi posebna matrika, ki jo imenujemo matrika za preverjanje paritete (parity-check matrix). Ker smo na oddajni strani

uporabili matriko za generiranje kodnih besed z dodano pariteto, lahko trdimo da obstaja tudi inverzna matrika, ki nam omogoči da to pariteto preverimo. To matriko zapišemo po enačbi (7). Za računanje pa uporabimo transponirano matriko H^T.

$$H = \left[I_{n-k} \quad | \quad P^T \right] \tag{7}$$

Velja tudi da je zmnožek poljubne kodne besede U in matrike H^T enak 0, torej $\mathbf{UH^T = 0}$. Na sprejemni strani tako sprejeti vektor **r** zmnožimo z $\mathbf{H^T}$ in preverimo rezultat. Če med prenosom ni prišlo do napake, bo rezultat enak nič in naš sprejeti vektor je enak eni od pravilnih kodnih besed. Če je rezultat različen od nič to pomeni, da je prišlo do napake ki jo moramo popraviti.

4.3.2 Sindrom napake

Sindrom nam predstavlja rezultat množenja sprejetega vektorja z matriko za preverjanje paritete. S pomočjo sindromov lažje identificiramo do kakšne napake je prišlo in jo tako tudi hitreje odpravimo. Sindrom tako izračunamo po enačbi (8)

$$S = r\mathbf{H}^T \tag{8}$$

Sprejeti blok nam predstavlja vsoto naše osnovne kodne besede in motenj **e** na kanalu, torej **r = U + e**. Ker pa vemo, da je zmnožek kodne besede in matrike H^T enak 0, iz tega sledi, da je sindrom neposreden odraz napake do katere je prišlo oziroma je enak 0, če do napake ni prišlo, kot je razvidno iz enačbe (9)

$$r = U + e, \quad S = (U + e)H^T = UH^T + eH^T = eH^T \tag{9}$$

Če pa želimo imeti še sposobnost, da zaznane napake popravimo, mora porazdelitev paritetnih bitov v matrikah **G** in **H** zadostiti dvema pogojema:

- V nobenem stolpcu matrike ne smejo nastopati same ničle, ker drugače na pripadajočem bitu nismo sposobni zaznati napake
- Vsak stolpec mora biti edinstven, ker drugače ne bi mogli ugotoviti na katerem bitu, ki mu pripadata enaka stolpca je prišlo do napake.

Če pa tem zahtevam zadostimo dobi sprejemnik sposobnost da odpravi nastale napake.

4.3.3 Odpravljanje napak

Sedaj imamo sposobnost prepoznavanja vsake napake posebej, ker vsaki napaki pripada njen lastni sindrom. Torej lahko za vsako kodno besedo ugotovimo ali je prišlo do znane napake in jo nato popravimo. Za bolj enostaven pregled se sistem kodnih besed, možnih napak in pripadajočih sindromov zapiše posebno matriko (Standard Array). V prvi vrstici te matrike so zapisane vse možne kodne besede $\mathbf{U_i}$, v prvem stolpcu vse možne napake

$\mathbf{e_i}$, ostale vrednosti pa predstavljajo vsoto kodne besede in ene od napak $\mathbf{U_i + e_j}$. Zgradba te matrike je razvidna iz (10)

$$\begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 & \dots & \mathbf{U}_i & \dots & \mathbf{U}_{2^k} \\ \mathbf{e}_2 & \mathbf{U}_2+\mathbf{e}_2 & \dots & \mathbf{U}_i+\mathbf{e}_2 & \dots & \mathbf{U}_{2^k}+\mathbf{e}_2 \\ \mathbf{e}_3 & \mathbf{U}_2+\mathbf{e}_3 & \dots & \mathbf{U}_i+\mathbf{e}_3 & \dots & \mathbf{U}_{2^k}+\mathbf{e}_3 \\ \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{e}_j & \mathbf{U}_2+\mathbf{e}_j & \dots & \mathbf{U}_i+\mathbf{e}_j & \dots & \mathbf{U}_{2^k}+\mathbf{e}_j \\ \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{e}_{2^{n-k}} & \mathbf{U}_2+\mathbf{e}_{2^{n-k}} & \dots & \mathbf{U}_i+\mathbf{e}_{2^{n-k}} & \dots & \mathbf{U}_{2^k}+\mathbf{e}_{2^{n-k}} \end{bmatrix} \quad (10)$$

Iz te zgradbe je razvidno, da lahko prepoznamo 2^{n-k} različnih napak do katerih lahko pride med prenosom. Torej nam večji **n** omogoči več zaznanih in odpravljenih napak. Vsak stolpec nam predstavlja set (coset), prva vrednost stolpca pa je vodilna vrednost stolpca (coset leader).

Kodna beseda, sestavljena iz samih ničel, pa tu služi dvema namenoma. Lahko jo uporabimo kot samo kodno besedo, ki nam predstavlja smiseln blok originalnega sporočila, ali pa jo lahko uporabimo za prepoznavanje samih napak, ker vsi spremenjeni biti te kodne besede predstavljajo spremembe, ki so nastale zaradi napake.

Če dekodirni algoritem sedaj zazna napako, najprej izračuna njen sindrom. Nato preveri ali se sindrom sprejete napake nahaja v standardni matriki. Če to drži, se lahko iz te matrike odčita pravilna vrednost sprejetega vektorja **r**. Najprej se izračuna sindrom ter se prestavi v pravilno vrstico te matrike. Potem pa se po tej vrstici poišče vrednost, ki je enaka sprejetemu vektorju **r.** Ko se ta vrednost najde, sistem ve, da je pravilna vrednost sprejetega vektorja enaka vrednosti v prvi vrstici stolpca, kjer se nahaja **r.**

Če tako pri sprejemu ni prišlo do take spremembe, da bi sprejeta vrednost sovpadala z eno od možnih kodnih besed (vektor se toliko spremeni da izpade kot druga beseda) in je prišlo do napake ki jo sistem pozna, jo bo zaznal in pravilno odpravil. Z uporabo sindroma pa se samo iskanje in popravljanje še veliko hitreje izvaja in postane uporabno za večji nabor možnih povezav.

Celoten postopek zaznavanja in popravljanja napake poteka v naslednjem vrstnem redu:

- Izračunamo sindrom **S** sprejetega vektorja **r.**
- Če je **S** različen od nič, to pomeni da je prišlo do napake. V standardni matriki se poišče pripadajoči vzorec iz prvega stolpca.
- Ko ga najdemo, predvidevamo da je to napaka $\mathbf{e_i}$**,** do katere je prišlo.
- V vrstici, ki pripada tej napaki, poiščemo vrednost, ki je enaka **r.**
- Vodilna vrednost tega seta je nato pravilna sprejeta vrednost **r.**

4.4 Zmogljivost kodnega postopka

Zaradi realnih zmogljivosti sistemov, kjer se to kodiranje uporablja ter zaradi nesmiselnosti dodajanja redundance preko vseh meja, ni možno sistem postaviti tako, da bi bil sposoben zaznati vse možne napake do katerih lahko pride. Ker lahko sistem zazna samo 2^{n-k} različnih napak, bi lahko število zaznanih napak povečali z večanjem **n**, vendar bi na ta način slabšali izkoristek prenosne poti, ker bi začeli prenašati veliko več bitov redundance za izračun paritete kot pa samih bitov sporočila. Pri določanju zmogljivosti kode si pomagamo z več računskimi orodji kot so uteži med vektorji in razdalje linearne kode.

Drugo orodje, s katerim si lahko pomagamo, pa je **Hammingova meja**. S to mejo lahko točno izračunamo, koliko različnih kombinacij napak je naša blokovna koda sposobna pravilno popraviti. Hammingovo mejo lahko izračunamo glede na število paritetnih bitov po enačbi (11) ali pa glede na število setov po enačbi (12). Spremenljivka **t** nam v teh enačbah predstavlja število pokvarjenih bitov med prenosom.

$$n-k \geq \log_2\left[1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}\right] \tag{11}$$

$$2^{n-k} \geq \left[1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}\right] \tag{12}$$

Če to neenačbo rešimo, torej poiščemo največji **t** pri katerem je enačbe še rešljiva, bomo ugotovili, katere vzorce napak bo koda vedno sposobna popraviti. Vzorce, ki bodo vsebovali več kot **t** pokvarjenih bitov, pa bo sistem zaznal, a vseh ne bo sposoben odpraviti.

Zadnje orodje, ki nam pomaga pri določanju sposobnosti kode, pa je **Plotkinova meja**. Ta meja se podobno kot Hammingova nanaša na sposobnost odpravljanja napak. Ker se Hammingova meja ne nanaša direktno na minimalno razdaljo, se lahko zgodi, da se izbere tak **n**, da bi teoretično bilo možno odpraviti vse napake z **t** ali manj pokvarjenimi biti, a se to v praksi ne bi zgodilo, ker ne bi bila dosežena minimalna razdalja $\mathbf{d_{min}}$. Zato je potrebno izračunati še Plotkinovo mejo, ki nam poda kakšen **n** zares potrebujemo. Izračunamo jo po enačbi

$$d_{\min} \leq \frac{n \times 2^{k-1}}{2^k - 1} \tag{13}$$

V praksi se izkaže, da koda ki ima dovolj velik **n**, hkrati zadosti obema enačbama, če pa je **n** manjši je potrebno sistem prilagoditi.

4.5 Uteži in razdalja med binarnimi vektorji

Čeprav se lahko linearne blokovne kode uporabljajo v sistemu, ki uporablja simbole, je še vedno najbolj razširjen način komunikacije binaren. Zato v tem poglavju opišemo, kako si pomagamo pri določanju sposobnosti kode v takem okolju.

Sposobnost odpravljanja napak je zelo odvisna od Hammingove razdalje. Ta razdalja temelji na podatku, koliko se dva vektorja razlikujeta med seboj. Za vsak bit, v katerem se razlikujeta, razdaljo povečamo za ena. Če bi na primer imeli $\mathbf{V_1 = 100101101}$ in $\mathbf{V_2 = 011110100}$, bi bila Hammingova razdalja med njima enaka 6.

Ker vemo, da v sistemu linearnih blokovnih kod velja, da je vsota dveh vektorjev enaka tretjemu iz istega podprostora, ugotovimo, da ima ta vektor enice točno na tistih mestih, kjer se začetna vektorja razlikujeta med seboj. Če izračunamo vsoto teh enic (kar sovpada z Hammingovo razdaljo), torej dobimo merilo, kako različna sta si ta dva vektorja. Zato lahko definiramo tudi utež binarnega vektorja, ki je merilo za raznolikost kodnih besed. Izračunamo pa jo po enačbi (14).

$$d(U,V) = w(U+V) \tag{14}$$

Utež med dvema vektorjema je torej enaka razdalji med njima. Če tako izračunamo vse možne razdalje med vektorji izbranega podprostora, dobimo merilo za uspešnost kodnega postopka pri zaznavanju napak. Večje uteži namreč pomenijo, da se vektorji bolj razlikujejo, zato sistem laže določi kje je prišlo do napake. Če pa imamo opravka z zelo majhnimi utežmi, se lahko zgodi da se sistem zmoti, ko poizkuša ugotoviti kam spada sprejeti signal **r**.

Za primer – če je med dvema vektorjema razdalja **w=1**, to pomeni, da bo sistem ob spremembi enega samega bita lahko sporočil, da smo na sprejemni strani sprejeli drugi vektor, čeprav smo v resnici pošiljali prvega.

4.6 Minimalna razdalja linearne kode

Razdalja med posameznimi vektorji ima torej velik pomen pri zmogljivostih linearne kode. Ker nam najšibkejši člen predstavlja najmanjša obstoječa razdalja, je potrebno razložiti kaj ta razdalja predstavlja oziroma kašne so njene zaželene vrednosti.

Najmanjša razdalja nam predstavlja zgornjo mejo napake, ki jo je sistem še pripravljen prenesti. Če je vsota osnovne kodna besede in napake, do katere pride med prenosom, manjša od minimalne razdalje ($\mathbf{U_i+e_j < d_{min}}$), bo sistem še pravilno ugotovil, da gre za kodno besedo, ki smo jo zares poslali. Če pa bo vsota večja ($\mathbf{U_i+e_j > d_{min}}$), bo sistem napačno interpretiral sprejeti vektor in bo na izhod dal neko drugo kodno besedo.

Minimalno razdaljo sistema $\mathbf{d_{min}}$ izračunamo tako, da izračunamo uteži vseh kodnih besed, ki smo jih izbrali za naš podprostor. Najmanjša od teh uteži

nam nato predstavlja tudi minimalno razdaljo tega sistema. Če sistem načrtujemo, nam ta razdalja omeji izbiro kodnih besed, ker vemo da se morajo posamezne besede med seboj razlikovati za vsaj $\mathbf{d_{min}}$ bitov.

4.7 Dekodiranje sprejetih blokov

Glavna naloga dekodirnika je, da se po sprejemu vhodnega vektorja **r** odloči katero kodno besedo **U** je zares sprejel in le-to posreduje naprej. Po primerjanju vektorja s posameznimi besedami se odloči za besedo $\mathbf{U_i}$ če zanjo velja enačba (15). Oziroma drugače – če je za dano besedo najbolj verjetno da ji pripada sprejeti vektor.

$$r \in U_i \quad \check{c}e \quad P(r \mid U_i) = \max P(r \mid U_j) \quad za \quad vse \quad U_i \tag{15}$$

Proces odločanja je lahko zasnovan tudi na primerjanju razdalj med vektorji, kot je nakazano v enačbi (16), le da tu iščemo najmanjšo razdaljo torej besedo ki je **r** najbolj podobna.

$$r \in U_i \quad \check{c}e \quad d(r \mid U_i) = \min d(r \mid U_j) \tag{16}$$

Minimalna razdalja je tako postala merilo za število bitov, ki se lahko spremeni, a bo sistem kljub temu pravilno sprejel kodno besedo. Če se spremeni več kot $\mathbf{d_{min}/2}$ bitov pa bo sistem besedo napačno dekodiral. Meja je postavljena na $\mathbf{d_{min}/2}$, ker nam razdalja predstavlja celotno pot med dvema besedama, torej je potrebno odločitev postaviti na pol poti med njima. Sistem, ki ima $\mathbf{d_{min}=7}$ bo tako lahko pravilno zaznal besedo tudi če se bodo v njej spremenili do trije biti.

Za lažjo predstavo je zato definirana sposobnost odpravljanja napak **t**, ki jo izračunamo po enačbi (17). Vrednost **t** nam tako direktno poda največje dovoljeno število spremenjenih bitov, če še želimo imeti pravilno dekodirano kodno besedo.

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{17}$$

Splošno smo ugotovili, da je (n,k) linearna blokovna koda sposobna odpraviti 2^{n-k} napak. Verjetnost, da pride do napake pri kodiranju, je tako izražena v enačbi (18). Pri tem nam *p* predstavlja verjetnost da je med prenosom prišlo do napake.

$$P_M \le \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j} \tag{18}$$

Če pa želimo izračunati verjetnost za posamezen bit, uporabimo enačbo (19).

$$P_B \approx \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j} \qquad (19)$$

Poleg samega odpravljanja napak je pomemben tudi podatek koliko napak je sistem sploh sposoben zaznati. Koda z minimalno razdaljo $\mathbf{d_{min}}$ je tako zanesljivo sposobna zaznati največ $\mathbf{e = d_{min}-1}$ napak oziroma vzorcev napak. Poleg tega je koda sposobna zaznati tudi vzorce z več kot $\mathbf{d_{min}}$ spremenjenimi biti, vendar tu ni zagotovljena popolna detekcija. Izkaže se, da je koda sposobna zaznati $2^n - 2^k$ različnih napak v vektorjih dolžine **n**.

Vseh napak ni sposobna zaznati nobena koda. Koda ni sposobna zaznati napake, ki sprejeti vektor tako močno spremeni, da sovpada z eno od drugih kodnih besed. Takih napak je točno $2^k - 1$. Če pa pride do katerekoli druge napake, se bo ta pokazala pri izračunu sindroma, nakar jo bo sistem zaznal.

Vsak sistem je torej sposoben zaznati večino napak, do katerih lahko pride, vendar je njegova sposobnost odpravljanja le-teh odvisna od minimalne razdalje s katero je bil zasnovan.

4.8 Porazdelitev uteži kodnih besed

Ker uteži posameznih kodnih besed neposredno določajo tudi minimalno razdaljo končne kode, je potrebno uteži primerno razporediti. S pravilno porazdelitvijo dosežemo v danih razmerah najboljše zaznavanje napak. S porazdelitvijo imamo v mislih koliko imamo besed s kakšno utežjo oziroma koliko besed ima enako vrednost uteži.

Pri urejanju porazdelitve si lahko pomagamo z enačbo (20), s pomočjo katere lahko izračunamo kolikšna je pri dani porazdelitvi verjetnost, da se pri sprejemu napaka spregleda. A_j nam pri tem predstavlja število kodnih besed z utežjo j.

$$P_{ND} = \sum_{j=1}^{n} A_j p^j (1-p)^{n-j} \qquad (20)$$

4.9 Hkratno zaznavanje in odpravljanje napak

Ker želimo, da bi naš sistem deloval s čim manjšo zakasnitvijo, bi bilo koristno, če bi lahko napake hkrati zaznal in odpravil. V normalnih razmerah mora sistem napake najprej zaznati, šele potem se začne popravljanje.

Izkaže se, da je taka realizacija možna, vendar prinese v sistem svojo ceno. Če želimo, da procesa potekata vzporedno, se zato zmanjša sposobnost sistema, da napake zaznava. Torej jih zazna manj (zmanjša se **t**), vendar jih lahko sproti popravi. Ker vemo, da je zaznavanje napak osnovano na

minimalni razdalji, se le-ta sedaj deli na dva dela, kot je razvidno iz enačbe (21). V tej enačbi nam α predstavlja število napak, ki jih še lahko popravimo, β pa največje število napak, ki jih lahko zaznamo.

$$d_{\min} \geq \alpha + \beta + 1 \qquad (21)$$

Zaznamo lahko torej **β** napak, popravimo pa $\mathbf{d_{min} - 1 - \beta}$ napak. Iz tabele je razviden primer za d_{min}=9.

Število zaznanih napak (β)	Število popravljenih napak (α)
4	4
5	3
6	2
7	1
8	0

Tabela 1: Hkratno zaznavanje in popravljanje napak

Glede na izbiro parametrov **α** in **β** lahko torej dosežemo, da sistem vse zaznane napake tudi popravi (prva vrstica tabele), lahko pa je število popravljenih napak manjše v korist boljšemu zaznavanju napak.

4.10 Popravljanje izbrisanih bitov

Sistem je lahko zastavljen tudi tako, da namesto da bite označuje za pokvarjene, le-te izbriše. Če tak sistem zazna, da je med prenosom prišlo do motenj oziroma, da je bil bit sprejet nejasno, bo ta bit označil za izbrisanega. Za realizacijo tega sistema je potrebno sistemu omogočiti to dodatno oznako, torej da ima na voljo dodatno zastavico.

Kadar sistem naleti na tako napako, potrebuje za popravilo dva podatka. Vedeti mora, kje je do napake prišlo in kakšna je pravilna vrednost pokvarjenega bita. Prvi podatek dobi avtomatično z uporabo prej omenjene zastavice. Če sistem uporablja binarno kodo, potrebuje samo lokacijo napake, ne pa tudi pravilne vrednosti bita.

Uporaba izbrisanih bitov močno poenostavi dekodirni postopek, ker se točno ve, kateri biti kodne besede so napačni, medtem ko v prejšnjih primerih nismo vedeli, kje je napaka. Zato lahko v tej situaciji kodno besedo še vedno pravilno dekodiramo, če preverimo ostale bite. Izkaže se, da je sistem z minimalno razdaljo $\mathbf{d_{min}}$, ki uporablja izbrise, veliko bolj efektiven pri odpravljanju izbrisov, kot enak sistem za popravljanje napak. Sistem z izbrisi lahko namreč pravilno popravi $\mathbf{d_{min}-1}$ izbrisanih bitov, medtem ko lahko klasičen sistem popravi samo $\mathbf{(d_{min}-1)/2}$ napak.

Še vedno je možno sistem realizirati tako, da hkrati odpravlja tako izbrise kot napake. Uspešnost takega sistema je odvisna od njegove minimalne razdalje $\mathbf{d_{min}}$. Iz enačbe (22) je razvidno, da lahko tak sistem odpravi največ **α** napak in **β** izbrisov. Razvidno je tudi, da vsaka napaka šteje dvojno h končni omejitvi števila rešenih napak in izbrisov.

$$d_{\min} \geq 2\alpha + \beta + 1 \qquad (22)$$

4.10.1 Potek hkratnega odpravljanja napak in izbrisov

Postopek hkratnega odpravljanja izbrisov in napak poteka v drugačnem vrstnem redu kot samo odpravljanje napak.

- V prvi fazi sistem vse izbrisane bite nadomesti z ničlami. Tako spremenjen blok nato dekodira v kodno besedo U_1.
- V naslednji fazi postopek ponovi, le da tokrat izbrisane bite nadomesti s samimi enicami. Iz tega bloka dekodira kodno besedo U_2.
- Sistem nato v obeh kodni besedah preveri, koliko napak se nahaja na mestih, kjer ni bilo izbrisani bitov in izbere za pravilno tisto, ki vsebuje manj napak.

Tak pristop bo vedno dal pravilen rezultat pod pogojem, da je upoštevana enačba (22).

4.11 Zahteve pri načrtovanju sistema

Ko načrtujemo sistem, moramo torej upoštevati več zahtev, da dosežemo najboljše možne rezultate. Če jih na hitro naštejemo:

- Zaželeno je, da je sistem sposoben popraviti napake kjer sta bila pokvarjena en ali dva bita, torej **t = 2**.
- Velikost bloka na katere razdelimo osnovno sporočilo naj bo vsaj 2 bita, torej $\mathbf{k_{min} = 2}$.
- Ko imamo znan **k** in **t**, lahko iz Hammingove meje izračunamo potreben **n**, da bo sistem pravilno deloval
- Poleg Hammingove meje moramo upoštevati še Plotkinovo mejo.
- Pri sami izbiri kodnih besed moramo vedno izbrati vektor samih ničel ter doseči da velja $U_i+U_j=U_k$.

4.12 Simbolne linearne blokovne kode

V vseh dosedanjih poglavjih smo opisovali delovanje linearne blokovne kode, ki se je uporabljala v sistemu, ki komunicira binarno. Vendar se blokovne kode

lahko uporabljajo tudi v sistemih, ki medsebojno komunicirajo z uporabo simbolov.

Sistem lahko zasnujemo tako, da n vhodnih bitov preslika v m izhodnih simbolov. Dober primer tega je sistem, ki pozna več nivojev in lahko namesto enic in ničel prenaša nivoje +3, +1, -1 in -3. V takem sistemu lahko po dva bita prekodiramo v enega od simbolov in tako dosežemo prenos, ki ne vsebuje enosmerne komponente in je zato odpravljanje napak bolj enostavno.

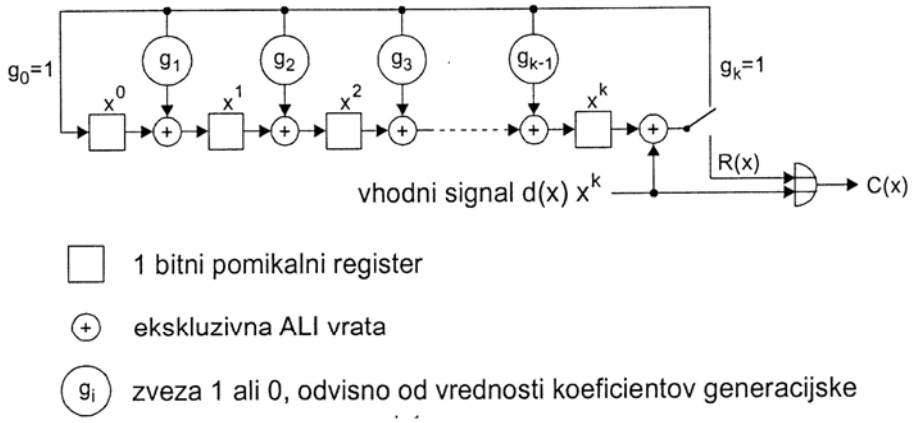
Pri pretvarjanju med biti in simboli je potrebno paziti na to, da izberemo pravilno porazdelitev, da preprečimo pretirano spreminjanje bitov v primeru napake. Če se simbol zaradi šuma na kanalu spremeni v drugega, je zato zaželeno, da se bo pri dekodiranju spremenil samo en bit in bomo zato lahko napako še vedno zaznali. Če pa bi se spremenil cel blok, bi bilo napako veliko težje zaznati.

Simbolne blokovne kode se danes uporabljajo predvsem tam, kjer poteka še analogna komunikacija z več nivoji. Dober primer uporabe bi bil ISDN, kjer se uporablja zgoraj našteti primer, ko 2 bita preslikamo v 4 nivoje in se na ta način znebimo enosmerne komponente med prenosom.

4.13 Ciklične linearne blokovne kode

Ciklične kode so pomembna podskupina linearnih blokovnih kod. Njihova posebnost je, da se kodne besede lahko generirajo z uporabo enostavnega generatorskega polinoma ter pomikalnega registra. Vsak ciklični premik kodne besede nam tako zopet da kodno besedo.

Kodirno vezje lahko realiziramo z uporabo povratno sklopljenega pomikalnega registra kot je prikazano na sliki 2.



Slika 4: Realizacija kodirnika za ciklične kode [3]

Dekodiranje cikličnih kod pa poteka v dveh fazah. V prvi napake detektiramo, v drugi pa napake popravimo. V prvi fazi se sprejeta beseda deli z generacijskim polinomom. Če med prenosom ni prišlo do napak, bo sindrom

tega bloka enak nič. Če pa je do napake prišlo, se le-to lahko odpravi z uporabo tabel ali pa z rabo posebnih algebraičnih postopkov.

4.14 Prednosti in slabosti linearnih blokovnih kod

Linearne blokovne kode so torej postopek, s katerim lahko na sprejemniku pravilno zaznavamo in odpravljamo napake. Glavna prednost, ki nam jo nudijo je neodvisnost oddajnika in sprejemnika. Ker se vsi biti za odkrivanje napak zakodirajo in prenesejo na drugo stran, sprejemnik ne potrebuje povratne povezave z oddajnikom. Zato se lahko ta postopek uporablja v že obstoječih omrežjih. Postopek nam nudi dobro zaznavanje in popravljanje napak, do katerih lahko pride. Poleg tega postopek nudi tudi vse prednosti, ki smo jih opisali v tretjem poglavju.

Vendar pa v zameno za to sposobnost postopek zahteva dodajanje redundance sporočilu, zaradi česar prenosni kanal ni več popolnoma izkoriščen za prenos sporočil. Poleg tega pa ne omogoča zaznave vseh možnih napak do katerih pride, ker je njegova sposobnost zaznavanja omejena s tem koliko bitov redundance dodamo. Ker le-teh ne moremo dodati neskončno, to pomeni da se še vedno lahko zgodi, da bo kakšna napaka ušla sistemu.

Ne glede na to je postopek zelo koristen in danes tudi dokaj razširjen.

5 Zgled

Za zgled še na hitro opišimo zasnovo sistema (6,3).

Želimo sestaviti sistem, ki bi iz bloka treh bitov sestavil kodno besedo dolgo 6 bitov. Tri bitni bloki nam nudijo 2^3=8 različnih sporočil. V našem prostoru pa je možnih 2^6=64 različnih kodnih besed. Zato moramo sedaj med njimi izbrati 8 primernih besed, ki nam bodo zagotovile največjo d_{min}. Iz tabele je razvidno katere kodne besede smo izbrali.

Blok sporočila	Kodna beseda
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

Tabela 2: Izbira kodnih besed

Po izračuni ugotovimo, da imajo tako izbrane kodne besede minimalne razdaljo d_{min} enako 3. Torej lahko popravimo vse napake, kjer je prišlo do spremembe enega samega bita.

Sedaj lahko izračunamo še generatorsko matriko **G** s katero bo kodirnik avtomatsko pretvarjal med izvornim sporočilom in kodnimi besedami. Dobimo

$$G = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

Ta matrika nam prestavlja sistematsko kodo, ker prvi trije stolpci predstavljajo izračun paritete, drugi trije stolpci pa identiteto.

Ker je potrebno na sprejemni strani nato napake zaznati in popraviti, potrebujemo še matriko $\mathbf{H}^T$, s pomočjo katere lahko izračunamo sindrom napake. Za naš primer dobimo:

$$H^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad (24)$$

Sedaj lahko sestavimo standardno matriko, iz katere bomo kasneje na sprejemni strani lahko ugotovili, do katere napake je prišlo in le-to pravilno popravili. Kot vemo moramo v prvi stolpec vpisati vse napake, ki jih lahko pravilno zaznamo in odpravimo, v prvo vrstico vse naše kodne besede, ostalo pa zapolnimo z vsoto napak in besed. Standardna tabela je razvidna iz naslednje tabele.

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110

Tabela 3: Standardna matrika za (6,3) kodo

Zadnji podatek, ki ga naš sistem še potrebuje za pravilno delovanje, je tabela sindromov posamezne napake. To dobimo tako, da vsako napako $\mathbf{e_i}$ iz prvega stolpca pomnožimo z $\mathbf{H^T}$. Tako dobimo še zadnjo tabelo.

Vzorec napake	Sindrom napake
000001	101
000010	011
000100	110
001000	001
010000	010
100000	100
010001	111
000000	000

Tabela 4: Sindromi vzorcev napak

Sedaj ima naš sistem na razpolago vse potrebne podatke.

Če sedaj na primer na oddajniku želimo poslati blok **110**, se le-ta najprej pomnoži z matriko **G.** Tako dobimo kodno besedo **U = 101110**. Ta se nato prenese preko prenosne poti, kjer pride do napake. Sprejemnik zato sprejme **r= 101010**. Sprejemnik nato sprejeti vektor **r** množi z matriko za preverjanje paritete $\mathbf{H}^T$. Ker rezultat ni enak nič, ve da je prišlo do napake. Ker je rezultat množenja enak sindromu $\mathbf{S = rH}^T \mathbf{= 110}$ napake, do katere je prišlo, preveri v zgornji tabeli za katero napako gre. Ugotovi, da je prišlo do napake **e = 000100**. Sedaj gre v standardno matriko in v vrstici, ki pripada tej napaki, poišče vrednost, ki je enaka **r**. Ko jo najde ve, da je sprejel v resnici vrednost ki se nahaja v prvi vrstici stolpca, kjer je to vrednost našel. Torej je v resnici sprejel **r = 101110**. Zato lahko sistem sedaj pravilno ugotovi, da je sprejel blok **110**.

6 Linearne blokovne kode v rabi

Ker so danes linearne blokovne kode dokaj razširjene, bi bilo koristno opisati nekatere od njih, ki se uporabljajo v resničnih sistemih.

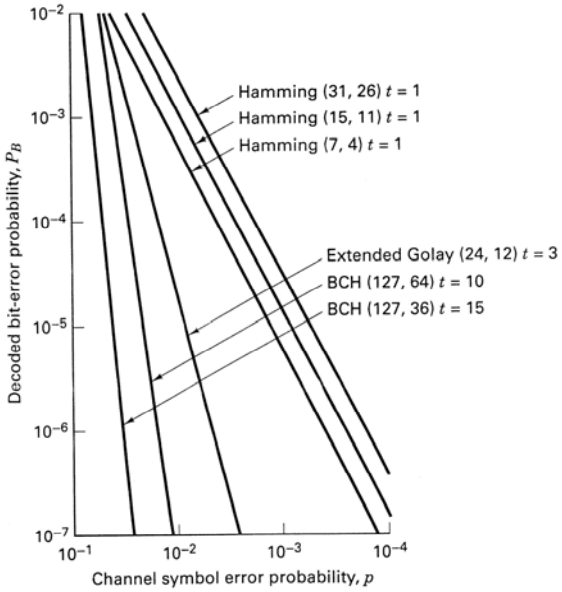
6.1 Hammingove kode

To so linearne blokovne kode, katerih cilj je biti enostavne za uporabo. Razmerje med osnovno velikostjo bloka **k** in velikostjo kodne besede **n**, se pri teh kodah izračuna po enačbi (25).

$$(n,k) = (2^m - 1, 2^m - 1 - m) \qquad (25)$$

Te kode imajo minimalno razdaljo d_{min} enako 3 in so zato sposobne popraviti vse napake, kjer je prišlo do enega pokvarjenega bita, ter zaznati vse napake, kjer sta se spremenila dva ali manj bitov. Te kode zelo dobro delujejo pri uporabi sindroma za zaznavo napake, ker imajo to sposobnost, da lahko že iz sindroma ugotovijo, kje je do napake prišlo.

Čeprav niso najboljše oziroma ne nudijo največ izboljšav, so se uvrstile med perfektne kode. To pomeni, da lahko najbolje odpravljajo vse napake, ki jih lahko zaznajo.



Slika 5: Različne Hammingove kode [1]

Pri Hammingovih kodah lahko namesto enačbe (18), za izračun verjetnosti napake pri dekodiranju uporabimo poenostavljeno enačbo (26)

$$P_B \approx p - p(1-p)^{n-1} \tag{26}$$

6.2 Razširjena Golayeva koda

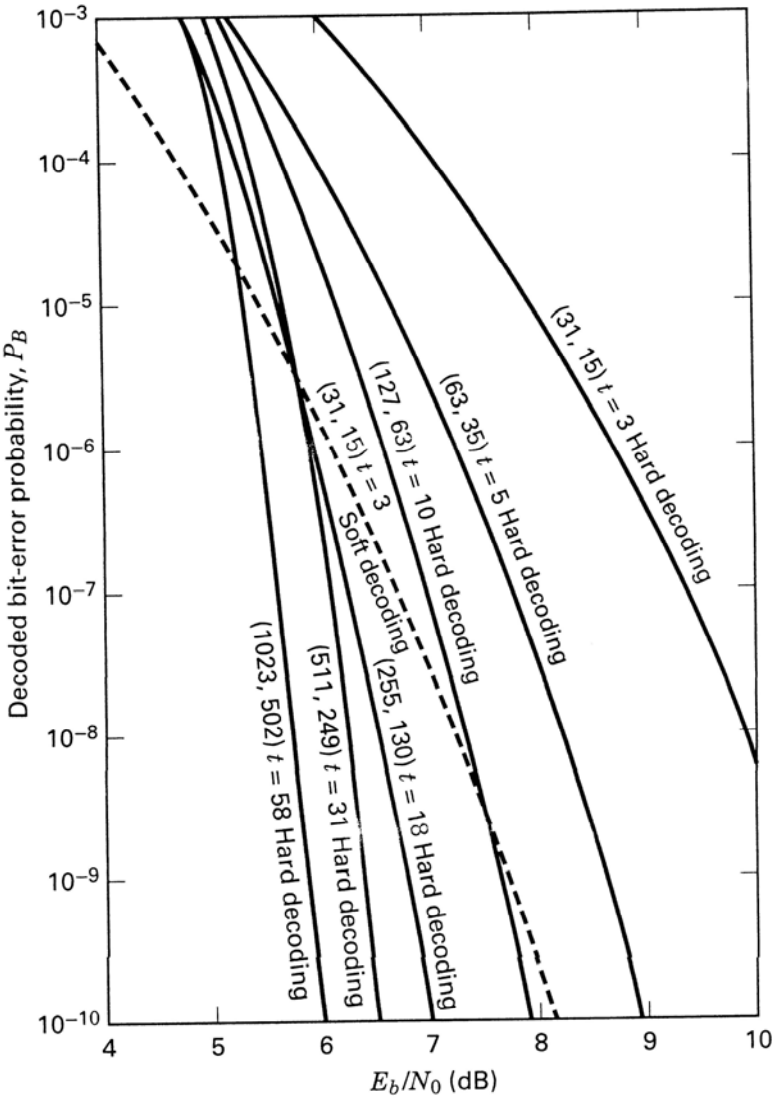
Golayeva koda je koda tipa (23,12), ki ima d_{min} enako 7. Razširjeno kodo pa dobimo, če ji dodamo še en dodaten paritetni bit, ki drastično izboljša lastnosti kode. Taka (24,12) koda ima d_{min} enako 8. Poleg tega se lažje realizira za uporabo v sistemih. Ta tip kode je veliko boljši od Hammingovih kod, vendar v zameno za to veliko zahteva. Sistem potrebuje kompleksno dekodirno vezje, da lahko pravilno dekodira kodne besede. Poleg tega se zmanjša bitna hitrost prenosa in posledično potrebuje veliko večjo pasovno širino za uspešen prenos.

Ker ima ta koda d_{min} = 8, to pomeni, da je sposobna pravilno popraviti vse napake, ker so se med prenosom pokvarili največ trije biti. Sprejemnik je lahko zasnovan tako, da je sposoben popraviti tudi napake (vendar ne vseh), kjer so se med prenosom pokvarili po štirje biti. Verjetnost napake med dekodiranjem pri uporabi Golayeve kode izračunamo po enačbi (27).

$$P_B \approx \frac{1}{24}\sum_{j=4}^{24} j\binom{24}{j}p^j(1-p)^{24-j} \tag{27}$$

6.3 BCH kode

Bose-Chadhuri-Hocquenghem (BCH) kode so predelava Hammingovih kod. V nasprotju z ostalimi opisanimi kodami so to zelo močne ciklične kode, ki nam nudijo veliko izbiro prenosnih hitrosti, velikosti blokov in sposobnosti popravljanja napak. BCH kode so zelo pomembne, ker v primeru zelo velikih velikosti bloka (**k**) dajejo boljše rezultate od drugih blokovnih kod.



Slika 6: BCH kode [1]

7 Zaključek

V tej seminarski smo predstavili linearne blokovne kode, ki nam predstavljajo zelo pomembno orodje pri kanalskem kodiranju. Z uporabo teh kod lahko dosežemo veliko manjšo verjetnost napak med prenosom, kot če bi signal prenašali brez kodiranja. Omogočajo nam tudi, da odpravljamo napake, do katerih je prišlo in na ta način še dodatno izboljšamo kvaliteto prenosa.

Vendar v zameno za te izboljšave postavijo tudi svojo ceno. Za uporabo teh kod moramo v sistem vgraditi dekodrino vezje, ki je sposobno napake prepoznati in popraviti. Glavna cena, ki jo postavijo kodni postopki, pa se izraža v potrebi po zvečani pasovni širini. Sicer v zameno za povečano pasovno širino veliko pridobimo, vendar nam to še vedno predstavlja zgornjo mejo izboljšav, ki jih lahko dosežemo.

Kljub temu so linearne blokovne kode postale del današnjih komunikacijskih sistemov in bodo zelo verjetno tam ostale še veliko časa. Ker se ves čas razvijajo novi kodni postopki, se bodo zelo verjetno s časom zmanjšale tudi zahteve po pasovni širini oziroma se še izboljšale prednosti, ki jih dobimo s povečanjem le-te.

Literatura

[1] B. Skalar, Digital Communications, Communications Engineering Services, Tarzana, Clifornia and University of California, Los Angeles, January 2004.

[2] S. Tomažič, Digitalne komunikacije, zapiski, 2003.

[3] J. Bešter, B. Pehani, Osnove Telekomunikacij II, založba FE in FRI, 2002.

[4] S. Tomažič, Osnove telekomunikacij I, 1. izdaja, Založba FE in FRI, Ljubljana, 2000.

[5] I. Medič, Digitalne komunikacije, zapiski, 2002.

[6] M. Z. Wang, Error Correcting Codes, internet, http://www.eie.polyu.edu.hk/~enmzwang/adc/l-notes/l-notes.html

[7] M. Vidmar, Sevanje in razširjanje valov, Založba FE in FRI, Ljubljana, 2004.