UNIVERZA V LJUBLJANI

Fakulteta za elektrotehniko

Stanislav Kovačič

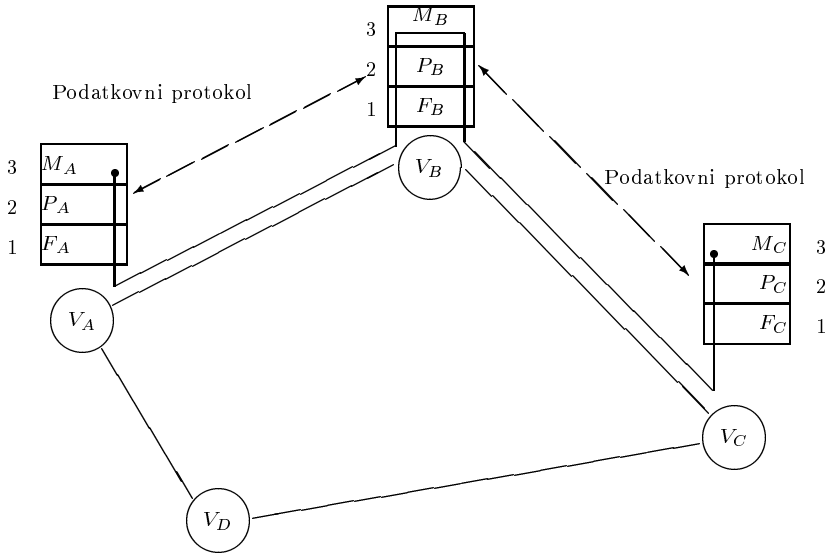
Komunikacije v avtomatiki

(študijsko gradivo)

Ljubljana, 25. oktober 2001

4 Elementi podatkovnega sloja

Podatkovni sloj nudi storitve neposredno višjemu mrežnemu sloju. Njegova naloga je, da poskrbi za prenos podatkov z visoko stopnjo zanesljivosti od procesa (aktivnosti) mrežnega sloja na eni strani do procesa mrežnega sloja na drugi strani, glej sliko 56. Proces mrežnega sloja M_A vozlišča V_A želi komunicirati s procesom M_C vozlišča V_C. Zato "prosi" podatkovni sloj, ki skrbi za prenos podatkov med V_A in V_B, naj mu omogoči komunikacijo s procesom M_B vozlišča V_B. Podobno proces M_B prosi podatkovni sloj, ki je zadolžen za povezavo od V_B do V_C, naj mu omogoči komunikacijo s procesom M_C. Vmesno vozlišče V_B v tem primeru opravlja funkcije posrednika. Posredništvo se zgodi na tretjem (mrežnem) nivoju. Takemu vozlišču navadno pravimo 'usmerjevalnik' (ang. Router). Če bi se posredništvo zgodilo na podatkovnem sloju brez angažiranja mrežnega sloja, bi vozlišče V_B imenovali most (ang. Bridge).



Slika 56: Podatkovni sloj nadgradi nad nezanesljivim prenosom bitov fizičnega sloja zanesljiv prenos paketov mrežnega sloja. Aktivnosti podatkovnega sloja sosednjih vozlišč se pri tem sporazumevajo po podatkovnem protokolu.

Podatkovni sloj se poslužuje storitev sosednjega nižjega - fizičnega sloja. Ta mu zagotavlja prenos binarnih simbolov (signalov) od vozlišča do vozlišča, ki pa zaradi nepopolnosti naprav in povezav ter zunanjih motilnih vplivov v splošnem ni nikoli popolnoma zanesljiv. Nezanesljiv prenos podatkov je s stališča višjih slojev premalo kvalitetna storitev. Zato podatkovni sloj nadgradi nad nezanesljivim fizičnim slojem zanesljiv prenos podatkov med sosednjimi vozlišči. To doseže z delitvijo zaporedja bitov na okvirje ali okvirjenjem (ang. Framing) in z nedvoumnim označevanjem začetka in konca okvirja, prenosom posameznih okvirjev od vozlišča do vozlišča skupaj z ugotavljanjem in redkeje s popravljanjem napak, ponavljanjem prenosa poškodovanih okvirjev in reševanjem problema podvojenih

in izgubljenih okvirjev (ang. Flow and Error Control). Podatkovni sloj opravlja še druge naloge v zvezi s prenosom podatkov, denimo vzpostavljanje, vzdrževanje in sproščanje zveze. Zelo važna naloga tega sloja posebno v lokalnih omrežjih pa je nadzor nad dostopom do skupnega prenosnega medija (ang. Media Access Control - MAC). Pravila in dogovore, ki jih oddalejene (istorodne) aktivnosti podatkovnega sloja pri tem upoštevajo, imenujemo protokol podatkovnega sloja. Protokol bi torej lahko imenovali tudi krajevno porazdeljen algoritem za prenos podatkov.

4.1 Okvirjenje

Delitev neprekinjenega niza bitov na krajše enote, tako imenovane okvirje, imenujemo okvirjenje. Z okvirjenjem nedvoumno označimo kdaj okvir začne in kdaj konča. Štirje najpogostejši načini okvirjenja so:

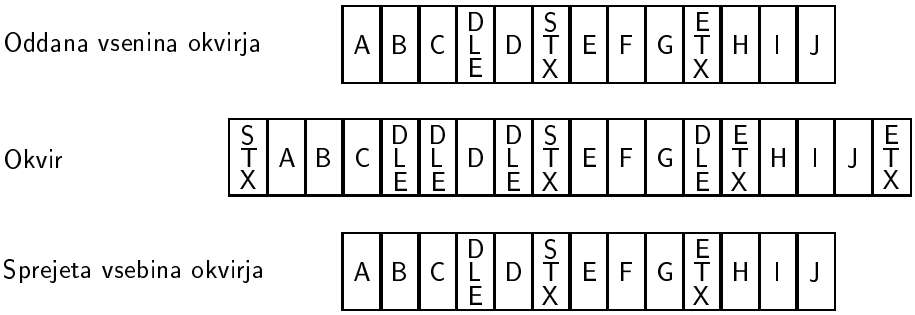
- označevanje začetka in konca okvirja z domenjenimi nadzornimi znaki ter vrivanje napovednega znaka pred podatke, ki so morebiti enaki nadzornim znakom,
- označevanje začetka in konca okvirja z domenjenim bitnim vzorcem ter vrivanje 'polnilnih' bitov pred podatkovne bite v primeru dvoumnosti,
- označevanje začetka in konca okvirja z drugačno obliko signala, kot je predvidena za prenos podatkov,
- zapis dolžine okvirja na začetku okvirja.

Zadnji način predvideva na začetku (v "glavi" ali "zaglavju") okvirja polje, ki vsebuje število podatkov v okvirju. Če je to polje okvirja med prenosom poškodovano, sprejemnik ne more ugotoviti dolžine tekočega okvirja, lahko zgreši tudi začetek naslednjega okvirja in sprejemnik izgubi sinhronizacijo z oddajnikom. Brez dodatnih mehanizmov je praktično nemogoče ponovno vzpostaviti sinhronizacijo sprejemnika z oddajnikom na začetek okvirja. Zato se zapis števila podatkov v okvirju za okvirjenje skoraj nikoli ne uporablja samostojno, ampak skupaj z enim od drugih treh načinov.

Znakovno usmerjen način okvirjenja reši problem sinhronizacije sprejemnika na začetek okvirja z vnaprej dogovorjenim *začetnim znakom*, ki je tipično ASCII[20] kodiran znak z imenom STX (Start of TeXt). Pred oddajo vsebine okvirja oddajna naprava odda začetni znak, ki ga pričakuje sprejemnik. Podobno se konec okvirja označi z domenjenim *končnim znakom* ETX (End of TeXt) ali ETB (End

[20] v IBM EBCDIC kodu ima enako ime

of Block). Ker se ASCII kod uporalja za kodiranje besedila, pri prenosu 'čistega' besedila ni težav, saj se nadzorni znaki kot sta znaka STX in ETX nikoli ne pojavijo med podatki znotraj okvirja. Težave se začnejo pri prenosu "binarnih" podatkov, kot so stanja stikal, vrednosti meritev, i.t.d., kjer je za podatke možna vsaka kombinacija binarnih simbolov. Ena od rešitev je avtomatično vstavljanje napovednega znaka (na primer ASCII DLE - Data Link Escape) na oddajni strani pred vsak podatek, ki je enak nadzornemu znaku. Sprejemna naprava razume napovedni znak DLE kot opozorilo, da bo sledil podatek in ne nadzorni znak, odstrani napovedni znak in ohrani podatek, ki je sicer enak nadzornemu znaku. Primer 'polnjenja' znakov je narisan na sliki 57. Napovedni znak je potreben tudi tedaj, ko se njegova koda DLE pojavlja kot podatek znotraj okvirja.



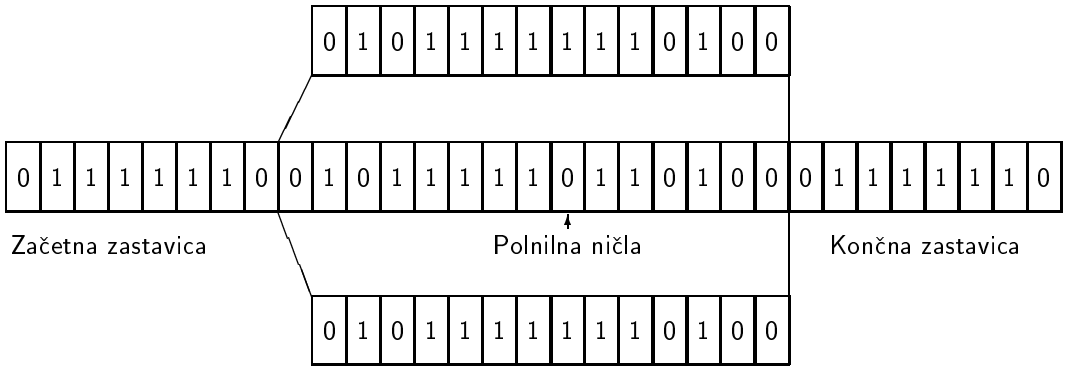
Slika 57: Okvirjenje z začetnim in končnim znakom in vstavljanje napovednega znaka. Zgoraj: prvotno zaporedje podatkov (vsebina okvirja). V sredini: okvir v času oddaje dopolnjen z začetnim, s končnim in z napovednimi znaki pred podatki, ki so enaki nadzornim znakom. Spodaj: 'očiščeni' sprejeti podatki.

To ni edina možnost za rešitev problema dvoumnosti med podatki in nadzornimi znaki. Možen je tudi obraten dogovor, in sicer: napovedni znak naj služi kot opozorilo, da bo sledil nadzorni znak in ne podatek. Okvir začne z DLE in STX ter konča z DLE in ETX, medtem ko se nadornih znakov znotraj okvirja ne napoveduje, razen kadar se kot podatek pojavi DLE.

Označevanje začetka in konca okvirja z domenjenimi nadzornimi znaki nas veže na osembitne podatke in način (ASCII) kodiranja. Uporablja se, kot pravimo, v znakovno usmerjenih podatkovnih protokolih za sinhroni in asinhroni prenos. Znakovne protokole so danes skoraj popolnoma izrinili bitno usmerjeni protokoli, ki niso vezani na način kodiranja.

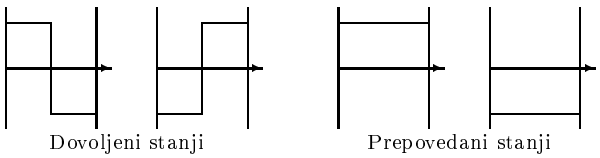
Okvirjenje z začetnim in končnim bitnim vzorcem je neodvisno od načina kodiranja (ASCII, EBCDIC), in dovoljuje poljubno število bitov na podatek. Okvir začne z domenjenim začetnim bitnim vzorcem, ki ji rečemo 'otvoritvena' ali začetna zastavica (ang. Opening Flag). Začetno zastavico sestavlja uvodna ničla, neprekinjeno zaporedje šestih enic in zaključna ničla: 01111110. Končna zastavica (ang. Closing Flag) je kar enaka začetni zastavici. Ko sprejemnik sprejme zaporedje šetnih ničel to razume kot začetek novega (ali konec tekočega) okvirja.

Zato se znotraj okvirja tako zaporedje binarnih simbolov ne sme nikoli pojaviti. Kadarkoli v nizu podatkovnih bitov oddajnik odkrije pet ali več zaporednih enic, za peto enico avtomatično vstavi eno 'polnilno' ničlo. Šest zaporednih enic, ki bi jih sprejemnik lahko zamenjal z začetnim vzorcem, se med podatki tako ne more pojaviti. Ko sprejemnik sprejme pet zaporednih enic ve, da mu sledi polnilna ničla, 'vsrka' ničlo in niz podatkovnih bitov dobi prvotni pomen. Izgled okvirja in primer vstavljanja ničle (ang. Bit Stuffing) prikazuje slika 58.



Slika 58: Okvirjenje z začetnim in končnim vzorcem in polnjenjem ničle, če se v nizu podatkovnih bitov pojavi zaporedje enic, ki je daljše od petih bitov. Zgoraj: prvotno zaporedje bitov. V sredini: oddani okvir s primerom polnilne ničle. Spodaj: 'očiščeno' sprejeto zaporedje bitov.

Zadnja možnost okvirjenja je označevanje začetka in konca okvirja s takim vzorcem informacijskega signala, ki se znotraj okvirja ne sme pojaviti, je prepovedan. Na primer, z informacijskim signalom, ki zmore štiri stanja, lahko eno stanje pomeni logično nič, eno stanje pomeni logično ena, ostali dve stanji pa se izkoristita za označevanje začetka in konca okvirja, glej sliko 59. Znotraj bitne celice sta dovoljeni dve stanji, prehod signala z visokega na nizek nivo in prehod signala z nizkega na visok nivo. Stanji signala cel čas nizek nivo ali cel čas visok nivo sta za kodiranje podatkov prepovedani, lahko pa ju izkoristimo za kodiranje začetka in konca okvirja.



Slika 59: Primer signala s štirimi možnimi stanji, prvi dve sta predvideni za kodiranje podatkov (0 in 1). Drugi dve stanji sta za kodiranje podatkov prepovedani, služita pa za označevanje začetka in konca okvirja.

4.2 Nadzor nad napakami in nad pretokom podatkov

V tem podpoglavju se bomo posvetili postopkom za prenos podatkov, ki zagotovljajo, da je tok podatkov med oddajno in sprejemno napravo v vseh okoliščinah čimbolj zanesljiv in kar se da tekoč. S temi postopki rešujemo probleme, ki so posledica nepopolnosti naprav in povezav, končne hitrosti delovanja naprav oziroma razlike v hitrosti med oddajanjem in sprejemanjem, omejene velikosti sprejemnih in oddajnih medpomnilnikov, kasnitve med sprejetim in oddanim signalom in podobno. To vedno vključuje odkrivanje in včasih popravljanje napak.

V glavnem se postopki med seboj razlikujejo glede na stopnjo zanesljivosti, v pogledu prepustnosti, po izkoriščenosti kanala, po kasnitvah med oddajo in sprejemom, in v zahtevnosti izvedbe. Nekateri postopki (protokoli) zagotavljajo boljšo prepustnost na račun daljšega čakalnega časa, drugi spet skrajšajo čakalni čas na račun izkoriščenosti kanala. Vsi pa vnašajo redundanco.

Kadar govorimo o prepustnosti (ang. Throughput), nas ne zanima trenutna hitrost prenašanja podatkov, ampak se zanimamo za efektivno hitrost prenašanja podatkov gledano čez daljše časovno obdobje. *Prepustnost* je po definiciji enaka povprečni vrednosti pretoka koristnih podatkov. Kaj so 'koristni' podatki ni enoznačno določeno. Včasih je to cel okvir (recimo glava in podatki) včasih pa je to samo podatkovni del okvirja.

Izkoriščenost (ang. Efficiency) je razmerje med prepustnostjo in maksimalno možno hitrostjo prenosa (to je hitrostjo oddajanja, ki se običajno enači s kapaciteto kanala). Običajno se izkoriščenost računa kot razmerje časa, ko se prenašajo koristni podatki in časa, ki je v povprečju potreben za prenos teh podatkov.

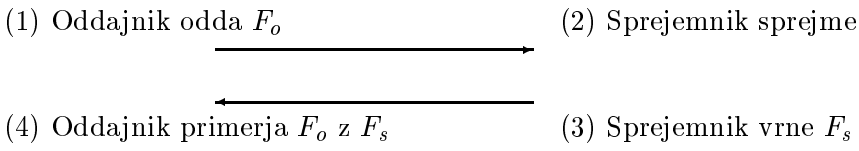
Odzivni čas ali reakcijski čas (ang. Response Time) je tisti čas, ki preteče od zahteve za prenos (za oddajo) do tedaj, ko so podatki (okvir) prenešeni. Je seštevek časa čakanja, da pride okvir na vrsto, časa oddajanja in sprejemanja (procesiranja) in časa širjenja signala od oddajnika do sprejemnika. Hiter odziv je v sistemih stvarnega časa pomembnejši od visoke prepustnosti.

Uporabljajo se trije osnovni koncepti nadzora nad tokom podatkov med oddajno in sprejemno napravo:

- preverjanje odmeva (ang. Echo Checking),
- vnaprejšnje popravljanje napak (ang. Forward Error Correction - FEC) in
- (avtomatska) zahteva za ponovitev prenosa okvirja (ang. Automatic Repeat Request - ARQ).

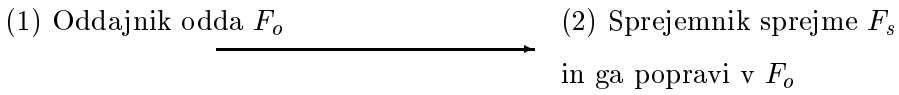
Preverjanje odmeva je najenostavnejši način za odkrivanje napak med

prenosom. Oddajna naprava odda podatek ali zaporedje podatkov in nato čaka na odgovor sprejemne naprave. Sprejemna naprava sprejme podatke in jih vrne oddajniku v nespremenjeni obliki. Oddajna naprava primerja oddane podatke s sprejetimi in če se ne razlikujejo, to pomeni, da je sprejemik podatke pravilno sprejel. Preverjanje pravilnosti prenosa opravlja oddajnik. Takšen način zahteva dupleksni kanal in zasede pol kapacitete kanala za preverjanje pravilnosti prenosa (polovico za prenos podatkov/ informacije v eni smeri in drugo polovico za prenos v drugi smeri).



Slika 60: Preverjanje odmeva. Preverjanje opravlja oddajnik.

Vnaprejšnje popravljanje napak (FEC) se pojavlja skupaj z enim od kodov za popravljanje napak (na primer s Hammingovim kodom, RS kodom, BCH kodom, Golay-evim kodom, ...). Oddajna naprava doda h koristni informaciji dovolj odvečne informacije, da lahko sprejemna naprava odkrije (ali vsaj z zelo veliko verjetnostjo) napako in jo tudi popravi. Slabost vnaprejšnjega popravljanja napak je nizka izkoriščenost kanala zaradi velike odvečnosti v prenašanih podatkih. To je glavni vzrok, da se razmeroma malo uporablja. Prednost tega postopka se pokaže pri simpleksnih prenosnih poteh, pri prenosu na velike razdalje, kjer postane zakasnitev sprejemnika glede na oddajnik odločilnega pomena, in pri prenosu po zelo motenih prenosnih poteh.

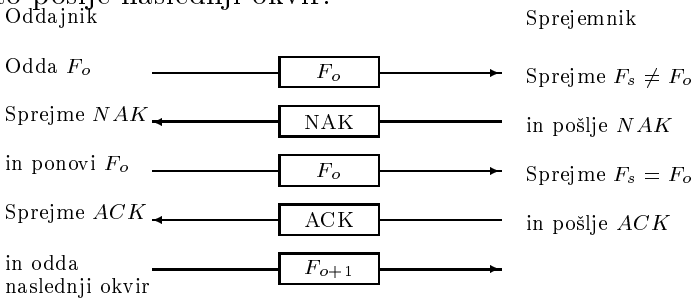


Slika 61: Vnaprejšnje popravljanje napak. Popravljanje opravlja sprejemnik.

Avtomatska *zahteva za ponovitev* (ARQ) se v podatkovnih omrežjih največ uporablja. Eden od vzrokov za to je najbrž tudi v dejstvu, da so sodobne prenosne poti razmeroma zanesljive (napake se redko pojavijo), in da se informacija običajno prenaša v obliki blokov (okvirjev ali paketov). Zahteva za ponoven prenos napačno prenešenega okvirja se uporablja skupaj s kodirnimi postopki za odkrivanje napak (tipično s preverjenjem parnosti ali s cikličnim preverjanjem). Ti vnašajo znatno manj redundance kot postopki za popravljanje napak.

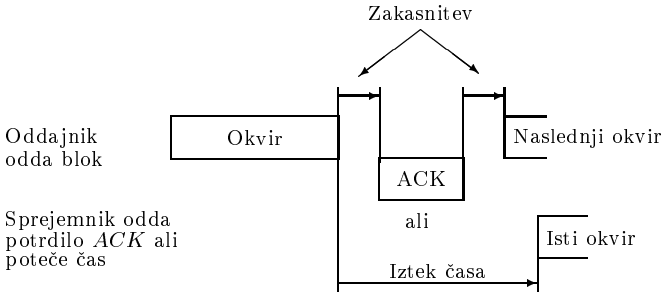
Postopek ARQ vključuje preverjanje okvirjev s strani sprejemne naprave. Zato mora oddajna naprava koristni informaciji pred oddajo v kanal dodati po nekem postopku dovolj odvečne informacije, da je sprejemna naprava zmožna

odkriti napako, ki se mogoče pojavi med prenosom. V primeru, da napako odkrije ($F_s \neq F_o$), zavrže sprejeti okvir ter po povratnem kanalu z negativno potrditvijo, ki se tipično označuje z NAK (ang. Negative Acknowledge), zahteva od oddajne naprave ponovno oddajo napačno prenešenega okvirja. V primeru, da je okvir prenešen brez napake ($F_s = F_o$) ali sprejemnik napake ne odkrije, pošje oddajniku pozitivno potrdilo ACK (ang. Positive Acknowledge) in s tem potrdi pravilen sprejem. Oddajnik nato pošlje naslednji okvir.



Slika 62: Avtomatska zahteva za ponovitev. Preverjanje opravlja sprejemnik.

Pri avtomatski zahtevi za ponovitev se lahko potrjuje tudi samo pravilno sprejete okvirje, nepravilen prenos pa pomeni odsotnost potrdila. Razmere prikazuje slika 63. Oddajnik pošlje okvir in nato čaka na potrdilo. V ta namen nastavi časovni števec in če se ta 'izteče' (ang. Time-Out) predno prispe pozitivno potrdilo od sprejemnika, sledi ponovitev okvirja. Čakalni interval mora biti primerno izbran. Predolgo čakanje po nepotrebnem zmanjša prepustnost. Vendar mora biti čakalni interval vsaj nekoliko daljši od časa za prenos potrdila. Pozitivno in negativno potrjevanje (ACK in NAK) zato običajno zagotavlja boljšo izrabo prenosne poti.

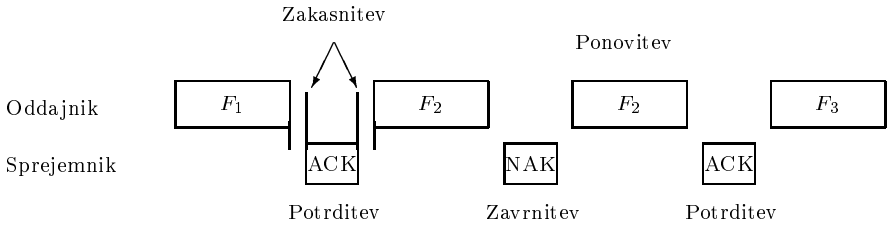


Slika 63: Potrjevanje s čakanjem in z iztekom časa.

Obstajata dve možnosti za realizacijo avtomatske zahteve za ponovitev: potrjevanje s čakanjem (ang. Stop-And-Wait ali tudi Send-And-Wait) in potrjevanje brez čakanja.

4.2.1 Potrjevanje s čakanjem

Pri *potrjevanju s čakanjem* oddajna naprava po oddaji tekočega ovirja in pred oddajo naslednjega vedno počaka na sprejem potrdila, glej sliko 64. Šele nato bodisi odda naslednji okvir bodisi ponovi tekoči okvir. Zaradi čakanja na potrditev in zaradi prenosa potrdila samega pride med dvema zaporednima okvirjema do časovnega presledka (premora). V tem času se ne prenašajo koristni podatki in kanal je neizkoriščen. Premor je še posebno dolg pri polduplekšnih zvezah, ker se za preklop z oddaje na sprejem porabi še nekaj več časa. Posledica pa je slaba izkoriščenost prenosne poti.



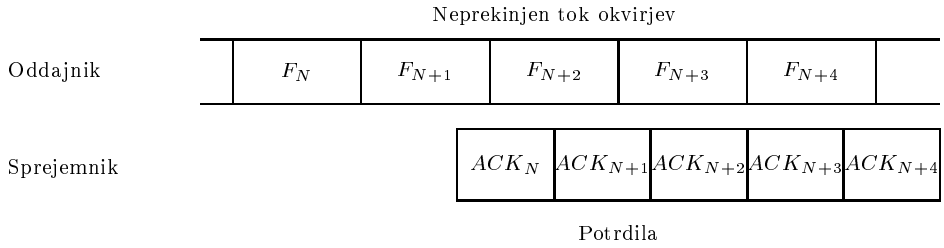
Slika 64: Potrjevanje s čakanjem.

Kaj se zgodi, če se izgubi (ali pokvari) potrdilo sicer pravilno sprejetega okvirja? Oddajna naprava v takem primeru napačno sklepa, da je bil sprejem tekočega okvirja neuspešen in še enkrat pošlje isti okvir. Posledica tega je dvakratno pošiljanje in sprejem istega okvirja. Oziroma, oddajnik ponovi že pravilno sprejeti okvir in sprejemnik sprejme (ohrani in preda naprej) isti okvir dvakrat. Problem podvojenih okvirjev rešimo s številčenjem okvirjev. Vsak okvir je označen z zaporedno številko. Če sprejemnik sprejme dva okvirja z enako številko, enega od obeh enostavno zavrže.

Številčenje okvirjev je nujni sestavni del vsakega podatkovnega protokola. Številčenje okvirjev uporablja tudi protokol XMODEM za prenos podatkov med majhnimi računalniki, ki je bil podlaga za nastanek bolj izpopolnjenih protokolov (YMODEM, ZMODEM). Pri potrjevanju s čakanjem je vedno nepotrjen samo eden (zadnji) okvir, ki ga je potrebno razlikovati od predhodnjega okvirja. Torej je dovolj, da izmenoma označujemo (številčimo) okvirje enkrat z nič in drugič z ena. Eden najbolj znanih protokolov potrjevanja s čakanjem uporablja nič/ena številčenje in je po njem dobil tudi ime ABP protokol (ang Alternating-Bit-Protocol). Sami se prepričajte, da je potrebno ne le številčenje okvirjev, ampak tudi številčenje potrdil.

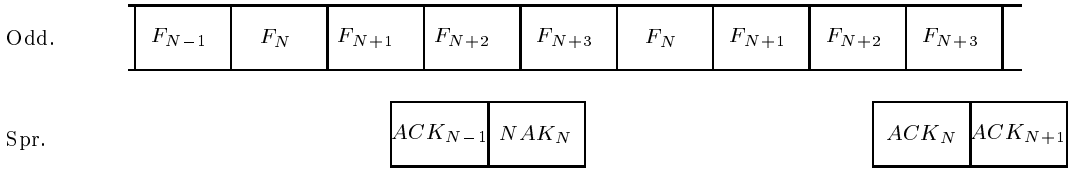
4.2.2 Potrjevanje brez čakanja

Pri *potrjevanju brez čakanja* je tok okvirjev od oddajnika k sprejemniku neprekinjen. Okvirji si sledijo drug za drugim, kot je narisano na sliki 65. Prenosna pot je zato bolje izkoriščena. Ker oddajnik s potrjevanjem okvirjev 'zamuja', je poleg številčenja okvirjev nujno potrebno tudi številčenje potrdil. Iz številke potrdila sprejemnik ve, na kateri okvir se nanaša potrdilo. Takšen način potrjevanja ima prednost pred potrjevanjem s čakanjem posebno na kvalitetnih prenosnih poteh s sicer relativno veliko zakasnitvijo sprejemnika proti oddajniku. Zahteva pa na oddajni strani začasno shranjevanje večjega števila oddanih, a še ne potrjenih okvirjev in s tem večji oddajni medpomnilnik.



Slika 65: Princip potrjevanja okvirjev brez čakanja.

Denimo, da pride med prenosom do napake okvirja z zaporedno številko N. Na voljo sta dve možnosti: vračanje nazaj na okvir s številko N ali s kratico GBN (ang. Go-Back-N) in selektivna ponovitev okvirja ali s kratico SRQ (ang. Selective-Repeat-Request), imenovana tudi SRP (ang. Selective Repeat Protocol). Pri *vračanju nazaj na N* pošlje sprejemnik, ko ugotovi napako okvirja s številko N, oddajniku negativno potrdilo (zavrnitev) s številko N (NAK_N), ali pa negativnega potrdila enostavno ne pošlje in pride do izteka časa. Sprejemnik zavrže N-ti okvir, prezre pa tudi vse kasnejše okvirje, ki jih je že oddal oddajnik (slika 66).

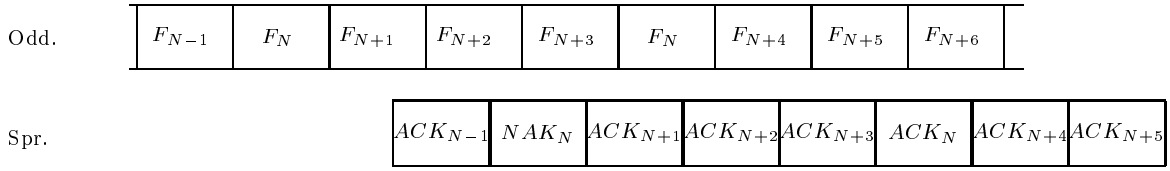


Slika 66: Ponavljanje z vračanjem nazaj na N (GBN).

Oddajnik razume zavrnitev NAK_N kot zahtevo, da se mora oddajanje vrniti nazaj na N-ti okvir. Zato odda najprej pokvarjeni okvir s številko N, za njim pa še vse naslednje okvirje ($N+1, N+2$, i.t.d.) tudi če so bili prenešeni brez napake. Posledica takega načina ponavljanja je razmeroma slaba izkoriščenost manj kvalitetnih prenosnih poti, ker je veliko tudi nepotrebnih ponovitev. Dobra

lastnost GBN pa je, da na sprejemni strani (na podatkovnem sloju) ne zahteva medpomnilnika večjega od velikosti enega okvirja.

Pri *selektivnem ponavljanju* v primeru napake N-tega okvirja oddajna naprava na zahtevo sprejemne naprave ponovi samo pokvarjeni (N-ti) okvir, kasnejših okvirjev $N+1, N+2$, i.t.d. pa ne. Posledica takega načina ponavljanja je, da imajo pravilno sprejeti okvirji drugačen vrstni red kot oddani okvirji. Z drugimi besedami, možno je, da sprejemnik prej dobi okvir, ki je bil oddan kasneje. Možne razmere so narisane na sliki 67. Da je na sprejemni strani možno obnoviti prvoten vrstni red okvirjev, je potrebno preurejanje okvirjev. To zahteva začasno pomnenje sprejetih okvirjev in kajpak razmeroma velik vmesni pomnilnik sprejemne naprave. S selektivnim ponavljanjem blokov torej bolje izkoristimo kanal (ni nepotrebnega ponavljanja), potrebujemo pa dodaten vmesni pomnilnik sprejemnika. Ta pomnilnik mora biti dovolj velik, da hrani vse sprejete in še ne potrjene okvirje. Če je možno veliko število nepotrjenih okvirjev, moramo predvideti velik medpomnilnik. Ker je težko oceniti zadostno velikost medpomnilnika, se ta način razmeroma manj uporablja kot GBN protokol.



Slika 67: Selektivno ponavljanje napačno sprejetega okvirja.

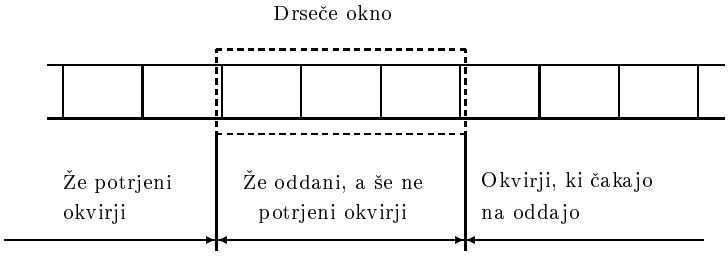
Na osnovi povedanega lahko povzamemo naslednje. V splošnem se pri prenosu podatkovnih okvirjev javljajo tile problemi:

- sprejemnik mora biti sposoben vzdrževati pravilno zaporedje podatkovnih okvirjev,
- vsi že oddani, vendar še ne potrjeni okvirji morajo biti na oddajni strani začasno shranjeni zaradi morebitne potrebe po ponovitvi,
- sprejemnik ima omejeno hitrost sprejemanja in omejeno velikost pomnilnika. Torej je možno, da bi oddajnik oddajal hitreje, kot je sprejemnik sposoben sprejemati.

Iz tretje ugotovitve neposredno sledi, da bo v nekaterih primerih oddajnik moral za določene čas počakati z oddajo. V predhodnih razdelkim pa smo tudi že ugotovili, da je za vzdrževanje pravilnega zaporedja podatkovnih okvirjev potrebno številčenje okvirjev in v večini primerov tudi številčenje potrdil. V primeru, da

prenašamo daljša zaporedja okvirjev lahko postanejo zaporedne številke prevelike. Za zapis velikih števil pa moramo predvideti več bitov. Vendar enoznačno številčenje okvirjev od začetka do konca sporočila sploh ni potrebno. Dejansko mora biti enoznačeno označenih (številčenih) samo toliko okvirjev, kolikor jih sme biti v določenem času že oddanih in še ne potrjenih. Število takih okvirjev imenujemo velikost okna. Mislimo si lahko, da po časovnem zaporedju okvirjev drsi okno velikosti W, okvirje pa potem številčimo z $0, 1, \ldots, N-1 = W$ in nato spet od 0 naprej (glej sliko 68). Torej po modulu N. Če število nepotrjenih okvirjev doseže N, mora oddajnik z oddajo počakati, dokler sprejemnik vsaj nekaj okvirjev ne potrdi. Takšen pristop k nadzoru nad tokom podatkov je znan pod izrazom *drseče okno* (ang. Sliding-Window-Protocol) in se uporablja skupaj s selektivnim ponavljanjem ali z vračanjem nazaj na N. Zadnja rešitev ima za posledico naslednji pridobitvi:

- omejimo velikost zaporedne številke (recimo na 8) in s tem število bitov za zapis številke (v tem primeru 3),
- omejiti smemo velikost sprejemnega in oddajnega pomnilnika.



Slika 68: Koncept drsečega okna.

Poglavje zaključimo z razvrstitvijo tehnik prenašanja, ki spadajo v skupino avtomatske zahteve za ponovitev – ARQ. Razlikujejo se glede na način potrjevanja, in sicer:

- samo s pozitivim potrdilom (ACK) ali
- s pozitivnim in z negativnim potrdilom (ACK/NAK).

Tako prva kot druga oblika je možna pri potrjevanju

- s čakanjem (ABP) ali
- brez čakanja (GBN ali SRP) z drsečim oknom.

4.3 Vrednotenje podatkovnih protokolov

4.3.1 Potrjevanje s čakanjem

Poglejmo, kako potrjevanje s čakanjem vpliva na izkoriščenost in s tem prepustnost kanala. Kadarkoli po kanalu ne prenašamo ničesar ali po kanalu prenašamo kaj drugega kot koristne podatke (nadzorne podatke, kot so številka okvirja, biti za preverjanje pravilnosti prenosa, potrdilo ali kaj drugega), je kanal neizkoriščen. Čim več je takšnih premorov ali nadzornih podatkov, tem manj je kanal izkoriščen, po drugi strani pa je nižja tudi prepustnost kanala. Proučujmo različico potrjevanja s čakanjem in z iztekom časa v primeru napake pri prenosu okvirja, glej sliko 69. Naj bo za naš izračun kapaciteta kanala C enaka 9600 bitov na sekundo, za dolžino okvirja F vzemimo 512 bitov, za dolžino potrdila A izberimo 8 bitov, kasnilni čas T_z naj bo 0.2 milisekund. Verjetnost za napako okvirja p in vrednost za iztek časa T_o bomo določili kasneje. Nadalje predpostavimo, da so vsi okvirji enako dolgi in da vsak okvir predvideva $D = 496$ bitov za koristne podatke in $G = 16$ bitov za potrebe samega protokola, recimo za preverjanje parnosti, $F = D + G$.

Izkoriščenost E definiramo z razmerjem med časom T_D, ko kanal prenaša koristne podatke in celotnim časom, ki je potreben za prenos teh podatkov T_S,

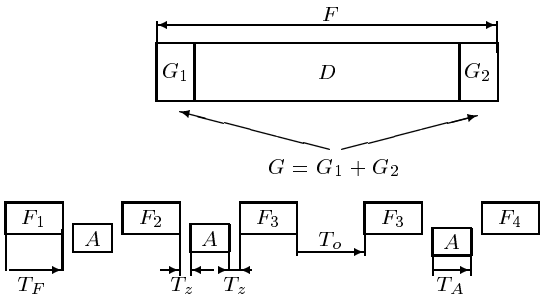
$$E = \frac{T_D}{T_S}.$$

Če napak ne bi bilo, bi bil čas T_S enak seštevku časov trajanja okvirja T_F, trajanja potrdila T_A in trajanja dveh kasnitev T_z, glej sliko 69: $T_S = T_F + T_z + T_A + T_z$. Predpostavljamo, da prenašamo podatke z največjo možno hitrostjo, ki jo dovoljuje kanal. Potem imamo:

$$E = \frac{D/C}{F/C + A/C + 2T_z} = \frac{D}{(D+G) + A + 2 \times T_z \times C}.$$

Za izbrane vrednosti je izkoriščenost kanala $E = \frac{496}{512+8+2\times 0.0002\times 9600} \approx 0.94685$. Izkoriščenost kanala ne bi bila 100% niti tedaj, ko ne bi bilo napak.

V primeru, da se zgodi napaka na okvirju (ali potrdilu), je kanal neizkoriščen čas $T_1 = T_F + T_o$. To drži, če v naslednjem poskusu prenos okvirja uspe. V nasprotnem primeru, ko se ta isti okvir pokvari še drugič, izgubimo čas $T_2 = 2 \times (T_F + T_F)$. Naj bo verjetnost neuspešnega prenosa okvirja (verjetnost napake med prenosom) enaka $p = 10^{-3}$. Z drugimi besedami, od 1000 uspešnih prenosov je eden prenos neuspešen. Verjetnost, da prenašamo isti okvir dvakrat, je enaka produktu $p \times (1-p)$. Torej produktu verjetnosti, da je prvi prenos neuspešen (p) in da je drugi prenos, ki pa je hkrati tudi zadnji, uspešen ($1-p$). Verjetnost, da prenašamo isti okvir k-krat pa je enaka verjetnosti, da po $(k-1)$ ponesrečenih



Slika 69: Izgled okvirja (zgoraj) in primer komunikacije (spodaj).

poskusih prenos končno uspe. Ta verjetnost je $p^{k-1} \times (1-p)$. Povprečno število prenosov na okvir je:

$$\overline{k} = \sum_{k=1}^{\infty} k \times p^{k-1} \times (1-p) = (1-p) \sum_{k=1}^{\infty} k \times p^{k-1} = (1-p)\frac{1}{(1-p)^2} = \frac{1}{1-p}.$$

Povprečno število ponesrečenih prenosov je za ena manjše, $\overline{k} - 1 = \frac{1}{1-p} - 1 = \frac{p}{1-p}$, povprečno trajanje ponavljanja (v bitih) pa je $\frac{p}{1-p} \times (F + T_o \times C)$. Dobljeni rezultat upoštevamo v imenovalcu izraza za izkoristek,

$$E = \frac{D}{(F + A + 2 \times T_z \times C) + \frac{p}{1-p}(F + T_o \times C)}.$$

Ko ni napak, je $p = 0$ in ta izraz je enak prvotnemu izrazu za E. Za iztek časa T_o je smiselno izbrati malenkost daljši čas od časa, ki je skupaj s kasnitvijo potreben za potrdilo, $T_o \times C \approx A + 2 \times T_z \times C$. Izraz za izkoristek kanala dopolnimo z zadnjo ugotovitvijo,

$$E = \frac{D \times (1-p)}{p \times (F + T_o C) + (1-p) \times (F + T_o C)} = \frac{D \times (1-p)}{F + T_o C} = (1-p) \times \frac{D}{D+G} \times \frac{1}{1 + \frac{T_o C}{D+G}}.$$

Prvi člen prispevajo napake in linearno zmanjšuje izkoristek E. Drugi člen je posledica odvečnosti znotraj okvirja. Na G običajno ne moremo kaj dosti vplivati, lahko pa večamo dolžino podatkovnega dela okvirja D (in s tem F) ter tako večamo iskoristek. V zadnjem členu se kaže odvisnost izkoriščenosti kanala od potrjevanja s čakanjem. Za naš primer je:

$$E = (1 - 10^{-3}) \times \frac{496}{512} \times \frac{1}{1 + \frac{11.84}{512}} = 0.999 \times 0.96875 \times 0.9774 = 0.9459.$$

Vidimo, da morebitno ponavljanje okvirjev zaradi prisotnosti napak pri verjetnosti neuspelega prenosa $p = 10^{-3}$ zanemarljivo malo vpliva na zmanjšanje izkoriščenosti.

Sedaj pa poskusimo pri danih lastnostih kanala (pri dani verjetnosti napake bita p_b in pri dani kapaciteti kanala C) določiti optimalno dolžino okvirja, torej

tako dolžino, ki zagotavlja najboljši izkoristek. Gotovo je pri dani verjetnosti napake na bitu verjetnost napake na okvirju podatkov tem večja, čim daljši je okvir. Po drugi strani pa moramo zaradi napak okvirje večkrat ponavljati. Poiščimo najprej zvezo med verjetnostjo napake na okvirju in verjetnostjo napake na bitu. Da je prenos okvirja pravilen, morajo biti pravilno prenešeni prav vsi biti. Verjetnost, da se to zgodi, je $p_u = (1 - p_b)^F$. Ker lahko napaka nastane tudi na potrdilu, je verjetnost uspešnega prenosa enega okvirja enaka

$$(1 - p_b)^F \times (1 - p_b)^A = (1 - p_b)^{(F+A)}.$$

Recimo za naš primer, ko smo privzeli za verjetnost neuspelega prenosa okvirja $p = 10^{-3}$, je verjetnost napake bita

$$1 - 10^{-3} = (1 - p_b)^{520} \rightarrow p_b \approx 1.9 \times 10^{-6},$$

kar je tipična vrednost za verjetnost napake bita na najetem telefonskem vodu. Izkoriščenost kanala v odvisnosti od verjetnosti napake bita in dolžine podatkovnega dela okvirja je:

$$\begin{aligned} E &= (1 - p_b)^{(G+A)} \times (1 - p_b)^D \times \frac{D}{D + G + T_o C} \\ &= (1 - p_b)^{(G+A)} \times (1 - p_b)^D \times \frac{1}{1 + G + T_o C D^{-1}} \\ &= K_1 \times (1 - p_b)^D \times \frac{1}{1 + K_2 D^{-1}} \end{aligned}$$

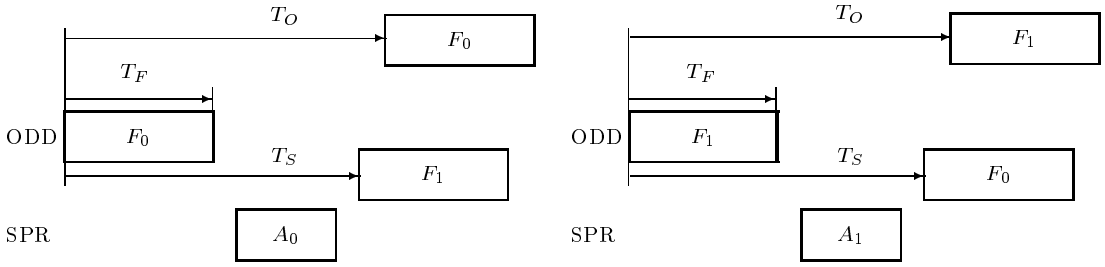
Ta izraz odvajamo po D in izenačimo z nič, $\frac{\partial E}{\partial D} = 0$ ter izračunamo optimalno dolžino podatkovnega dela okvirja v odvisnosti od verjetnosti napake bita. Sami se lahko prepričate, da je

$$D_{opt} \approx \frac{1}{\sqrt{p_b}} \times \sqrt{G + T_o C}.$$

To pomeni, da optimalna dolžina z naraščanjem verjetnosti napake bita hitro upada. Za naš primer je $D_{opt} \approx \sqrt{\frac{16+11.84}{210^{-6}}} \approx 3730$ in optimalna izkoriščenost je $E_{opt} = (1 - 0.0000019)^{3730+16+8} \times \frac{3730}{3730+16+11.84} \approx 0.985$.

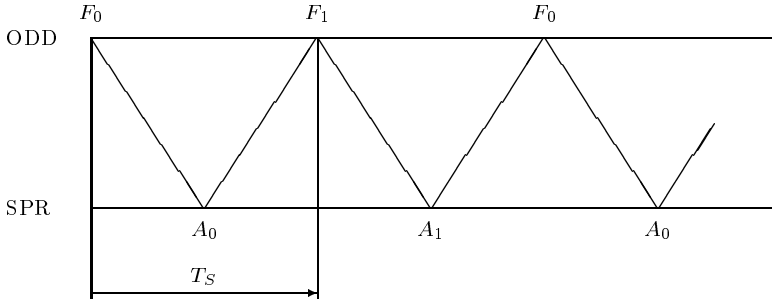
4.3.2 Izkoristek protokola ABP

Izračunajmo izkoristek protokola ABP (Alternating Bit Protocol) s pozitivnim potrjevanjem (pozitivnim odgovorom), ki uporablja nič/ena številčenje okvirjev in potrdil. Delovanje protokola je skicirano na sliki 70.



Slika 70: Protokol ABP (nič/ena številčenje s pozitivnim potrjevanjem).

Obravnava protokola bo podobna predhodni, vendar nas bo zdaj zanimalo samo to, koliko na izkoriščenost kanala vpliva protokol (pozitivno potjevanje s čakanjem). Odvečnosti znotraj okvirja ne bomo upoštevali, ker je ne pripisujemo samo obravnavanemu protokolu (tudi drugi protokoli vnašajo redundanco znotraj okvirja). Model, na katerem bomo zasnovali izračun, prikazuje slika 71.

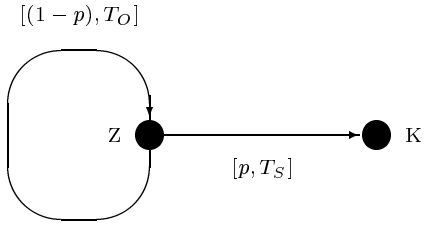


Slika 71: Model za analizo protokola ABP.

Slika prikazuje dogajanje med oddajnikom in sprejemnikom. "Cik-cak" črta ponazarja potovanje okvirjev in potrdil med oddajnikom in sprejemnikom. S T_S je označen čas, ki preteče od oddaje prvega bita okvirja do sprejema zadnjega bita potrdila, skupaj z vmesnimi kasnitvami, ki jih vnašajo kanal, oddajnik in sprejemnik. Po času T_S je možna oddaja naslednjega okvirja. Oddajnik nastavi časovnik na začetku oddaje prvega bita okvirja. Po času T_O se časovnik izteče. Če v tem času oddajnik ne dobi potrdila, ponovi isti okvir. Izkoriščenost definira naslednji obrazec:

$$E_{ABP}(p) = \frac{T_F}{\bar{T}}.$$

Pri tem je T_F čas trajanja okvirja, $\bar{T}$ je čas, ki je v povprečju potreben za prenos enega okvirja in p je verjetnost napake na okvirju (ali potrdilu). Izkoristek ne bomo računali po dolgem postopku, ki smo se ga poslužili v prejšnjem razdelku, ampak bomo skušali računati s čim manj truda. Pri tem nam bo pomagala skica 72.



Slika 72: Verjetnostni model za določitev časa $\bar{T}$, ki je v povprečju potreben za prenos enega okvirja.

Prenos okvirja od oddajnika do sprejemnika smo ponazorili kot problem prehoda iz začetnega stanja Z v končno stanje K. V začetnem stanju smo od trenutka, ko začnemo z oddajanjem okvirja, dokler nam poskus prenosa tega okvirja ne uspe. Z verjetnostjo $(1-p)$ nam prenos uspe in s to verjetnostjo se selimo v končno stanje. To traja čas T_S. Ta zadnji prehod prispeva k povprečnemu času $(1-p) \cdot T_S$. Verjetnost neuspešnega poskusa prenosa je p. Pri vsakem neuspelem poskusu izgubimo čas T_O. Vsak neuspel poskus nam povprečni čas poveča za T_O. To se zgodi z verjetnostjo p. Povprečen čas, da pridemo iz začetnega v končno stanje je:

$$\bar{T} = p \cdot (T_O + \bar{T}) + (1-p) \cdot T_S$$

in ko izrazimo $\bar{T}$, dobimo:

$$\bar{T} = T_S + \frac{p}{1-p} T_O.$$

Izkoristek v odvisnosti od verjetnosti napake je:

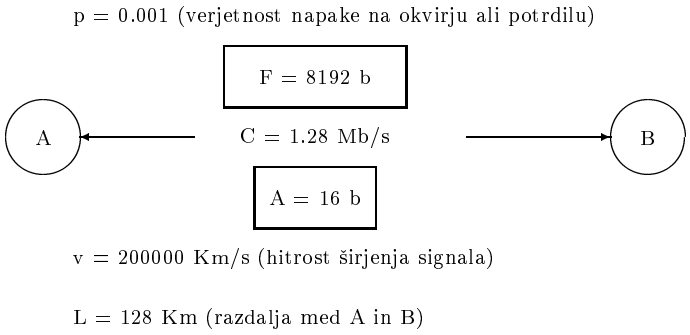
$$E_{ABP}(p) = \frac{T_F}{T_S + \frac{p}{1-p} T_O}.$$

Ker je v praksi $T_O \approx T_S$, je $\bar{T} = \frac{1}{1-p} T_S$ in

$$E_{ABP}(p) = (1-p)\frac{T_F}{T_S},$$

ali če upoštevamo posamezne prispevke k času T_S:

$$E_{ABP}(p) = (1-p)\frac{F}{F + A + 2 \cdot T_z \cdot C}.$$



Slika 73: Podatki za izračun izkoristka protokola ABP.

Izkoristek pada z verjetnostjo napake, s hitrostjo prenosa (oziroma s kapaciteto kanala C) in s kasnilnim časom, ki je odvisen tudi od razdalje med oddajnikom in sprejemnikom, kot smo enkrat že ugotovili.

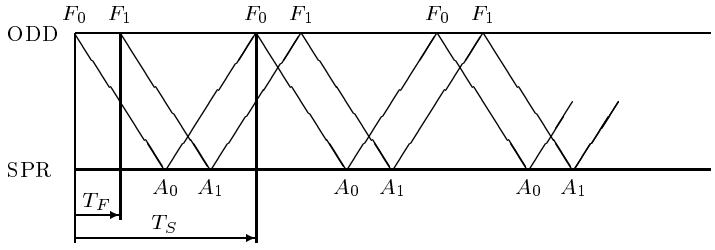
Za primer vzemimo naslednje podatke, glej sliko 73.

Izkoristek je:

$$E_{ABP} = 0.999 \frac{8192}{8192 + 16 + 2 \cdot 64 \cdot 10^{-5} \cdot 1.28 \cdot 10^{6}} = 0.83$$

4.3.3 Analiza protokola GBN

Analizirajmo izkoristek protokola z vračanjem nazaj na N (GBN) in z drsečim oknom. Po tem protokolu oddaja oddajnik brez premorov, če le dobi pravočasno potrdilo za oddani okvir. Razmere, ko ni napak, prikazuje slika 74. Slika prikazuje dogajanja za okno velikosti $w = 2$. Oddajnik odda dva okvirja F_0 in F_1, potem pa mora počakati na potrdilo.



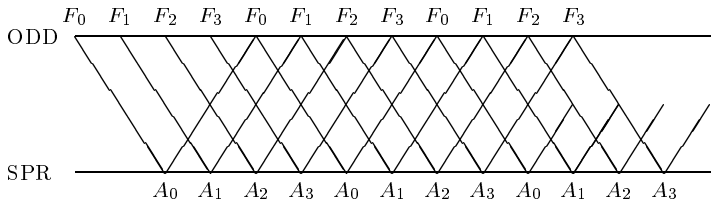
Slika 74: Vračanje na N za $w = 2$.

V času T_S po oddaji prvega bita okvirja F_0 pride potrdilo zanj (A_0) v celoti nazaj do sprejemnika. Po tem času je možna oddaja tretjega okvirja, ki je označen enako kot prvi okvir s F_0 in ker je medtem prišlo tudi potrdilo A_1 za drugi okvir,

lahko takoj po oddaji tretjega okvirja odda še četrtega, ki je ponovno označen s F_1. Zatem sledi spet čakanje na potrdilo. V času T_S je oddajnik oddal $w = 2$ okvirja. Izkoristek je:

$$E_{GBP}(p = 0) = \frac{w \cdot T_F}{T_S}.$$

Na sliki 75 smo predpostavili, da je velikost okna $w = 4$ izbrana tako, da je čas povratka potrdila ravno tak, da pride potrdilo nazaj tik predno bi oddajnik moral prekiniti z oddajanjem, $T_S = w \cdot T_F$. Izkoristek je za tako izbiro okna enak ena.



Slika 75: Vračanje na N za $w = 4$ in $wT_F = T_S$).

Izkoristek bi bil ena tudi v primeru, da bi potrdilo prišlo že prej. Manjši od ena je izkoristek v primeru premajhnega okna. Velja,

$$E_{GBN}(p = 0) = \min\{1, w\frac{T_F}{T_S}\},$$

medtem ko je izkoristek protokola ABP enak

$$E_{ABP}(p = 0) = \frac{T_F}{T_S}.$$

Razmere, ki smo jih skicirali na sliki 75, se ohranjajo, dokler ne pride do prve napake. Slika 76 prikazuje primer, ko se zgodi napaka na okvirju s številko F_0, vendar bi si lahko zamislili napako na kateremkoli okvirju, saj način številčenja do prve napake ne vpliva na izkoristek. Ker po času T_O oddajnik ne dobi potrdila, ponovno oddaja okvir F_0 in tudi vse tiste, ki jih je med tem že oddal. V primeru napake se zato izgubi čas T_O.

Izračunajmo povprečni čas za prenos enega okvirja. V primeru, da ni napake, se za prenos okvirja porabi čas T_F. To se zgodi z verjetnostjo $(1 - p)$. V primeru, da pride do napake, se izgubi čas T_O in okvir še ni prenešen. To se zgodi z verjetnostjo p. V primeru, da bi bila možna na istem okvirju samo enkratna napaka, bi bil povprečni čas za prenos okvirja enak $(1 - p)T_F + p(T_O + T_F)$. Ker pa se lahko na istem okvirju napaka ponovi poljubno mnogokrat, postane izraz za povprečni čas prenosa enega okvirja podoben tistemu pri analizi protokola ABP:

$$\bar{T} = (1 - p)T_F + p(T_O + \bar{T})$$

in

$$\bar{T} = T_F + \frac{p}{1-p} T_O.$$

Končno z upoštevanjem $T_O \approx T_S$ in $wT_F = T_S$, sledi:

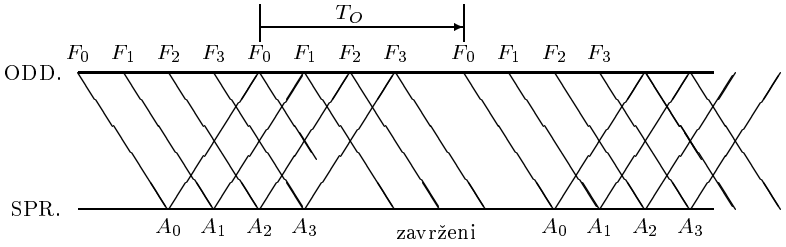
$$E_{GBN}(p) = \frac{T_F}{T_F + \frac{p}{1-p} w T_F} = \frac{1}{1 + \frac{p}{1-p} w}$$

Ker je verjetnost napake p majhna, je $(1-p)$ približno ena,

$$E_{GBN}(p) = \frac{1}{1+pw}$$

Če privzamemo, da je v praksi pw tudi majhen, je

$$E_{GBN}(p) \approx 1 - wp = 1 - p\frac{T_S}{T_F}.$$



Slika 76: Vračanje na N v primeru napak.

Za številčni primer izberimo enake podatke kot za protokol ABP, le hitrost prenašanja naj bo višja, in sicer $C = 128 Mb/s$. Pri tej hitrosti traja okvir $T_F = F/C = 8192/128 \times 10^{-6} = 128\mu s$, čas povratka potrdila pa je $T_S = F/C + A/C + 2 \cdot T_z = 128 + 0.125 + 1280 \approx 1408\mu s$. Izberemo velikost okna $w = 11$ in izkoristek je:

$$E_{GBN} = 1 - 0.001 \times \frac{1408}{128} = 1 - 0.011 = 0.989,$$

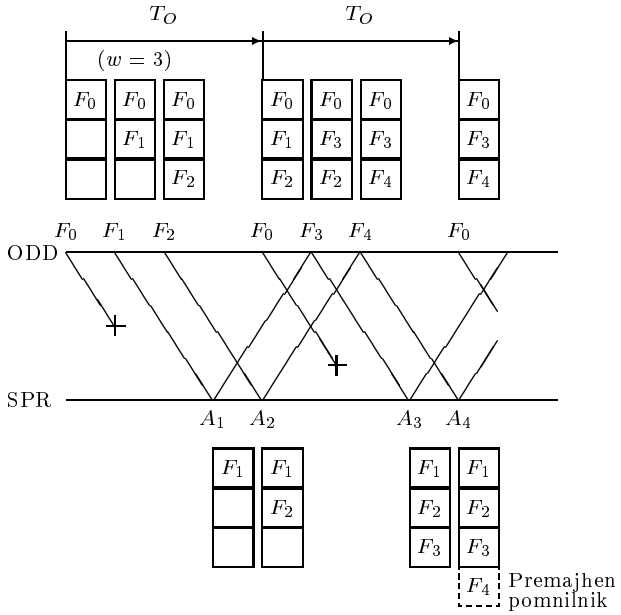
za protokol ABP pa samo:

$$E_{ABP} = 0.999 \frac{128}{1408} = 0.091$$

Protokol ABP je primeren le za nizke hitrosti in majhne razdalje.

4.3.4 Analiza protokola SRP

Videli bomo, da je določanje izkoristka protokola SRP bolj zahtevno. Predno pa se lotimo računanja izkoristka, se prepričajmo, da je za protokol SRP z velikostjo okna w potrebno številčiti okvirje po modulu $2w$ in ne po modulu w kot pri protokolu GBN. Nadalje, velikost okna je v tem primeru definirana z razliko med številko oddanega in številko potrjenega okvirja (številčeno po modulu $2w$), in ne s številom oddanih okvirjev, ki čakajo na potrdilo.



Slika 77: Selektivno ponavljanje in definicija velikosti okna (primer večkratne napake na okvirju F_0).

Najprej dokažimo, da drži zadnja trditev. Dokaz ne bo veliko trpel na splošnosti, če za velikost okna izberemo neko konkretno vrednost. Naj bo $w = 3$. Velikost oddajnega in sprejemnega pomnilnika je enaka velikosti okna. Razmere prikazuje slika 77. Zaenkrat okvirjev ne številčimo po modulu $2w$, kot smo trdili, da bi sicer zadostovalo, ampak si privoščimo številčenje kar od začetka do konca: $0, 1, 2, 3, 4, 5, 6, \ldots$. Zamislimo si, da sprejemnik prekine z oddajanjem takrat, kadar je oddal največ tri okvirje po zadnjem potrjenem potrdilu. Torej ne upošteva zahteve po razliki med številko oddanega okvirja in sprejetega potrdila. To bi moralo povzročiti težave.

Na sliki smo si zamislili napako na okvirju z zaporedno številko nič (F_0). Predno poteče čas čakanja na potrdilo, oddajnik odda še dva okvirja, ker je $w = 3$. Oddajnik hrani oddane in še ne potrjene okvirje v vmesnem pomnilniku, dokler ne dobi potrdil, ker bo morda moral katerega od okvirjev ponoviti. Po času T_O ne dobi potrdila A_0 in ponovno odda okvir (F_0), ki pa že spet ne bo prišel brez napake

do sprejemnika. Med tem je sprejemnik potrdil in tudi shranil okvirja F_1 in F_2. Ko oddajnik za ponovljenim prenosom okvirja F_0 dobi potrdilo za F_1, sprosti njegov medpomnilnik, vanj shrani okvir F_3 in ga odda. (Pozor, številka oddanega in še ne potrjenega okvirja se sedaj že razlikuje za več kot je velikost okna). Ko dobi potrdilo še za okvir F_2, sprosti njegov medpomnilnik, vanj vpiše okvir F_4 in ga odda. Med tem je medpomnilnik sprejemnika že poln, vendar medpomnjenih okvirjev ne more dati naprej, ker še vedno čaka na okvir F_0 in bi bilo zaporedje predanih okvirjev nepravilno. Da bi zagotovil zahtevi po pravilnem vrstnem redu (zaporednosti) okvirjev, bi mu morali povečati medpomnilnik. Teoretično bi moral biti medpomnilnik neskončen, kar je v praksi nemogoče. Končna velikost pa bi zadoščala, če bi upoštevali zahtevo po največji razliki številk okvirjev in potrdil.

Prepričajmo se še v potrebnost (in zadostnost) številčenja okvirjev in potrdil po modulu $2w$. Naj bo N zaporedna številka okvirja (številčeno od začetka do konca sporočila), ki ga je sprejemnik kot zadnjega predal naprej (konkretno mrežnemu sloju). Torej so bili dostavljeni tudi vsi prejšnji okvirji, medtem ko okvirja s številko $N+1$ še ni potrdil, ker bi ga sicer tudi že dal naprej. Oddajnik je med tem oddal največ še $n+w$-ti okvir, ker mora biti razlika med številko oddanega in številko potrjenega okvirja manjša ali enaka w. Oddajnik je med tem moral že dobiti potrdila za vse okvirje vključno z $N-w$-tim okvirjem, ker bi sicer ne oddal okvirja N. Iz tega sledi, da lahko sprejemnik dobi kot naslednji okvir okvir s številko iz množice

$$\{N-w+1, N-w+2, \ldots, N-1, N, N+1, \ldots, N+w-1, N+w\}$$

Pri tem pa tudi ve, da števila

$$\mathrm{STAR} = \{N-w+1, N-w+2, \ldots, N-1, N\}$$

pomenijo številke starih okvirjev, ki jih je že potrdil in tudi dal naprej. Take okvirje zavrže in vseeno potrdi. Če pa dobi okvir s številko

$$\mathrm{NOV} = \{N+1, \ldots, N+w-1, N+w\}$$

je lahko to samo okvir, ki ga je že uspešno sprejel in se zato nahaja v njegovem medpomnilniku ali pa ga še ni sprejel. Če še ni v pomnilniku ga shrani, sicer pa zavrže, v vsakem primeru pa ga potrdi. Vidimo, da je vseh potencialno možnih okvirjev $2 \cdot w$, ti pa bodo različno številčeni, če številčimo po modulu $2 \cdot w$. Sprejemnik ne bo pri takem načinu številčenja zamenjal starega okvirja z novim, kar bi vodilo v podvajanje ali izgubljanje okvirjev. Sprejemnik ne more zgrešiti.

Poglejmo primer za $w = 4$, ko zahtevamo številčenje po modulu $2w = 8$. Naj bo zaporedna številka zadnje predanega okvirja enaka $N = 121$. Pri številčenju po modulu 8 bo številka tega okvirja $121/8 = 5$. Sprejemnik lahko sprejme naslednje stare okvirje

$$\mathrm{STAR} = \{118, 119, 120, 121\}$$

ki so označeni z

$$\{2, 3, 4, 5\}$$

in naslednje nove okvirje

$$\text{NOV} = \{122, 123, 124, 125\}$$

ki so označeni z

$$\{6, 7, 0, 1\}$$

Oznake starih in novih okvirjev se res razlikujejo.

Obstaja še vprašanje, ali lahko tako številčenje zmede oddajnik. Oddajnik je gotovo že dobil vsa potrdila do vključno A_{n-4}, saj je že oddal okvir s številko N, vendar pa zagotovo še ni oddal nobenega okvirja za okvirjem s številko $N+4$. Torej morajo biti potrdila znotraj množice

$$\{N-3, N-2, \ldots, N, N+1, \ldots, N+4\}$$

Tudi ta števila so različna, če za velikost okna $w = 4$ številčimo po modulu osem.

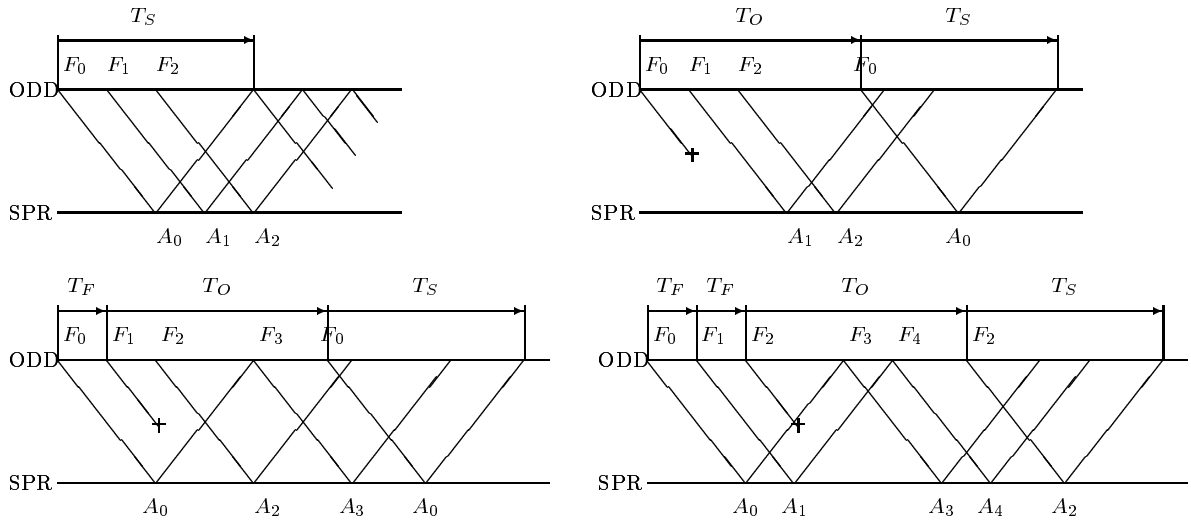
Izkoristek Izračunajmo izkoristek protokola SRP. Pri tem nam bo pomagala slika 78. V slučaju, da ni napak, oddamo v času T_S ravno w okvirjev. Zato je kanal izkoriščen čas $w \cdot T_F$ (v narisanem primeru je izbrana velikost okna $w = 3$). Izkoristek je:

$$E_{SRP}(p = 0) = \min\{1, w \cdot \frac{T_F}{T_S}\},$$

enako kot za protokol GBN. Predno se lotimo splošnega računanja izkoristka z upoštevanjem napak, si problem poenostavimo s predpostavko, da sta medpomnilnika oddajnika in sprejemnika neskončna (tako velika, da ni potrebno zaradi pomankanja pomnilnika nikoli čakati). V tem primeru bi prenašali okvirje drugega za drugim, brez premorov. Zaradi napak bi nekatere okvirje prenašali večkrat. Ponavljanje bi pomenilo izgubo časa. Čbi ne bilo napak, bi vsak okvir prenašali samo enkrat in izkoristek bi bil ena. V primeru, da bi bila verjetnost napačnega prenosa $p = 0.5$, bi v povprečju okvir prenašali dvakrat, povprečno število prenosov istega okvirja je namreč enako $\frac{1}{1-p}$. Do tega pridemo na enak način, kot smo to že naredili v prejšnjih razdelkih. Izkoristek protokola SRP z neskončnim oknom ($w = \infty$) je:

$$E_{SRP,w=\infty}(p) = \frac{T_F}{\frac{1}{1-p}T_F} = 1 - p.$$

To je zgornja meja izkoristka v odvisnosti od verjetnosti napake. Zaradi končne velikosti okna bo izkoristek v stvarnih razmerah manjši. Izračunajmo ga. Da bo problem lažje rešljiv, bomo upoštevali, da je verjetnost ponovne napake istega



Slika 78: Primeri razmer za protokol SRP: brez napake, napaka na prvem, na drugem ali na tretjem okvirju (za $w = 3$).

okvirja zanemarljiva. Podobno, verjetnost napake na dveh okvirjih znotraj okna, naj je tudi zanemarljiva.

Za velikost okna $w = 3$ obstajajo v tem primeru tri možnosti napak, napaka na prvem, na drugem ali na tretjem okvirju, glej sliko 78. V primeru napake na prvem okvirju (F_0), prenesemo v času $t_0 = T_O + T_S$ tri okvirje, $n_0 = 3$. V primeru napake na drugem okvirju prenesemo v času $t_1 = T_F + T_O + T_S$ štiri okvirje, $n_1 = 4$. Podobno prenesemo pri napaki na tretjem okvirju v času $t_2 = 2 \cdot T_F + T_O + T_S$ pet okvirjev, $n_2 = 5$. Splošen izraz za t_i in n_i v odvisnosti od velikosti okna, je:

$$n_i = w + i, \qquad t_i = T_O + T_S + i \cdot T_F \quad (i = 0, 1, 2, \ldots, w - 1),$$

Pri pogoju $T_S \approx T_O$ in $T_S = w \cdot T_F$ je

$$t_i = 2 \cdot w \cdot T_F + i \cdot T_F = T_F \cdot (2 \cdot w + i).$$

Verjetnost napake na enem izmed treh okvirjev je

$$p_i = p \cdot (1 - p)^2 \approx p$$

in je neodvisna od številke okvirja, na katerem se zgodi napaka. Verjetnost, da ni napake, pa je $p_n(1 - p)^3 \approx 1 - 3p$, ali v splošnem $1 - wp$. V tem primeru prenesemo v času $t_w = T_S$ tri okvirje, $n_w = 3$. Če zanemarimo možnost, da se zgodi napaka na dveh okvirjih, so to tudi vsi možni dogodki. Čase t_i in število prenešenih okvirjev n_i lahko obravnavamo kot diskretni naključni spremenljivki T in N, katerih zalogo vrednosti in porazdelitveni zakon smo pravkar ugotovili,

$$T = \begin{pmatrix} t_0, & t_1, & t_2, & \ldots, & t_w \\ p, & p, & p, & \ldots, & 1 - wp \end{pmatrix}.$$

$$N = \begin{pmatrix} n_0, & n_1, & n_2, & \ldots, & n_w \\ p, & p, & p, & \ldots, & 1-wp \end{pmatrix}.$$

Izračunajmo povprečne vrednosti naključnih spremenljivk T in N,

$$\begin{aligned}
\bar{T} &= \sum_{i=0}^{i=w} p_i \cdot t_i \\
&= T_F[\cdot \sum_{i=0}^{i=w-1} p \cdot (2 \cdot w + i) + (1 - wp) \cdot w] \\
&= T_F[p \cdot 2w^2 + p\frac{(w-1) \cdot w}{2} + w - p \cdot w^2] \\
&= w \cdot T_F[1 + \frac{p}{2}(3w - 1)].
\end{aligned}$$

$$\begin{aligned}
\bar{N} &= \sum_{i=0}^{i=w} p_i \cdot n_i \\
&= \sum_{i=0}^{i=w-1} p \cdot (w + i) + (1 - wp)w \\
&= p \cdot w^2 + p\frac{w(w-1)}{2} + w - p \cdot w^2 \\
&= w + p\frac{w(w-1)}{2}.
\end{aligned}$$

Izkoristek pa je:

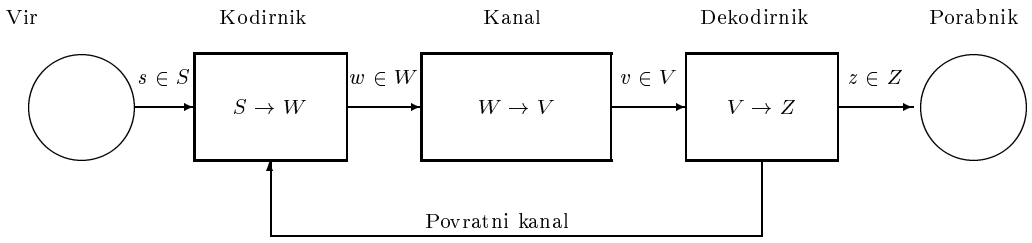
$$E_{SRP}(p) = \frac{T_F \bar{N}}{\bar{T}} = \frac{2 + p(w-1)}{2 + p(3w-1)}.$$

4.4 Odkrivanje in popravljanje napak

Pri prenosu podatkov se *napakam* ne moremo nikoli popolnoma izogniti. Podatki, ki jih sprejme sprejemna naprava, se zaradi nepopolnosti prenosnih naprav ter zunanjih motilnih vplivov, ki se s časom nakljućno spreminjajo, v splošnem razlikujejo od oddanih podatkov. Pravimo, da prenos podatkov ni popolnoma zanesljiv. Zanesljivost prenosa lahko povečamo na več načinov. Na primer tako, da za prenos uporabimo kvalitetnejše (dražje) prenosne poti. Zanesljivost prenosa se da vedno povečati na račun nižje hitrosti. Pri nižji hitrosti prenosa oddani signal dalj časa vstraja v enem od možnih stanj, zato ga sprejemna naprava lažje loči od nakljućnih stanj šuma. Tretjo možnost za povečanje zanesljivosti prenosa daje *kodiranje*. Če povzamemo, zanesljivost prenosa je odvisna od kvalitete prenosne poti, od hitrosti prenosa in načina kodiranja.

V tem podpoglavju si bomo ogledali osnovne postopke kodiranja, ki omogočajo, da v danih pogojih (pri dani prenosni poti) z *odkrivanjem* in redkeje s *popravljanjem* napak povečamo zanesljivost prenosa. Kljub številnim študijam kodov za popravljanje napak se v praksi več uporabljajo postopki za odkrivanje napak, skupaj z enim od mehanizmov (protokolov) za potrjevanje pravilno ali napačno sprejetih podatkov s strani sprejemne naprave.

Zamisel odkrivanja in popravljanja napak bomo razložili na modelu komunikacijskega sistema, ki je narisan na sliki 79.



Slika 79: Model komunikacijskega sistema. Informacija ($z \in Z$), ki prispe do porabnika informacije ni nujno enaka informaciji, ki jo ustvari vir $s \in S$. S primernim načinom kodiranja, prenašanja in dekodiranja pa skušamo doseči, da bi bila ($z = s$).

Informacijski vir ustvarja informacijo, oddaja simbole iz končne zaloge S. To informacijo kodirna naprava (za)kodira in pošlje v *informacijski kanal*. Zaradi napak med prenosom sprejeti podatki $v \in V$ niso vedno enaki oddanim podatkom $w \in W$. Dekodirna naprava sprejete podatke V dekodira in da porabniku informacijo $z \in Z$. Informacija, ki prispe do porabnika informacije, zaradi napak ni nujno enaka informaciji, ki jo ustvari vir. S primernim načinom kodiranja, pošiljanja in dekodiranja pa skušamo to doseči.

Praktično vsi postopki za odkrivanje ali popravljanje napak temeljijo na vnašanju odvečnosti (redundance) v sporočila. Informacijskim simbolov $i_1, i_2, \ldots, i_k$, ki nosijo informacijo vira S, se na oddajni strani po nekem pravilu f_α določi (izračuna) odvečne simbole $o_{k+1}, o_{k+2}, \ldots, o_n$,

$$o_\alpha = f_\alpha(i_1, i_2, \ldots, i_k) \quad (\alpha = k+1, k+2, \ldots, n)$$

nakar se vse skupaj pošlje v kanal,

$$w = i_1, i_2, \ldots, i_k, o_{k+1}, o_{k+2}, \ldots, o_n.$$

Sprejemna naprava sprejme zaporedje simbolov w in po enakem pravilu kot oddajna naprava izračuna odvečne simbole, te pa primerja s sprejetimi. V primeru, da se izračunano zaporedje ne ujema s sprejetim, pomeni to napako oziroma točneje, da je napaka odkrita. Nekateri načini kodiranja omogočajo, da sprejemna naprava na osnovi izračuna določi mesto napake in jo tudi popravi. V primeru, da sprejemna naprava napake ne zna popraviti, pa zahteva ponovitev prenosa.

Osnovni parametri, s katerimi vrednotimo uspešnost kodirnega sistema so koristnost, odvečnost in zanesljivost.

Koristnost koda E je definirana z razmerjem med številom informacijskih simbolov k in številom vseh simbolov n (informacijskih in odvečnih),

$$E = \frac{k}{n}.$$

Odvečnost koda R je podana z razmerjem med številom odvečnih ali kontrolnih simbolov in številom vseh prenašanih simbolov,

$$R = \frac{n-k}{n} = 1 - E.$$

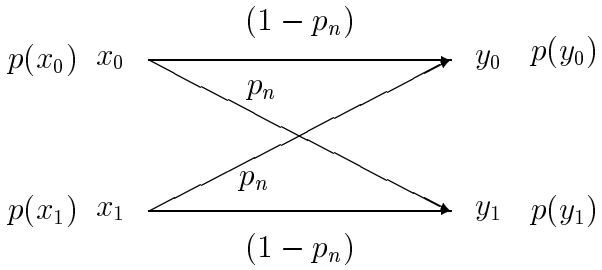
Odvečnost in koristnost sta običajno, a ne nujno, povezana z zmožnostjo koda za odkrivanje in/ali popravljanje napak. Večja odvečnost omogoča s stališča odkrivanja in popravljanja boljše kode.

Zanesljivost prenosa je definirana z razmerjem med številom pravilno prenešenih simbolov in številom vseh simbolov, ki jih prenašamo.

4.4.1 Osnovna zamisel odkrivanja in popravljanja napak

Da bi razložili zamisel odkrivanja se prej spoznajmo z verjetnostnim modelom kanala. Napake so posledica motenj, motnje pa so naključne narave. Na manj

kvalitetnih prenosnih poteh so motnje in s tem napake bolj pogoste ali bolj *verjetne* kot na kvalitetnih prenosnih poteh. Verjetnost napake na simbolu, na skupini simbolov ali na celemu sporočilu, je eden od parametrov, ki podajajo lastnost prenosne poti. Slika 80 prikazuje verjetnostni model binarnega simetričnega kanala. Kanal je v tem primeru določen z verjetnostjo napake med prenosom, ostale značilnosti kanala (prenosni medij, oblika signala, namen uporabe) pa nas ne zanimajo. Na vhodu sta dovoljena dva simbola, to sta x_0 in x_1. Na izhodu sta možna dva simbola, y_0 in y_1. Zato je kanal *binaren*. Verjetnost, da se simbol x_0 zaradi napake spremeni v y_1 je enaka verjetosti, da se simbol x_1 spremeni v y_0. Zato je kanal *simetričen*.



Slika 80: Verjetnostni model binarnega simetričnega kanala z verjetnostjo napake na simbolu p_n. Za idealen kanal je $p_n = 0$, za neuporaben kanal je verjetnost napake 0.5. Na primer, za telefonski vod naj bi bila verjetnost napake približno 10^{-4}.

Sedaj pa si mislimo vir informacije, ki lahko zavzame eno od dveh enakoverjetnih stanj, s_0 in s_1. Priredimo tem dvem stanjem binarna simbola 0 in 1, na primer $s_0 \to w_0 = 0$ in $s_1 \to w_1 = 1$. Vzemimo, da zaporedja tako kodiranih stanj (sporočila) pošiljamo po binarnem simetričnem kanalu do sprejemnika. Naj bo verjetnost napake na kanalu $p_n = 0.1$. Z drugimi besedami, od 1000 oddanih simbolov se bo med prenosom 100 od njih spremenilo iz 1 v 0 ali iz 0 v 1, ne da bi na sprejemni strani za to vedeli. Ker simbol 0 pomeni stanje s_0, simbol 1 pa stanje s_1, bomo napačno razpoznali 100 stanj. Pri takem načinu kodiranja ne bo moč odkriti niti ene same napake.

Sedaj pa priredimo (zakodirajmo) vsakemu od stanj s_0 in s_1 po dva binarna simbola, in sicer: $s_0 \to w_0 = 00$ in $s_1 \to w_1 = 11$. Zaradi napak med prenosom so na izhodu možna vsa zaporedja $v_1 = w_0 = 00$, $v_2 = 01$, $v_3 = 10$, $v_4 = w_1 = 11$. V primeru, da se zgodi napaka na enem samem simbolu, sprejmemo zaporedje v_2 ali v_3. Ker sta na vhodu možni le kombinaciji $w_0 = 00$ in $w_1 = 11$ vemo, da je prišlo do napake. Iz sprejete kombinacije se seveda ne da razbrati, katera od besed w_0 in w_1 je bila oddana. Enako verjetna je napaka na prvem simbolu kot na drugem simbolu. Napake ne znamo popraviti. V primeru napake na dveh

simbolih (dvojne napake), se w_0 spremeni v $v_4 = w_1$ ali w_1 v $v_1 = w_0$. Sprejemnik se take napake ne zaveda. Verjetnost, da se to zgodi je:

$$p_{2n} = p_n \times p_n = 0.1 \times 0.1 = 0.01,$$

verjetnost, da se to ne zgodi pa 0.99. Sprejemnik se odloči za verjetnejšo možnost, ki pa je v 0.01 primerih napačna. To pomeni, da bi od tisoč oddanih stanj bilo deset razpoznanih napačno. Verjetnost, da napaka ni odkrita je za cel velikostni razred manjša kot v prvem primeru. Pri dani hitrosti prenosa podatkov bi v istem času prenesli pol manj informacije. Dejanski informacijski pretok se je zmanjšal na polovico.

Dodajmo še en simbol in zakodirajmo s_0 z $w_0 = 000$ in s_1 z $w_1 = 111$. Zaradi napak med prenosom je na izhodu kanala možno vsako zaporedje treh simbolov. Takih zaporedij je osem, v_i, $(i = 0, 1, \ldots, 7)$. Sprejemnik napake ne odkrije, če se zgodijo tri napake, saj se ena kodna beseda popolnoma spremeni v drugo. Verjetnost, da se to zgodi je

$$p_{3n} = p_n^3 = 0.1^3 = 0.001.$$

To pomeni, da sprejemnik v zaporedju tisočih stanj zagreši samo še eno napako.

Sedaj pa zahtevajmo, da sprejemnik napake ne le odkriva, ampak tudi popravlja. Iz zaporedja treh simbolov se mora sprejemnik odločiti, kaj je oddal oddajnik. Vzemimo, da se odloča takole: če sprejme več ničel kot enic, se odloči za s_0, če sprejme več simbolov 1 pa s_1. V primeru, da je verjetnost napake (spremembe simbola) manjša od 0.5, se zdi tak način odločanja smiseln.

Sprejem	Odločitev	Sprejem	Odločitev
$v_0 = 000$	s_0	$v_1 = 001$	s_0
$v_2 = 010$	s_0	$v_3 = 011$	s_1
$v_4 = 100$	s_0	$v_5 = 101$	s_1
$v_6 = 110$	s_1	$v_7 = 111$	s_1

Pravilno se odločimo vsakič, ko se od treh pokvari samo en simbol. Pravimo, da popravimo vsako enojno napako.

Napaka ni odkrita v primeru, da se na zaporedju treh simbolov zgodita dve ali tri napake, ne glede na način, kako se to zgodi. Izračunajmo verjetnost napačne odločitve. Pričakujemo, da je manjša kot prej, ko je znašala 0.1. Imamo:

$$\begin{aligned} p_{napake} &= 0.1 \times 0.1 \times 0.9 + 0.9 \times 0.1 \times 0.1 + 0.1 \times 0.9 \times 0.1 + \\ &+ 0.1 \times 0.1 \times 0.1 = \\ &= 3 \times 0.1^2 \times 0.9 + 0.1^3 = 0.028. \end{aligned}$$

Vidimo, da se je verjetnost napačne odločitve zmanjšala z 0.1 na 0.028, toda na račun nižje efektivne hitrosti prenosa, ki je trikrat manjša.

V zadnjem primeru smo vsako enojno napako ne le odkrili, temveč tudi popravili. Če bi dolžino kodnih besed še povečali, bi sposobnost odkrivanja in/ali popravljanja napak še povečali. To je razumljivo, saj se daljši kodni besedi bolj razlikujeta in se potemtakem ena beseda težje popolnoma spremeni v drugo. Izgubljamo pa na dejanski hitrosti prenosa.

Število simbolov, na katerih se dve enako dolgi zaporedji binarnih simbolov razlikujeta, imenujemo *Hammingova* razdalja. Na primer, v zadnjem primeru je Hammingova razdalja

$$d(w_0, w_1) = 3.$$

Kod je sposoben odkrivati in popravljati tem večkratne napake, čim večja je Hammingova razdalja med besedami koda. Zato želimo, da bi bila razdalja med besedami čim večja. To se da povečati z dodajanjem simbolov, ki sicer niso potrebni, ne nosijo informacije in so s tega stališča odveč. So pa bistveni za odkrivanje in popravljanje napak.

4.4.2 Hammingova razdalja in zmožnost koda za odkrivanje napak

Vzemimo, da želimo napraviti takšen kod $W = \{w_i\}$, ki je zmožen odkrivati vse napake na enem, na dveh, na e simbolih, ne glede na to, na kakšen način se napake pojavijo. Velja:

$$d(w_i, w_j) \geq e + 1, \tag{20}$$

kjer je $d(w_i, w_j)$ Hammingova razdalja med poljubnima kodnima besedama koda.

V to se prepričajmo na primeru. Vzemimo, da ima kod samo dve kodni besedi dolžine $n = 8$:

$$w_1 = 00000000 \quad in \quad w_2 = 11111111.$$

Razdalja med njima je $d(w_1, w_2) = 8$. Vzemimo, da sprejmemo zaporedje

$$v = 111111110.$$

Zaporedje v je neveljavno - ni kodna beseda. Veljavni sta samo zaporedji w_1 in w_2, sprejeto zaporedje pa ni enako nobeni od besed. Možno je, da je bila oddana beseda w_2, vendar smo zaradi napake na zadnjem simbolu sprejeli v. Manj verjetno, vendar tudi možno pa je, da je bila oddana beseda w_1, zaradi napak na prvih sedmih simbolih ($e = 7$) pa smo sprejeli v.

$$d = 8 = e + 1 \Longrightarrow e = 7.$$

V obeh primerih vemo, da je prišlo med prenosom do napake. V primeru osmih napak bi se ena beseda popolnoma spremenila v drugo, vendar iz sprejetega zaporedja za to ne bi mogli vedeti. V splošnem odkrijemo vsako tako kombinacijo napak, ki ene kodne besede ne spremeni v kakšno drugo kodno besedo. To se ne zgodi, če je najmanjša razdalja med kodnimi besedami vsaj za eno večja od števila napačnih simbolov.

4.4.3 Hammingova razdalja in zmožnost koda za popravljanje napak

Vzemimo, da želimo skonstruirati tak kod $W = \{w_i\}$, ki bi bil sposoben napake ne le odkrivati, ampak tudi popravljati. Velja naslednji izrek. Kod z lastnostjo

$$d(w_i, w_j) \geq 2 \times e + 1 \tag{21}$$

je sposoben odkrivati in popravljati vse e-kratne ali manj-kratne napake. Kod z lastnostjo

$$d(w_i, w_j) \geq 2 \times e \tag{22}$$

pa je sposoben odkrivati in popravljati vse $(e-1)$-kratne ali manj-kratne napake, medtem ko je e-kratne napake sposoben samo odkrivati, popravljati pa ne.

Spet se v to prepričajmo na primeru. Vzemimo, da sta w_i in w_j kodni besedi z najmanjšo razdaljo. Vse druge kodne besede so si bolj oddaljene. Naj bo med njima razdalja $d(w_i, w_j) = 5$. Označimo z v_1, v_2, v_3, v_4 poljubna možna zaporedja, ki se od w_i razlikujejo na enem, na dveh, na treh in na štirih mestih,

$$d(v_1, w_i) = 1,\ d(v_2, w_i) = 2,\ d(v_3, w_i) = 3,\ d(v_4, w_i) = 4.$$

Za razdalje do kodne besede w_j pa naj velja

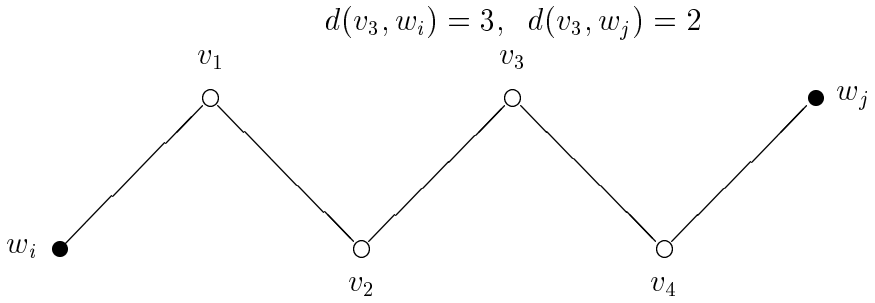
$$d(v_4, w_j) = 1,\ d(v_3, w_j) = 2,\ d(v_2, w_j) = 3,\ d(v_1, w_i) = 4.$$

Razmere so narisane na sliki 81.

Vzemimo, da sprejmemo zaporedje v_3. To zaporedje sprejmemo v primeru, da pošljemo w_j in se zgodita dve napaki, ali da pošljemo w_i in se zgodijo tri napake. Zaporedje v_3 je bližje kodni besedi w_j. Če je verjetnost napake manj od 0.5, je verjetnost treh napak manjša od verjetnosti dveh napak. Zato se odločimo, da je bila oddana kodna beseda, ki je bližja sprejetemu zaporedju. V našem primeru je $d(v_3, w_j) < d(v_3, w_i)$, zato se odločimo za besedo w_j.

V primeru dveh napak ($e = 2$) bi bila ta odločitev pravilna in napako smo uspeli popraviti, kot smo trdili v začetku:

$$d(w_i, w_j) = 5 = 2 \times e + 1 \Longrightarrow e = 2.$$



Slika 81: Primer koda s sposobnostjo popravljanja napak. Najmanjša razdalja med besedami koda je $d(w_i, w_j) = 5$. Kod je sposoben popravljati še vse dvakratne ($e = 2$) napake ($5 = 2e + 1$).

V primeru, ko bi se v resnici zgodile tri napake, bi bila odločitev napačna, napake ne bi popravili.

Če bi bila razdalja med besedama w_i in w_j enaka 4, bi bilo neko zaporedje v enako oddaljeno od obeh besed, $d(v, w_i) = d(v, w_j) = 2$. V primeru, da bi zaradi napak sprejeli zaporedje v, bi sicer vedeli, da so prisotne napake, napako bi odkrili, ne bi pa se znali odločiti, kaj je bilo oddano; od tu neenačba (22).

4.4.4 Preverjanje parnosti

Preverjanje parnosti je eden najpogostejših postopkov za odkrivanje napak. Pri oddaji dodamo h k informacijskim bitom $b_j, \;\; (j = 1, 2, \ldots, k)$ še parnostni bit. Možno je 'sodo' ali 'liho' preverjanje parnosti. Pri sodem preverjanju dodamo parnostni bit (simbol) tako, da je skupno število enic (skupaj s parnostnim bitom) v besedi sodo. Pri lihem preverjanju dodamo parnostni bit tako, da je skupno število enic liho. Od tu obe imeni. Vrednost parnostnega bita za k informacijskih bitov izračunamo pri sodem preverjanju parnosti po enačbi:

$$b_1 + b_2 + \cdots + b_j + \cdots + b_k + p = 0 \;\; (mod\ 2), \tag{23}$$

pri lihem preverjanju pa po enačbi:

$$1 + b_1 + b_2 + \cdots + b_j + \cdots + b_k + p = 0 \;\; (mod\ 2). \tag{24}$$

Sodo preverjanje parnosti je pogostejše od lihega. Če je zaporedje informacijskih bitov na primer:

$$1\ 1\ 0\ 0\ 1\ 1\ 0$$

je pri sodem preverjanju parnostni bit enak $\mathbf{0}$,

$$1 + 1 + 0 + 0 + 1 + 1 + \mathbf{0} = 0 \quad (mod\ 2),$$

odposlano zaporedje pa

$$1\ 1\ 0\ 0\ 1\ 1\ 0\ \mathbf{0}.$$

Ko sprejemna naprava sprejme zaporedje simbolov, najprej preveri pogoj parnosti:

$$c = b_1 + b_2 + \cdots + b_j + \cdots + b_k + p \pmod 2, \tag{25}$$

V primeru, da je c različen od nič, pomeni to napako med prenosom.

Najmanjša Hammingova razdalja koda je 2. Po enačbi (20) je možno odkrivati vse enkratne napake ($2 = e + 1 \Rightarrow e = 1$), zaradi pravila parnosti pa je možno odkriti vsako lihokratno število napak.

4.4.5 Vzdolžno in prečno preverjanje parnosti

Pri vzdolžnem in prečnem preverjanju parnosti dodamo zaporedjem k informacijskih bitov parnostni bit za prečno preverjanje:

$$\begin{array}{lllllllll}
w_\alpha & = b_{\alpha 1} & +b_{\alpha 2} & +\cdots & +b_{\alpha j} & +\cdots & +b_{\alpha k} & +p_\alpha & = 0 \pmod 2 \\
w_\beta & = b_{\beta 1} & +b_{\beta 2} & +\cdots & +b_{\beta j} & +\cdots & +b_{\beta k} & +p_\beta & = 0 \pmod 2 \\
\cdots & = & & \cdots & & & & & \\
w_\delta & = b_{\delta 1} & +b_{\delta 2} & +\cdots & +b_{\delta j} & +\cdots & +b_{\delta k} & +p_\delta & = 0 \pmod 2 \\
\cdots & = & & \cdots & & & & & \\
w_\omega & = b_{\omega 1} & +b_{\omega 2} & +\cdots & +b_{\omega j} & +\cdots & +b_{\omega k} & +p_\omega & = 0 \pmod 2
\end{array} \tag{26}$$

celotnemu zaporedju besed $w_\alpha, w_\beta, \ldots, w_\omega$, to je sporočilu ali delu sporočila pa parnostno besedo $w_p = (p_1, p_2, \ldots, p_j, \ldots, p_{k+1})$ za vzdolžno preverjanje:

$$\begin{array}{lllllllll}
b_{\alpha 1} & +b_{\beta 1} & +\cdots & +b_{\delta 1} & +\cdots & +b_{\omega 1} & +p_1 & = 0 \pmod 2 \\
b_{\alpha 2} & +b_{\beta 2} & +\cdots & +b_{\delta 2} & +\cdots & +b_{\omega 2} & +p_2 & = 0 \pmod 2 \\
 & & \cdots & & & & & & \\
b_{\alpha j} & +b_{\beta j} & +\cdots & +b_{\delta j} & +\cdots & +b_{\omega j} & +p_j & = 0 \pmod 2 \\
 & & \cdots & & & & & & \\
b_{\alpha k} & +b_{\beta k} & +\cdots & +b_{\delta k} & +\cdots & +b_{\omega k} & +p_k & = 0 \pmod 2 \\
p_\alpha & +p_\beta & +\cdots & +p_\delta & +\cdots & +p_\omega & +p_{k+1} & = 0 \pmod 2
\end{array} \tag{27}$$

Poglejmo primer. Naj bo zaporedje informacjskih bitov:

$$1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0.$$

Dodajmo vsakemu zaporedju treh simbolov en parnostni simbol (po vrsticah) in celotnemu zaporedju še parnostno besedo (parnostne simbole po stolpcih):

$$
\begin{array}{lcccccc|cc}
w_\alpha & = & 1 & + & 1 & + & 1 & + & 1 & = 0 \\
 & & + & & + & & + & & + & \\
w_\beta & = & 0 & + & 1 & + & 0 & + & 1 & = 0 \\
 & & + & & + & & + & & + & \\
w_\gamma & = & 1 & + & 0 & + & 0 & + & 1 & = 0 \\
 & & + & & + & & + & & + & \\
\hline
w_p & = & 0 & + & 0 & + & 1 & + & 1 & = 0 \\
 & & \| & & \| & & \| & & \| & \\
 & & 0 & & 0 & & 0 & & 0 &
\end{array}
$$

Pošljemo zaporedje $w_\alpha, w_\beta, w_\gamma, w_p$. Zaradi napak med prenosom v splošnem sprejmemo različno zaporedje $v_\alpha, v_\beta, v_\gamma, v_p$.

Vzemimo, da se med prenosom zgodi napaka in sprejmemo zaporedje simbolov:

$$1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1.$$

Kje je napaka? Preverimo vzdolžni in prečni pogoj parnosti:

$$
\begin{array}{lcccccc|cc}
v_\alpha & = & 1 & + & 1 & + & 1 & + & 1 & = 0 \\
 & & + & & + & & + & & + & \\
v_\beta & = & 0 & + & 0 & + & 0 & + & 1 & = 1 \\
 & & + & & + & & + & & + & \\
v_\gamma & = & 1 & + & 0 & + & 0 & + & 1 & = 0 \\
 & & + & & + & & + & & + & \\
\hline
v_p & = & 0 & + & 0 & + & 1 & + & 1 & = 0 \\
 & & \| & & \| & & \| & & \| & \\
 & & 0 & & 1 & & 0 & & 0 &
\end{array}
$$

V drugi vrstici in tretjem stolpcu pogoj parnosti ni izpolnjen. Napaka je na presečišču vrstice in stolpca. Ker vemo, kje je napaka, jo lahko popravimo.

Vzdolžno in prečno preverjanje parnosti omogoča odkrivanje in popravljanje vseh enkratnih napak ($e = 1$). Hammingova razdalja med besedami mora biti torej vsaj $2 \times e + 1 = 3$. Kljub temu, da kod omogoča tudi popravljanje napak, pa se v praksi uporablja v glavnem samo za odkrivanje napak.

4.4.6 Hammingov kod

Hammingov kod je poseben primer koda iz velike družine *parnostnih* kodov, ki omogoča ne le odkriti, ampak tudi popraviti vsako enojno napako (napako na enem simbolu). Tudi prej obravnavano preverjanje parnosti ter vzdolžno in prečno preverjanje parnosti sodita v družino parnostnih kodov. V osnovnem preverjanju parnosti smo imeli za skupino informacijskih simbolov en sam parnostni

simbol. Pri preverjanju vzdolžne in prečne parnosti je bilo parnostnih simbolov že več. V splošnem lahko po nekih pravilih dodamo poljubno mnogo parnostnih simbolov. S številom parnostnih simbolov se seveda veča odvečnost koda. Pričakujemo, da se istočasno z večanjem odvečnosti veča tudi zmožnost koda za odkrivanje in/ali popravljanje napak.

Vzemimo kod $W = \{w_i\}$ z dolžino kodnih besed n. Vse kodne besede so torej enako dolge in imajo po n simbolov. Naj bo od teh n simbolov k informacijskih in m parnostnih (odvečnih). Da določimo m parnostnih bitov potrebujemo m pogojev, m enačb oblike (23). Poglejmo primer. Naj bo $n = 6,\ k = 3$ in $m = 3$. Izberimo si simbole b_1, b_2, b_3 za informacijske, simboli b_4, b_5, b_6 pa naj bodo parnostni, kot to določajo naslednji trije pogoji:

$$\begin{array}{ccccc} b_1 & + & b_4 & = 0 & (mod\ 2) \\ b_2 & + & b_5 & = 0 & (mod\ 2) \\ b_3 & + & b_6 & = 0 & (mod\ 2) \end{array}$$

Enačbe lahko zapišemo v matrični obliki:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (mod\ \ 2).$$

ali bolj zgoščeno:

$$\mathbf{A}w^T = \mathbf{0}. \tag{28}$$

Matriko $\mathbf{A}$ imenujemo *parnostna* matrika in popolnoma določa kod W - kodne besede w tega koda so rešitve enačbe (28). Kodnih besed je toliko, kolikor je rešitev enačbe. Ko izberemo matriko $\mathbf{A}$ je torej že določena dolžina kodnih besed, število informacijskih simbolov in s tem število kodnih besed ter število parnostnih simbolov. S parnostno matriko je določena tudi sposobnost koda za popravljanje napak.

Hammingovemu kodu pripada naslednja parnostna matrika velikosti $(m \times n) = (3 \times 7)$:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Dolžina kodnih besed je $n = 7$. Brez dodatnih omejitev bi bilo možnih $2^n = 2^7 = 128$ kodnih besed. Vendar imamo za Hammingov kod še tri pogoje, tri linearno

neodvisne enačbe, iz katerih se da izračunati tri neznanke - tri parnostne simbole. Za zapis informacije ostanejo samo še štirje simboli ($k = 4$). Te lahko poljubno izberemo. Možno je izbrati $2^k = 2^4 = 16$ različnih kombinacij. Denimo, da so simboli b_1, b_2 in b_4 parnostni, ostali štirje simboli b_3, b_5, b_6 in b_7 pa informacijski.

Kodne besede koda določimo tako, da za vsako kombinacijo informacijskih simbolov izračunamo parnostne:

$$\begin{aligned} w_0 &= b_1\ b_2\ 0\ b_4\ 0\ 0\ 0 \\ w_1 &= b_1\ b_2\ 0\ b_4\ 0\ 0\ 1 \\ w_2 &= b_1\ b_2\ 0\ b_4\ 0\ 1\ 0 \\ w_3 &= b_1\ b_2\ 0\ b_4\ 0\ 1\ 1 \\ w_4 &= b_1\ b_2\ 0\ b_4\ 1\ 0\ 0 \\ \cdots &= \cdots \\ w_{15} &= b_1\ b_2\ 1\ b_4\ 1\ 1\ 1 \end{aligned}$$

Na primer, za kodno besedo w_3 imamo:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ b_4 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (mod\ \ 2).$$

Iz česar sledi $b_1 = 1$, $b_2 = 0$ in $b_4 = 0$. Kodna beseda pa je: $w_3 = 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1$.

Poglejmo, kako se da odkrivati napake. Denimo, da eno od kodnih besed pošljemo v kanal in sprejmemo zaporedje v. V splošnem $v \neq w$. Recimo, da je $v = w + \delta$, kjer je δ posledica napak in ima enke na mestih napak. V primeru ene napake ima samo eno enico. Na primer: naj pošljemo $w_3 = 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1$ in zaradi napake na šestem mestu sprejmemo $v = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$. Imamo:

$$v = w_3 + \delta = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 1] + [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0].$$

Na sprejemni strani preverimo pogoj parnosti, izračunamo vektor c,

$$c = \mathbf{A}v^T = \mathbf{A}(w + \delta)^T = \mathbf{A}w^T + \mathbf{A}\delta^T = \mathbf{A}\delta^T.$$

V primeru, da je napaka ena sama, vsebuje δ eno samo enico in sicer na mestu napake. Vektor c je v tem primeru kar enak ustreznemu stolpcu matrike $\mathbf{A}$. Za

naš primer:

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad (mod\ 2).$$

Mesto napake se da v tem primeru določiti iz vektorja c,
Mesto napake $= c_1 \times 2^2 + c_2 \times 2^1 + c_3 \times 2^0$. Za naš primer:
Mesto napake $= 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$.

Popravljanje napak sestavlja naslednje zaporedje opravil:

- sprejmemo v,
- izračunamo c,
- če je vektor c enak nič, napake ni (ali ni odkrita) in v je enak eni od besed. Odločimo se, da je bila oddana beseda $w_i = v$,
- če pa je vektor c različen od nič, iz njega določimo mesto napake in jo popravimo.

Hammingov kod je sposoben popravljati samo napake na enem simbolu. V primeru, da pride do dveh napak, vsebuje δ dve enici in c je kombinacija dveh stolpcev matrike $\mathbf{A}$. Ker je vsaka kombinacija dveh ali več stolpcev spet enaka nekemu drugemu stolpcu, dobimo za c enako vrednost, kot bi jo dobili v primeru ene same napake.

Obstaja več inačic Hammingovega koda, ki se razlikujejo po dolžini (in številu) kodnih besed in po številu parnostnih simbolov, vsi pa popravljajo enojne napake. Na primer, za dolžine kodnih besed $n = 15, 31, 63, \ldots$ imamo $m = 4, 5, 6, \ldots$ parnostnih simbolov, pripadajoča parnostna matrika pa je velikosti (4×15), (5×31), $(6 \times 63), \ldots$. Odvečnost Hammingovega koda pa je

$$R = 1 - \frac{m}{n}.$$

4.5 Ciklično preverjanje

Ciklično kodiranje ali preverjanje (ang. **C**yclic **R**edundancy **C**ode ali **CRC**) se pri prenosu in pri shranjevanju podatkov največ uporablja, preprosto zato, ker je razmeroma enostavno izvedljivo in neprimerno bolj učinkovito od drugih oblik kodiranja. Ciklični kodi spadajo v veliko družino linearnih blokovnih kodov. Ime ciklični izhaja iz dejstva, da je vsak krožni ('ciklični') pomik kodne besede tudi kodna beseda koda. Ta lastnost za naše potrebe razen zaradi poimenovanja postopka ni zanimiva.

Osnovna zamisel cikličnega preverjanja je preverjanje deljivosti zaporedja informacijskih simbolov z zaporedjem *delilnih simbolov*. Delilno zaporedje simbolov mora biti znano tako oddajni kot sprejemni napravi. Na oddajni strani delimo informacijsko zaporedje simbolov z delilnim zaporedjem, ostanek deljenja pripnemo informacijskim simbolom tako, da je skupno zaporedje deljivo z delilnim zaporedjem brez ostanka. ter vse skupaj pošljemo v kanal. Sprejemna naprava deli sprejeto zaporedje z enakim delilnim zaporedjem in če se deljenje **ne** izide, pomeni to napako. Postopek cikličnega preverjanje je tako popolnoma določen z izbiro zaporedja delilnih simbolov.

Ciklični kod imenujemo tudi polinomski kod (ang. Polynomial Code). To drugo ime izhaja iz dejstva, da se da zaporedje binarnih simbolov obravnavati kot koeficente polinoma potrebne stopnje. Zaporedje binarnih simbolov dolžine $k+1$

$$b_k\ b_{k-1}\ b_{k-2}\ \ldots\ b_i\ \ldots\ b_1\ b_0$$

se da predstaviti s polinomom $P_k(x)$ stopnje k:

$$P_k(x) = b_k x^k + b_{k-1}x^{k-1} + \cdots + b_i x^i + \cdots + b_1 x^1 + b_0.$$

Binarni simboli

$$b_i = \begin{cases} 0 \\ 1 \end{cases} \quad (i = k, k-1, \ldots, 1, 0)$$

določajo koeficiente polinoma. Na primer, zaporedju šestih simbolov 1 1 0 0 1 1 ustreza naslednji polinom pete stopnje:

$$P_5(x) = 1x^5 + 1x^4 + 0x^3 + 0x^2 + 1x^1 + 1 = x^5 + x^4 + x^1 + 1.$$

To dejstvo nam bo v pomoč pri razlagi osnovne zamisli ugotavljanja pravilnosti prenosa s cikličnim preverjanjem. Postopek cikličnega preverjanja bomo torej razložili s predstavitvijo binarnih zaporedij simbolov s polinomi.

Informacijsko zaporedje simbolov dolžine $(k+1)$ predstavimo s polinomom $P_k(x)$ stopnje k. Delilno zaporedje simbolov dolžine $r+1$ predstavimo s polinom $G_r(x)$ stopnje r. Temu polinomu pravimo *generatorjev* polinom. Določanje in preverjanje kontrolnih simbolov obsega naslednje zaporedje korakov:

- zaporedju $(k+1)$ informacijskih simbolov pripišemo r simbolov z vrednostjo nič. Dobljenemu zaporedju pripada polinom $P_k(x) \times x^r$.

- Polinom $P_k(x) \times x^r$ delimo (in sicer po modulu 2) z generatorjevim polinomom. Deljenje se v splošnem ne izide. Dobimo rezultat deljenja $Q(x)$ in ostanek $R(x)$, ki je največ stopnje $(r-1)$:

$$\frac{P_k(x) \times x^r}{G_r(x)} = Q(x) + \frac{R(x)}{G_r(x)}. \tag{29}$$

- Odštejemo (po modulu 2) ostanek deljenja $R(x)$, ki ima vedno največ r simbolov od $P_k(x) \times x^r$. Ostanek deljenja lahko tudi prištejemo, ker je odštevanje po modulu 2 enako seštevanju. Dobimo zaporedje

$$T(x) = P_k(x) \times x^r - R(x) = P_k(x) \times x^r + R(x),$$

  ki je deljivo z $G_r(x)$ brez ostanka. V to se prepričamo, če enačbo (29) množimo z $G_r(x)$. Dobimo

$$P_k(x) \times x^r = Q(x) \times G_r(x) + R(x).$$

  Ko nesemo ostanek $R(x)$ na levo stran, imamo

$$P_{k-1}(x) \times x^r + R(x) = Q(x) \times G_r(x).$$

  Desna stran enačbe vsebuje člen $G_r(x)$ in je gotovo deljiva z njim (brez ostanka). Ker je desna stran enaka levi, mora biti tudi ta deljiva z $G_r(x)$.

- Zaporedje $T(x)$ pošljemo v kanal. Sprejemnik sprejme $T(x)'$, ki zaradi napak med prenosom ni nujno enako $T(x)$,

$$T(x)' = T(x) + E(x).$$

  V polinomu $E(x)$ so zajete napake. Ko napak ni, je $E(x)$ identično nič. Vsaka napaka na simbolu prispeva en člen k polinomu $E(x)$. Na primer, v primeru napake na i-tem simbolu (šteto z desne proti levi, začenši z nič), je $E(x) = x^i$.

- Sprejemnik preveri pogoj deljivosti - deli sprejeto zaporedje $T'(x)$ z $G_r(x)$,

$$\frac{T(x)'}{G_r(x)} = \frac{T(x) + E(x)}{G_r(x)}.$$

  Ker je $T(x)$ deljiv z $G_r(x)$, se deljenje izide, če napake ni $(E(x) = 0)$ ali, če je polinom napak $E(x)$ deljiv z $G_r(x)$ brez ostanka. Takih napak se ne da odkriti.

Primer:

Informacijskemu zaporedju desetih simbolov

$$1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1$$

bomo določili kontrolne simbole. Naj bo delilno zaporedje

$$1\ 0\ 0\ 1\ 1.$$

Torej ustreza informacijskemu zaporedju polinom

$$P_9(x) = x^9 + x^8 + x^6 + x^4 + x^3 + x^1 + 1$$

in generatorjev polinom je

$$G_4 = x^4 + x^1 + 1.$$

Informacijskemu zaporedju dodamo štiri ničle ($r = 4$) in ga delimo z delilnim zaporedjem. Čeprav iščemo samo ostanek deljenja, bomo računali tudi rezultat deljenja. Računamo po modulu dva:

$$\begin{array}{llllllllllllll}
1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0 & : & 1\ 0\ 0\ 1\ 1 & = & 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \\
\underline{1\ 0\ 0\ 1\ 1} & & & & & & & & & \\
0\ 1\ 0\ 0\ 1\ 1 & & & & & & & & & \\
\quad \underline{1\ 0\ 0\ 1\ 1} & & & & & & & & & \\
\quad 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0 & & & & & & & & & \\
\qquad\qquad \underline{1\ 0\ 0\ 1\ 1} & & & & & & & & & \\
\qquad\qquad 0\ 0\ 1\ 0\ 1\ 0\ 0 & & & & & & & & & \\
\qquad\qquad\quad \underline{1\ 0\ 0\ 1\ 1} & & & & & & & & & \\
\qquad\qquad\quad 0\ 0\ 1\ 1\ 1\ 0 & & \text{Ostanek } (1\ 1\ 1\ 0) & & & & & & &
\end{array}$$

V kanal pošljemo zaporedje:

$$1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0.$$

Sposobnost odkrivanja napak s cilkličnim preverjanjem je odvisna od generatorjevega polinoma. Poglejmo, kakšne lastnosti mora imeti 'dober' generatorjev polinom.

Zmožnost odkrivanja enojnih napak V primeru enojne napake je $E(x) = x^i$, pri čemer i določa, na katerem simbolu je napaka. Če $G(x)$ vsebuje vsaj dva člena, potem $E(x)$ ne bo deljiv z $G(x)$ in vsaka enojna napaka bo odkrita.

Zmožnost odkrivanja dvojnih napak V primeru dveh napak vsebuje polinom napak $E(x)$ dva člena, $E(x) = x^i + x^j \quad (i > j)$. Ekvivalentno, $E(x) = x^j(x^{i-j}+1) = x^j(x^k+1)$ in k pomeni razdaljo med napakama. Če predpostavimo, da $G(x)$ ni deljiv z x, je zadosten pogoj za odkrivanje dvojnih napak, da $G(x)$ ne deli $x^k + 1$ za vsak k do dolžine sporočila. Polinomi razmeroma nizke stopnje s to lastnostjo so znani. Na primer, polinom $x^{15} + x^{14} + 1$ ne deli $x^k + 1$ vse do $k = 32768$.

Zmožnost odkrivanja lihokratnih napak V primeru lihega števila napak vsebuje polinom $E(x)$ liho število členov. Denimo, da je $E(x) = x^7 + x^2 + 1$ v primeru treh napak. Zanimivo je, da noben polinom z lihim številom členov ne vsebuje faktorja $(x+1)$ in se ga zato nikakor ne da zapisati kot $E(x) = (x+1)F(x)$. Torej ni deljiv z $(x+1)$. V to se prepričamo, če za x vstavimo vrednost 1. Vedno kadar polinom vsebuje liho število členov, je njegova vrednost 1. Na primer:

$$E(1) = 1^7 + 1^2 + 1 = 1 + 1 + 1 = 1 \quad (mod\ 2).$$

V primeru, da bi se $E(x)$ dalo zapisati kot $(x + 1)F(x)$, bi imeli:

$$E(1) = (1 + 1) \times F(1) = 0 \times F(1) = 0 \quad (mod\ 2),$$

kar je protislovno. Torej polinom z lihim številom členov ne vsebuje faktorja $(x + 1)$ in tudi ni deljiv z $(x + 1)$. Če hočemo odkrivati vse lihkokratne napake, moramo izbrati tak generatorjev polinom $G(x)$, ki ne deli nobenega polinoma z lihim številom členov. To dosežemo, če v generatorjev polinom vgradimo člen $(x + 1)$. Na primer, polinom (mednarodno priporočilo CCITT V.41)

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

vsebuje faktor $(x + 1)$.

Zmožnost odkrivanja zaporednih napak Pri prenosu podatkov se napake običajno ne pojavljajo kot napake na osamljenih simbolih, temveč si sledijo na zaporedju sosednjih simbolov. Najpogostejši vzrok napak je impulzni šum. Tipično trajanje impulznega šuma je okrog 10 milisekund in je večinoma posledica preklopov stikal ali drugih elektromehanskih naprav, razelektrenj v ozračju, delovanja motorjev i.t.d. V slišnem področju spoznamo impulzni šum po motečem prasketanju, pri prenosu podatkov s hitrostjo 2400 [b/s] pa tak šum uniči $0.01 \times 2400 \approx 24$ zaporednih bitov. Take 'izbruhe' zaporednih napak je dosti težje odkriti, kot pojav osamljenih napak na posameznih simbolih.

Ciklično preverjanje sporočil je, v nasprotju z drugimi postopki, učinkovito tudi pri odkrivanju zaporednih napak. Hitro se prepričamo, da ciklično preverjanje sporočil s polinom stopnje r omogoča odkrivanje vsakega zaporedja napak

na r ali manj simbolih. Izbruh napak na zaporedju k simbolov se da predstaviti kot polinom

$$x^i(x^{k-1} + \cdots + 1),$$

kjer člen x^i določa samo mesto napak vzdolž sporočila. Če generatorjev polinom vsebuje konstantni člen $x^0 = 1$, potem ne bo v nobenem primeru imel x^i za faktor in ostanek deljenja ne bo nič, dokler je stopnja polinoma v oklepajih nižja od stopnje $G(x)$. Zato:

$$k - 1 < r \Rightarrow k < r + 1 \quad ali \quad k \leq r.$$

V primeru izbruha napak na zaporedju $r + 1$ simbolov, bo ostanek deljenja sporočila z $G(x)$ enak nič samo v primeru, da je $E(x) = x^i G(x)$. Po definiciji napake na zaporedju simbolov dolžine $r+1$ morata biti napačna prvi in zadnji bit v zaporedju bitov dolžine $r + 1$, sicer bomo imeli opravka s krajšim zaporedjem. Da se zaporedje napak ujame z zaporedjem bitov za preverjanje $(G(x))$ je odvisno od $r-1$ vmesnih simbolov. Ob predpostavki, da so vse možne kombinacije napak enako verjetne, je verjetnost neodkrite napake $1/2^{r-1}$.

Naslednji trije generatorjevi polinomi so postali mednarodni standard:

$$\begin{aligned} CRC-12 &= x^{12} + x^{11} + x^3 + x^2 + x^1 + 1 \\ CRC-16 &= x^{16} + x^{15} + x^2 + 1 \\ CRC-CCITT &= x^{16} + x^{12} + x^5 + 1. \end{aligned} \tag{30}$$

Vsi trije vsebujejo faktor $(x + 1)$. Prvi polinom se uporablja pri prenosu šestbitnih podatkov, ostala dva pri prenosu 8-bitnih podatkov. Polinom CRC-16 se uporablja v protokolu IBM BSC, polinom CRC-CCITT je IBM najprej uporabljal v protokolu SDLC, zatem pa ge je CCITT standardiziral. Pri zadnjih dveh polonimoh šestnajste stopnje pripnemo k zaporedju informacijskih bitov zaporedje 16 kontrolnih simbolov. Da se pokazati, da odkrivata vsako enojno ali dvojno napako, vsako lihokratno napako, vsak izbruh napak na 16 zaporednih simbolih ali manj in 99.997 % izbruhov napak na 17 simbolih.

Ciklično preverjanje je v pogledu odkrivanja napak neprimerno učinkovitejše kot preverjanje parnosti, sicer pa se oba precej uporabljata. Preverjanje parnosti je v pogledu realizacije manj zahtevno od cikličnega preverjanja. Da se ga razmeroma enostavno realizirati s programom ali vezjem. Izračun kontrolnih simbolov pri cikličnem preverjanju je bolj zahteven, zato se ga skoraj vedno realizira z materialno opremo. Dobro znana so učinkovita vezja za računanje in preverjanje kontrolnih simbolov s preprostimi pomikalnimi registri. V časovno manj zahtevnih primerih realiziramo ciklično preverjanje s programom za deljenje. Znani pa so tudi učinkoviti algoritmi za določanje kontrolnih bitov s pomočjo pregledovalnih tabel [7].

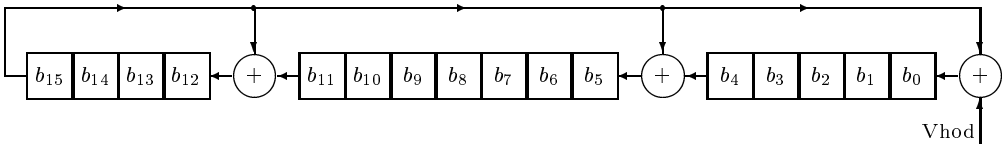
4.5.1 Vezja za ciklično preverjanje

Oglejmo si primer realizacije cikličnega preverjanja s pomikalnim registrom in logičnimi vrati za generatorjev polinom:

$$CRC - CCITT = x^{16} + x^{12} + x^5 + 1.$$

Poenostavljeno vezje je shematično narisano na sliki 82. Naj bodo vrednosti celic pomikalnega registra v začetku postavljene na nič. Seveda bi se lahko odločili za poljubno začetno vrednost pomikalnega registra, važno je le, da je le-ta enaka na oddajni in sprejemni strani. Vezje deluje kot delilnik po pravilih deljenja uporabljenih v prejšnjem primeru 'ročnega' deljenja. Zaporedje bitov (sporočilo) vstopa na sponki 'Vhod' ter se pomika proti levi. Ko prvi simbol z vrednostjo ena prispe v zadnjo celico (b_{15}), se njegova vrednost prenese nazaj in vpliva na vrednosti celic b_{12}, b_5 in b_0 (primerjaj z generatorjevim polinomom). Z drugimi besedami, dokler ne pride enica do konca registra, opravljamo samo pomik in opravimo 'odštevanje', ko pride na konec registra enica. Ko zaporedja informacijskih bitov konča, je potrebnih še šestnajst pomikov z vrednostjo nič na vhodu, da pride ostanek deljenja iz pomikalnega registra.

Za odkrivanje napak je potrebno enako vezje in enak postopek. Na vhodni sponki vstopajo najprej informacijski simboli, za njimi pa še kontrolni simboli. Če je po zadnjem pomiku vsebina vseh celic pomikalnega registra enaka nič, je sporočilo prenešeno brez napake.



Slika 82: Poenostavljena shema vezja za ciklično preverjanje s polinom CRC-CCITT.

Vezja, ki se praktično uporabljajo, so nekoliko drugačna od vezja, na katerem smo razložili delovanje. Ta vezja delujejo sicer po enakem principu, le da pomikanje konča takoj, ko je konec sporočila. Dodatno pomikanje, da dobimo ostanek iz registra, torej ni potrebno.

4.5.2 Ciklično preverjanje s programskim deljenjem

Če želimo realizirati ciklično preverjanje programsko, potem je najlažje napisati podprogram, ki deli dano informacijsko zaporedje z izbranim generatorjem, nekako tako, kot smo pokazali na primeru. Funckijo v jeziku C, ki na oddajni strani generira in na sprejemni strani preverja kontrolne simbole, prikazuje spodnji zapis. Predpostavljamo, da je generator stopnje šestnajst.

```
#include <stdio.h>
#include <stdio.h>
/*-------------------------------------------------------------------------
  Ime:       CrcDeljenje
             Funkcija za deljenje okvirja z generatorjem
  Okvir:     kazalec na polje 8-bitnih podatkov okvirja, vecje za 2 od
             stevila podatkov.
  Generator: bitni vzorec, ki definira generator - brez clena x^16
             npr: CRC-CCITT: 0001 0000 0010 0001 (0x1021).
  Dolzina:   dolzina podatkovnega dela okvirja - brez ostanka (kontrole).
  Vrne:      ostanek deljenja - 16 bitni (2 bajta) CRC ostanek.
             Ostanek doda tudi k okvirju - za podatki.
*/
typedef unsigned short int  ushort;
typedef unsigned char       uchar;

ushort CrcDeljenje( uchar *Okvir, ushort Generator, ushort Dolzina )
{
  ushort TestBit, Ostanek;
  uchar  Podatek;
  int    i, n;

  Ostanek = 0; Okvir[ Dolzina ] = Okvir[ Dolzina + 1 ] = 0;
  for( n = 0; n < Dolzina+2; n++ ){
    Podatek = Okvir[ n ];         /* na vrsti je n-ti bajt okvirja    */
    for( i = 0; i < 8; i++ ){
      TestBit = Ostanek & 0x8000; /* testiramo zgornji bit deljenca   */
      Ostanek <<= 1;              /* predno ga pomaknemo levo         */
      if((Podatek & 0x80)==0x80){ /* dodamo naslednji podatkovni bit  */
        Ostanek |= 1;             /* v primeru, da  je bit = 1        */
      }
      Podatek <<= 1;              /* V vsakem primeru pomik podatka   */
      if( TestBit ){
        Ostanek ^= Generator;     /* ''Odstejemo deljitelj''          */
      }
    }
  }
  Okvir[ Dolzina     ] = Ostanek >> 8;
  Okvir[ Dolzina + 1 ] = Ostanek & 0xff;
  return Ostanek;
} /* Konec CrcDeljenje */
```

4.5.3 Ciklično preverjanje s tabelo ostankov

Bolj učinkoviti algoritmi za ciklično preverjanja temeljijo na pregledovanju (indeksiranju) tabele ostankov (delnega) deljenja. Ostanki deljenja z izbranim generatorjem se najprej in enkrat za vselej izračunajo ter tabelirajo. Tabela in algoritem sta enaka na oddajni in sprejemni strani. V algoritem vstopa sporočilo bajt za bajtom. Za vsak naslednji bajt se v odvisnosti od vrednosti bajta, trenutnega ostanka deljenja in tabele računa nov ostanek deljenja, do konca okvirja.

Zamisel algoritma za ciklično preverjanje s pregledovanjem tabele bomo razložili na podoben način, kot smo razložili osnovno zamisel cikličnega preverjanja.

Naj bo $G_{16}(x)$ generatorjev polinom šestnajste stopnje in $P(x)$ naj tokrat pomeni začetnih nekaj bajtov okvirja. Predpostavimo, da je trenutni ostanek deljenja enak $S(x)$. Torej je

$$P(x)x^{16} = Q(x)G_{16}(x) + S(x).$$

Ostanek $S(x)$ je največ stopnje 15. Dodajmo k $P(x)$ še naslednjih osem bitov – naslednji bajt – okvirja. Podaljšanemu zaporedju potem ustreza polinom $P'(x)$,

$$P'(x) = P(x)x^8 + B(x),$$

pri čemer simbole dodanega bajta upoštevamo s polinomom $B(x)$,

$$B(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0.$$

Naj $S'(x)$ označuje ostanek deljenja podaljšanega polinoma $P'(x)$. Potem je

$$P'(x)x^{16} = Q'(x)G_{16}(x) + S'(x).$$

Z upoštevanjem zveze med $P'(x)$ in $P(x)$ dobimo za levo stran enačbe:

$$\begin{aligned}
[P(x)x^8 + B(x)]x^{16} &= P(x)x^{16}x^8 + B(x)x^{16} = \\
&= [Q(x)G_{16}(x) + S(x)]x^8 + B(x)x^{16} = \\
&= Q(x)G_{16}(x)x^8 + S(x)x^8 + B(x)x^{16} = \\
&= Q(x)G_{16}(x)X^8 + [S(x) + B(x)x^8]x^8.
\end{aligned}$$

Prvi člen je očitno deljiv z $G_{16}(x)$ brez ostanka. Ker nas zanima samo ostanek deljenja, se osredotočimo na zadnji člen. Zapišimo ostanek $S(x)$ kot vsoto dveh polinov sedme stopnje, ki opisujeta zgornji in spodnji bajt ostanka, $S(x) = S_H(x)x^8 + S_L(x)$ in upoštevajmo nove oznake v prejšnjem izrazu. Dobimo:

$$[S(x)] + B(x)x^8]x^8 = S_L(x)x^8 + [S_H(x) + B(x)]x^{16}.$$

Prvi člen je nižje stopnje od generatorja in je zato kar enak ostanku deljenja z generatorjem. Nov ostanek deljenja $S'(x)$ je:

$$S'(x) = S_L(x)x^8 + Ostanek\{[S_H(x) + B(x)]x^{16}]\}$$

Iz zadnjega izraza sledi, da se da izračunati ostanek sporočila podaljšanega za en dodani bajt iz spodnjega bajta prejšnjega ostanka in ostanka deljenja seštevka zgornjega bajta prejšnjega ostanka in dodanega bajta, $V(x)x^{16} = [S_H(x) + B(x)]x^{16}$. Obstaja samo 256 možnih kombinacij člena $V(x)$, za katere se da ostanek vnaprej izračunati in tabelirati. Rezultat je tabela 256 šestnajstbitnih ostankov deljenja, ki je odvisna samo od izbire generatorja.

Algoritem za ciklično preverjanje, ki temelji na pregledovanju tabele ostankov T, je na primer tak:

1. postavi začetni ostanek na nič, $S = 0$,
2. prištej po modulu 2 naslednji bajt sporočila k zgornjemu bajtu ostanka, $V = S_H + B$. Vrednost za V je indeks v tabelo ostankov T, vrednost ostanka je $T[V]$.
3. pomakni S za osem mest levo. S tem se zgornji del S_H izgubi. Rezultat je S_L pomaknjen za osem mest levo.
4. prištej po modulu 2 vrednost ostanka $T[V]$ k S, $S = S + T[V]$.
5. ponavljaj korake 2-4 do konca okvirja.

Naslednja funkcija izračuna tabelo vseh možnih ostankov deljenja z generatorjem CRC-CCITT. Pri deljenju uporablja funkcijo `CrcDeljenje`.

```
/*---------------------------------------------------------------------------
  Ime:       GenCrcTabelo
             Funkcija za generiranje tabele za dan generator. Tabela
             se izpi\ss e tudi na zaslon.
  CrcTabela: kazalec na tabelo ostankov - velikosti 256, ki jih
             izra\cc una (vrne) funkcija.
  Generator: bitni vzorec, ki definira generator - brez clena x^16
             npr: CRC-CCITT: 0001 0000 0010 0001 (0x1021).
*/

void GenCrcTabelo( ushort *CrcTabela, ushort Generator )
{
  int    Index;
  uchar  Temp[5];
  ushort CrcDeljenje( uchar *, ushort, ushort );

  for( Index = 0; Index < 256; Index++ ){
    Temp[ 0 ] = Index;  Temp[ 1 ] = Temp[ 2 ] = 0;
    CrcTabela[ Index ] = CrcDeljenje( Temp, Generator, 1 );
    if( (Index % 16) == 0) printf("\n");
    printf("%4x ", CrcTabela[ Index ] );
  }
  printf("\n");
} /* Konec GenCrcTabelo */
```

Tabela, ki jo izpiše zgornja funkcija, izgleda takole:

```
0000 1021 2042 3063 4084 50a5 60c6 70e7 8108 9129 a14a b16b c18c d1ad e1ce f1ef
1231 0210 3273 2252 52b5 4294 72f7 62d6 9339 8318 b37b a35a d3bd c39c f3ff e3de
2462 3443 0420 1401 64e6 74c7 44a4 5485 a56a b54b 8528 9509 e5ee f5cf c5ac d58d
3653 2672 1611 0630 76d7 66f6 5695 46b4 b75b a77a 9719 8738 f7df e7fe d79d c7bc
48c4 58e5 6886 78a7 0840 1861 2802 3823 c9cc d9ed e98e f9af 8948 9969 a90a b92b
5af5 4ad4 7ab7 6a96 1a71 0a50 3a33 2a12 dbfd cbdc fbbf eb9e 9b79 8b58 bb3b ab1a
6ca6 7c87 4ce4 5cc5 2c22 3c03 0c60 1c41 edae fd8f cdec ddcd ad2a bd0b 8d68 9d49
7e97 6eb6 5ed5 4ef4 3e13 2e32 1e51 0e70 ff9f efbe dfdd cffc bf1b af3a 9f59 8f78
9188 81a9 b1ca a1eb d10c c12d f14e e16f 1080 00a1 30c2 20e3 5004 4025 7046 6067
83b9 9398 a3fb b3da c33d d31c e37f f35e 02b1 1290 22f3 32d2 4235 5214 6277 7256
b5ea a5cb 95a8 8589 f56e e54f d52c c50d 34e2 24c3 14a0 0481 7466 6447 5424 4405
a7db b7fa 8799 97b8 e75f f77e c71d d73c 26d3 36f2 0691 16b0 6657 7676 4615 5634
d94c c96d f90e e92f 99c8 89e9 b98a a9ab 5844 4865 7806 6827 18c0 08e1 3882 28a3
cb7d db5c eb3f fb1e 8bf9 9bd8 abbb bb9a 4a75 5a54 6a37 7a16 0af1 1ad0 2ab3 3a92
fd2e ed0f dd6c cd4d bdaa ad8b 9de8 8dc9 7c26 6c07 5c64 4c45 3ca2 2c83 1ce0 0cc1
ef1f ff3e cf5d df7c af9b bfba 8fd9 9ff8 6e17 7e36 4e55 5e74 2e93 3eb2 0ed1 1ef0
```

Računanje CRC stanka s pregledovanjem prej izračunane tabele prikazuje spodnji program.

```
/*---------------------------------------------------------------------------
  Ime:       CrcPoBajtih
             Funkcija za CRC preverjanje s tabelo

  Okvir:     kazalec na polje 8-bitnih podatkov okvirja, vecje za 2 od
             stevila podatkov (torej Dolzina+2).

  CrcTabela: Kazalec na tabelo 256 ostankov za CRC-CCITT

  Dolzina:   dolzina podatkovnega dela okvirja.
  Vrne:      ostanek deljenja - 16 bitni (2 bajta) CRC ostanek
             Ostanek doda tudi k okvirju - za podatki.
*/

ushort CrcPoBajtih( uchar *Okvir, ushort *CrcTabela, ushort Dolzina)
{
  ushort Ostanek, Podatek;
  int    i;

  for( Ostanek = 0, i = 0; i < Dolzina; i++ ){
    Podatek = (ushort)Okvir[ i ];
    Ostanek = (Ostanek << 8)^CrcTabela[ (Ostanek>>8)^Podatek];
  }
  Okvir[ Dolzina     ] = Ostanek >> 8;
  Okvir[ Dolzina + 1 ] = Ostanek & 0xff;
  return Ostanek;
}
```

Primer preizkusnega programa, ki za dano zaporedje podatkov izračuna CRC ostanek na oba načina (s preprostim deljenjem in s pregledovanjem tabele), je zapisan spodaj.

```
/* Preizkusni program */

int
main( int argc, char **argv )
{
   ushort Polinom = (ushort)0x1021;
   ushort CrcTabela[256];

   ushort Ostanek;
   int    i;

   uchar Okvir[8] = { 1, 2, 3, 4, 5, 6, 0, 0 };

   /* CRC z deljenjem */
   Ostanek = CrcDeljenje( Okvir, Polinom, 6 );
   for( i = 0; i < 8; i++){
     printf("Okvir[ %2d ] = %x\n", i, Okvir[ i ] );
   }
   printf( "Ostanek = %x\n", Ostanek );

   /* CRC s tabelo  */
   GenCrcTabelo( CrcTabela, Polinom );
   Ostanek = CrcPoBajtih( Okvir, CrcTabela, 6 );
   for( i = 0; i < 8; i++){
     printf("Okvir[ %2d ] = %x\n", i, Okvir[ i ] );
   }
   printf( "Ostanek = %x\n", Ostanek );

   return 0;
}
```

4.5.4 Ciklično preverjanje z reducirano tabelo ostankov

S pomočjo tabele ostankov prihranimo na času, ki je potreben za izračun CRC ostanka, na račun pomnilnika, ki je potreben za shranjevanje tabele. Algoritem, ki zmanjša pomnilniški prostor za shranjevanje tabele na račun nekaj več računanja, upošteva naslednjo lastnost ostanka:

$$\begin{aligned} O\{V(x)x^{16}\} &= O(b_7x^{23} + b_6x^{22} + \cdots + b_0x^{16}) \\ &= O(b_7x^{23}) + O(b_6x^{22}) + \cdots + O(b_0x^{16}) \end{aligned} \tag{31}$$

in členi $b_i, (i = 0, 1, \ldots, 7)$ so lahko 0 ali 1. Za vsak člen b_i izračunamo ostanek deljenja z izbranim generatorjem. Dobimo osem 16-bitnih "delnih" ostankov, ki jih tabeliramo. Končni ostanek izračunamo tako, ta seštejemo tiste delne ostanke, za katere je $b_i = 1$. V povprečju so torej potrebna štiri dodatna seštevanja, potrebujemo pa osemkrat manj pomnilnika za shranjevanje tabele.

4.6 Primeri protokolov podatkovnega sloja

4.6.1 BSC

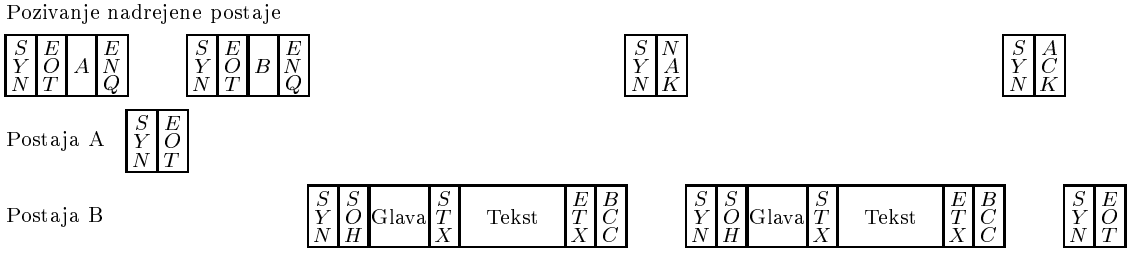
Protokol BSC (Binary Sinchronous Communications) ali z drugim imenom Bysinc je znakovno usmerjen protokol za sinhroni prenos. Predvideva IBM EBCDIC (Extended Binary Coded Decimal Interchange Code) kodiranje znakov, možna pa je tudi uporaba ASCII kodiranja. Protokol so razvili pri IBM-u in je v uporabi že od leta 1968. "BSC kompatibilnost" ali "emulacija" BSC protokola s strani drugih proizvajalcev komunikacijske opreme je bila včasih praktično obvezna. Konec sedemdesetih let so ga začeli izrivati bitno usmerjeni protokoli, recimo SDLC. BSC protokol tu omenjamo predvsem zato, ker se da njegove sledi marsikje opaziti tudi še danes.

Za nadzor komunikacije so izbrani kontrolni znaki: SYN (Synchronization), SOH (Start Of Header), STX (STart of teXt), ETX (End of TeXt), ETB (End of Block), EOT (End Of Transmission), NAK (Negative Acknowledge), DLE (Data Link Escape), ENQ (Inquiry), in še nakaj drugih. BSC okvir je deljen na več polj, od katerih sta glava in podatkovni del spremenljive dolžine. Oblika okvirja je skicirana na sliki 83.

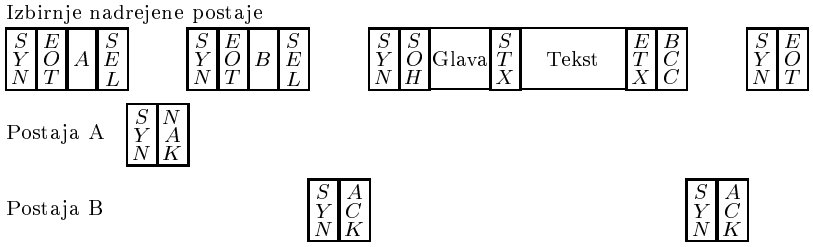
Slika 83: Oblika okvirja protokola BSC.

Okvir začne s sinhronizacijskima znakoma SYN SYN, z neobvezno "glavo" (ang. Header), ki začne s SOH in konča s STX. STX hkrati napove začetek podatkovnega dela okvirja. Podatkovni del je poljubne dolžine in konča z ETB za vmesni ali z ETX za zadnji okvir sporočila. Možno je ali vzdolžno in prečno preverjanje parnosti ali ciklično preverjanje s šestnajstimi biti (polje BCC). Transparenten prenos podatkov se zagotavlja z napovedovanjem kontrolnih znakov. Za napovedni znak je izbran DLE. Pravilen prenos okvirjev se izmenično potrjuje z ACK0 in ACK1. Ponoven prenos se zahteva z NAK. Znak EOT zaključi komunikacijo in postavi protokol v nadzorni način. BSC protokol se lahko nahaja v enem od dveh načinov, v kontrolnem ali v tekstovnem. V tekstovnem načinu se prenašajo podatki, v kontrolnem pa se opravljajo nadzorne funkcije kanala, kot je recimo vzpostavljanje zveze. V večtočkovnem ali zvezdastem omrežju je možno vzpostavljanje zveze s pozivanjem (ang. Polling) ali izbiranjem (ang. Selecting). Slika 84 ponazarja poenostavljen način pozivanja. Pri pozivanju nadrejena postaja krožno poziva oziroma naslavlja podrejene postaje. Poziv pomeni povabilo na oddajo. Če podrejena postaja nima pripravljenih podatkov, odgovori odklonilno z NAK. Sicer začne z oddajo podatkovnega okvirja. Ko odda zadnji

okvir pošlje EOT in s tem je zveza prekinjena. V primeru, da želi nadrejena postaja poslati podatke podrejeni postaji, jo "izbere". Prenos podatkov začne samo, če se izbrana postaja odzove pritrdilno in traja do oddaje EOT. Poenostavljen primer pozivanja je skiciran na sliki 85.



Slika 84: Vzpostavljanje zveze s pozivanjem. Nadrejena postaja najprej pozove postajo A, ki se odzove odklonilno. Za njo pozove postajo B, ki v odgovor pošlje podatke. Ker je prvi prenos okvirja neuspešen, ga nadrejena postaja zavrne z NAK. Ponovljeni prenos je uspešen, kar nadrejena postaja potrdi z ACK. Prenos konča podrejena postaja z EOT.



Slika 85: Vzpostavljanje zveze z izbiranjem. Nadrejena postaja najprej izbere postajo A (ukaz SEL), ki se odzove odklonilno z NAK. Za njo izbere postajo B, ki se odzove pritrdilno z ACK. Sledi oddaja okvirja, ki ga izrana postaja potrdi z ACK. Nadrejena postaja zaključi komunikacijo z oddajo EOT.

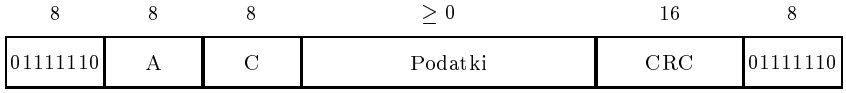
4.6.2 SDLC, HDLC in LAPB

Protokol SDLC (Synchronous Data Link Control) so razvili pri IBM-u v začetku sedemdesetih let (1974) za svojo arhitekturo omrežja SNA (System Network Architecture). IBM je specifikacijo protokola poslal v standardizacijo organizaciji ANSI, da bi ga potrdila kot ameriški standard in organizaciji ISO, da bi ga potrdila za mednarodni standard. Nobena ga ni potrdila brez sprememb. Tako sta brez večjih in bistvenih razlik nastala dva uradna protokola: ANSI ADCCP (Advanced Data Communication Control Procedure) in ISO HDLC (High-Level Data Link Control), ki ga je CCITT potem za svoje potrebe v omrežjih X.25

priredil in izdal pod imenom LAP (Link Access Procedure) ter nato v dopolnjeni obliki pod imenom LAPB (LAP-Balanced) (slika 86).

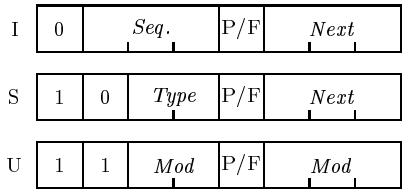
Slika 86: Razvoj protokolov podatkovnega sloja.

Vsi omenjeni protokoli so bitno usmerjeni in uporabljajo enako obliko okvirja, razlikujejo se samo v podrobnostih nadzornega polja, glej skico 87. Najkrajši ovir ne nosi podatkov in obsega 32 bitov (naslov + nadzor + preverjanje). Maksimalna dolžina podatkovnega dela ni predpisana.



Slika 87: Oblika okvirja protokola SDLC in njegovih naslednikov.

Za preverjanje okvirja se koristi polinom CRC-CCITT. Naslovni del (A) pri povezavi dveh vozlišč točka-točka nima pomena, pomemben je predvsem pri pozivanju podrejenih (ang. Slave ali Tributary) vozlišč v večtočkovnih ali zvezdastih omrežjih. Nadzorni del (C) razlikuje podatkovne od nadzornih okvirjev, služi za številčenje podatkovnih okvirjev in potrdil ter drugim nadzornim funkcijam. Oblika nadzornega dela je prikazana na sliki 88. Bit P (Poll/Final) uporablja nadrejeno vozlišče (ang. Master) za pošiljanje ukaza podrejenemu vozlišču, podrejeno vozlišče pa postavi bit F za javljanje odziva na ukaz.



Slika 88: Oblika nadzornega dela okvirja SDLC.

Možno je selektivno ponavljanje (SRP) okvirjev ali vračnaje nazaj na N (GBN) s pozitivnim in z negativnim potrjevanjem ter z drsečim oknom. Številčenje okvirjev in potrdil je po modulu osem s tribitno številko. Protokola HDLC in LAPB omogočata prehod na 7-bitno številčenje.

Obstajajo tri vrste okvirjev: I (information), S (Supervisory) in U (Unnumbered). Podatkovni okvir (I) nosi podatke in *Seq.* je tedaj zaporedna številka

okvirja. S podatkovnimi okvirji se lahko prenaša potrdila podatkovnih okvirjev, ki gredo v nasprotni smeri. V tem primeru je *Next* zaporedna številka potrdila oziroma številka naslednjega okvirja, ki ga pričakuje sprejemnik. Na primer, če se pokvari okvir s številko tri, bo sprejemnik zahteval ponoven prenos okvirja s številko tri (Next = 3). Možni so štirje tipi nadzornih (S) okvirjev: 0,1,2 in 3, ki jih kodira polje *Type*:

- Okvir tipa 0 je pozitivno potrdilo (ang. Receive Ready) in služi za potrjevanje okvirjev, kadar ni podatkovnih okvirjev v nasprotni smeri in *Next* je številka naslednjega pričakovanega okvirja.
- Okvir tipa 1 je negativno potrdilo (ang. Reject) za GBN način in *Next* je številka poškodovanega okvirja.
- Z okvirjem tipa 2 sprejemnik javlja, da sprejemnik ni pripravljen (ang. Receivenot Ready) in prosi za premor, dokler ne pošlje okvir tipa 0 ali 1.
- Tip 3 okvirja je veljaven samo v protokolih HDLC in ADCCP. Služi za selektivno zavrnitev (ang. Selective Reject) in torej potrjevanje SRP.

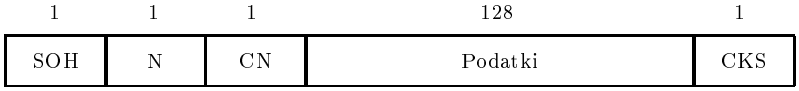
Protokoli se med seboj še najbolj razlikujejo v pogledu neštevilčenih (U) okvirjev. Vsi poznajo ukaz DISC (ang. Disconnect) za javljanje neaktivnosti vozlišča (izven obratovanja), vračanje nazaj v obratovanje SNRM (Set Normal Response Mode), in okvir UA (ang. Unnumbered Acknowledgement) za potrjevanje neštevilčenih okvirjev.

4.6.3 XMODEM

Eden najbolj priljubljenih protokolov za asinhrone prenose podatkov z ali brez modemov med manjšimi računalniki je protokol imenovan XMODEM. Razvil ga je Ward Christensen v poznih sedemdesetih letih, protokol pa je kasneje doživljal spremembe in dopolnila (MODEM2, MODEM7, YMODEM, ZMODEM). Oblika okvirja je skicirana na sliki 89. Protokol je razmeroma primitiven. Uvodni znak je ASCII SOH (binarno 00000001) (Start O Header), njemu sledi osembitna zaporedna številka okvirja in zaradi zanesljivosti njen eniški komplement. Okvirji so na žalost nespremenljive dolžine in nosiljo 128 podatkovnih bajtov. Podatkom sledi osembitna kontrolna vsota podatkovnega dela okvirja. Protokol XMODEM uporablja potrjevanje s čakanjem. Pobuda za prenos pride od sprejemnika, ki periodično vsakih nekaj (deset ali petnajst) sekund oddaja ASCII znak NAK in vabi oddajnik k oddaji. Različica protokola XMODEM-CRC uporablja ciklično preverjenje s polinomom CRC-CCITT. Sprejemnik se v tem primeru javlja z ASCII znakom C. Različica YMODEM pa poleg 128 podatkovnih bajtov dovoljuje tudi okvirje s 1024 podatki. Protokol ZMODEM je ena zadnjih različic protokola

XMODEM, ki nima z njim praktično skoraj nič skupnega, razen da omogoča XMODEM način prenosa.

Oddajnik na povabilo sprejemnika oddaja okvirje dokler sprejemnik ne prekine prenosa z oddajo ASCII znaka CAN (Cancel) ali pa potrdi regularen zaključek. Prenos teče nakako takole. Oddajnik odda okvir. Sprejemnik preveri zaporedno številko. V primeru napake zahteva prekinitev prenašanja z CAN. Nato preveri parnost. Če je pogoj parnosti izpolnjen, pošlje ACK, sicer pa NAK. Ko oddajnik konča z oddajanjem, pošlje EOT in sprejemnik potrdi zaključek z oddajo ACK.

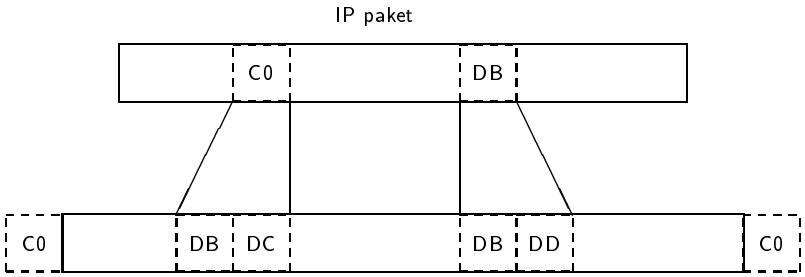


Slika 89: Oblika okvirja protokola XMODEM. Okvir začne z ASCII SOH, zaporedno številko okvirja (N) in njenim eniškim komplementom (CN), vsebuje 128 bajtov podatkov in konča s kontrolno vsoto podatkov.

4.6.4 SLIP

Protokol SLIP (Ang. Serial Link IP) služi za prenos IP paketov po serijskih linijah (točka – točka), na primer za priključitev osebnega računalnika od doma na Internet. V zadnjem času ga je izpodrinil protokol PPP (Point to Point protocol). SLIP je specificiran v dokumentu RFC 1055. Protokol SLIP ne opravlja preverjanja pravilnosti prenosa, to je naloga višjih slojev. Slika 90 prikazuje obliko SLIP okvirja. Okvir konča s posebnim končnim znakom, imenovanim END znak. END znak ima vrednost $C0. Ta isti znak se uporablja tudi na začetku okvirja. Znotaj okvirja se prenaša paket mrežnega sloja, ta je v omrežju Internet IP paket. Da bi sprejemnik podatka $C0 znotraj IP paketa ne tolmačil napačno kot konec okvirja, se znaki s to vrednostjo zamenjajo s kombinacijo SLIP ESC zanka, ki ima vrednost $DB in znaka $DC. Ko sprejemnik med sprejemanjem okvirja naleti na zaporedje $DB $DC ga tolmači kot en sam podatek $C0. Da ne bi prišlo do zamenjave "pravega" para podatkov $DB $DC, se podatek $DB nadomesti s kombinacijo $DB $DD.

Kot vidimo SLIP nima polja, ki bi nosil adreso ali ki bi označeval tip okvirja, niti ne omogoča številčenje okvirjev. Nadzor na nivoju podatkovnega protokola zato ni možen, potreben je torej na višjih slojih. SLIP opravlja zgolj okvirjenje IP paketov in s tem omogoča transparenten prenos IP paketov po serijskih linijah.



Slika 90: Oblika okvirja protokola SLIP.

Literatura

[1] R. Ash, *Information Theory*, John Wiley & Sons, 1965.

[2] R. Cole, *Computer Communications*, Springer-Verlag 1982.

[3] F. da Cruz, B. Catchings, "Kermit: A File-Transfer Protocol for Universities", Part I, Part II, BYTE, Jun. in Jul. 1984.

[4] L. Gyergyek, *Teorija informacij*, ZAFER, Ljubljana 1988.

[5] T. Madron, *Micro-Mainframe Connection*, HOWARD W.SAMS & COMPANY, 1987.

[6] N. Pavešić, *Informacija in kodi*, ZAFER, Ljubljana 1997.

[7] T. Ramabadran, S. Gaitonde, "A Tutorial on CRC Computations", IEEE Micro, Aug. 1988, pp. 62-74.

[8] W. R. Stevens, *TCP/IP Illustrated, Vol. 1*, Addison–Wesley, 1994.

[9] A. Tanenbaum, *Computer Networks*, Prentice-Hall 1989.

[10] J. Walrand, *Computer Communications: A First Course*, Pacific Palisades, CA: Asken Associates, 1991.