Tehničko resenje: Softver za simulaciju koda koji koriguje snop grešaka u bajtu

Rukovodilac projekta: Vladimir Vujičić
Odgovorno lice: Vladimir Vujičić
Autori: A. Radonjić ,V. Vujičić
Razvijeno: u okviru projekta tehnološkog razvoja TR-32019
Godina: 2011.
Primena: 10.12.2011.

Kratak opis

Prilikom prenosa mernih podataka uvek postoji verovatnoća pojave transmisionih grešaka, koje menjaju sadržaj informacija koje se šalju. Ovaj fenomem nastaje kao posledica nesavršenosti komunikacionog medijuma (podložnost uticaju elektromagnetnih zračenja), ali i usled fizičkih oštećenja koja se javljaju tokom dugogodišnje upotrebe. Kako bi se ovaj problem eliminisao ili bar značajno ublažio, u praksi se primenjuju algoritmi zaštitnog kodovanja, koji prijemniku omogućavaju da detektuje i eventualno koriguje greške koje su nastale tokom prenosa.

Medjutim, značajan problem postojećih algoritama leži u činjenici da oni koriste operacije u polju Galoa, što ih čini nepraktičnim za softversku implementaciju (potrebno je dosta vremena kako bi se izvršile procedure kodovanja, dekodovanja i korekcije grešaka). Usled te činjenice, ovi algoritmi se gotovo uvek realizuju upotrebom specijalizovanog hardvera, što značajno poskupljuje njihovu primenu, a ujedno ih čini i neupotrebljivim za ostale primene (hardver je izuzetno teško modifikovati, za razliku od softvera).

S druge strane, da bi algoritmi zaštitnog kodovanja bili efikasno implementirani u softveru, neophodno je da koriste operacije bazirane na celobrojnoj aritmetici, za koje su procesori opšte namene optimizovani. Primer zaštitnog koda, koji koristi ove operacije, jeste intidžerski $(B_lEC)_b$ kod. Pored izuzetno jednostavne realizacije, ovaj algoritam ima zavidne karakteristike u pogledu detekcije i korekcije transmisionih grešaka. Preciznije, primenom ovog algoritma moguće je detektovati i korigovati snopove grešaka unutar bajta, što je sa aspekta prenosa mernih podataka od izuzetne važnosti.

Realizatori:
Fakultet tehničkih nauka u Novom Sadu.

Korisnici:
Fakultet tehničkih nauka u Novom Sadu; ION Solutions d.o.o. kao participant; moguć je prenos tehnologije prema svim zainteresovanim softverskim firmama.

Podtip rešenja:
Softver (M 85)

Stanje u svetu

Osnovna funkcija distribuiranog mernog sistema jeste daljinsko prikupljanje i obrada mernih podataka. Iako mogu imati različite namene, svi distribuirani merni sistemi dele jedinstvenu strukturu sastavljenu od četiri medjusobno povezane celine (slika 1):

Tehnološki proces - npr. proizvodnja i distribucija električne energije.
Hardverski podsistem - celokupan hardver koji se instalira za potrebe nadzora i upravljanja.
Softverski podsistem - celokupan softver, uključujući i pomoćne programe za kontrolu hardvera.
Komunikacioni podsistem - hardverski i softverski elementi koji obezbedjuju pouzdan prenos podataka.

Sl.1. Struktura distribuiranih mernih sistema

U ovako organizovanom sistemu ključnu ulogu igraju komunikacije, jer one obezbedjuju pravovremen prenos mernih podataka. Medjutim, da bi se ova funkcija odvijala korektno, neophodno je koristiti algoritmi zaštitnog kodovanja, koji omogućavaju prijemniku da proveri da li su tokom prenosa desile neke greške. U gotovo svim distribuiranim mernim sistemima ova "provera" se realizuje hardverski: s obe strane komunikacije ugradjuje se poseban deo hardvera zadužen za proveru ispravnosti primljenih podataka. Tipični primeri su uredjaji za nadzor trafostanica *D90^{PLUS}* i *TESLA Model 3000*, koji sadrže hardversku implementaciju CRC kodova. Iako ovo rešenje deluje elegantno, ono dodatno poskupljuje realizaciju mernog sistema. Pored toga, pristup zasnovan na hardveru karakteriše nefleksibilnost, jer je hardver teško modifikovati.

S druge strane, pristup zasnovan na softveru je poželjan, kako zbog toga što se softver može lako modifikovati, tako i zbog konstantnog progresa na polju mikroprocesorske elektronike. Medjutim, glavni problem klasičnih zaštitnih kodova leži u činjenici da oni koriste operacije u polju Galoa, za koje procesori opšte namene nisu optimizovani. Stoga je njihova implementacija vrlo kompleksna, i zahteva dosta procesorskog vremena. Od svih algoritama zaštitnog kodovanja koji se koriste u praksi, jedino se *internet čeksam algoritam* implementira u softveru. Razlog tome je činjenica da ovaj algoritam koristi operacije zasnovane na celobrojnoj aritmetici, koje se brzo i lako izvršavaju na procesorima opšte namene. Medjutim, glavni nedostatak ovog algoritma leži u slabim karakteristikama u pogledu provere integriteta podataka. Tačnije, ovaj algoritam može da detektuje prisustvo snopa grešaka, ali ne i da ih koriguje. To znači da svaki put kada se ova vrsta greške desi, neophodno je ponovno poslati podatke, što

značajno usporava komunikaciju. Ovaj nedostatak moguće je otkloniti korišćenjem intidžerskih $(B_lEC)_b$ kodova, koji su po kompleksnosti i strukturi vrlo slični internet čeksam algoritmu.

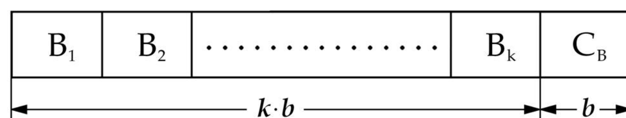
Intidžerski $(B_lEC)_b$ kodovi: osnovni principi

Intidžerski kodovi su originalno namenjeni za upotrebu u specifičnim aplikacijama, kao što su sinhronizacija frejmova, kodovana modulacija, itd. Medjutim, u radu autora A. Radonjić i V. Vujičić, "*Integer Codes Correcting Burst Errors within a Byte*", štampanom u časopisu *IEEE Transactions on Computers* (M 21), pokazano je da se koncept intidžerskih kodova može proširiti i za potrebe ispravljanja transmisionih grešaka (tzv. intidžerski $(B_lEC)_b$ kodovi). Predloženi koncept je vrlo sličan dobro poznatom internet čeksam algoritmu, koji se koristi u internet komunikaciji.

Drugim recima, procedura kodovanja zasniva se na formiranju čekbajta C_B:

$$C_B = [C_1 \cdot B_1 + C_2 \cdot B_2 + \cdots + C_k \cdot B_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot B_i \ (\mathrm{mod}\ 2^b - 1)$$

pri čemu B_i označava aritmetičku vrednosti i-tog bajta, dok koeficijenti C_i predstavljaju cele brojeve koji zadovoljavaju kriterijume u pogledu jednoznačnosti greške. Nakon izračunavanja vrednosti čekbajta C_B, on se kao dodatni bajt stavlja na kraj informacione poruke (slika 2).



Sl.2. Struktura kodne reči kod intidžerskih $(B_lEC)_b$ kodova

Kada ovako definisana poruka stigne do prijemnika, on, na bazi primljenih vrednosti bajtova $\hat{B}_i$ započinje sa istom procedurom

$$\hat{C}_B = [C_1 \cdot \hat{B}_1 + C_2 \cdot \hat{B}_2 + \cdots + C_k \cdot \hat{B}_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot \hat{B}_i \ (\mathrm{mod}\ 2^b - 1)$$

koja mu omogućava da izračuna vrednost sindroma S, odnosno

$$S = [C_{\hat{B}} - \hat{C}_B] \ (\mathrm{mod}\ 2^b - 1)$$

Logika kojom se rukovodi prijemnik je vrlo jednostavna: ukoliko je $S = 0$, podaci su preneti bez greške, dok se u suprotnom slučaju desila neka greška. Ukoliko je reč o snopu greške koja se desila unutar bajta, prijemnik će ovu grešku korigovati koristeći izraze:

$$B_i = [\hat{B}_i + e_w] \ (\mathrm{mod}\ 2^b - 1)$$

$$C_B = [\hat{C}_B + e_w] \ (\mathrm{mod}\ 2^b - 1)$$

pri čemu su vrednosti parametara i i e_w unapred sačuvane u luk-ap (eng. look-up) tabeli. Na slici 3 dat je primer jedne takve tabele za slučaj (24, 16) intidžerskog $(B_2EC)_8$ koda.

n_c	Elements of set $\xi_{8,2}$ (b bits)	Location of erroneous byte ($\lceil \log_2(n+1) \rceil$ bits)	Elements of set $\varepsilon_{8,2}$ ($e_{\overline{w}}$) (b bits)
1	1	3	1
⋮	⋮	⋮	⋮
16	28	1	251
17	31	1	32
⋮	⋮	⋮	⋮
41	108	2	243
42	111	2	16
⋮	⋮	⋮	⋮
90	254	3	254

$\xi_{8,2}$	1	2	3	4	6	8	12	16	24	32	48	63	64	96	127	Third byte in error
	128	159	191	192	207	223	231	239	243	247	249	251	252	253	254	
	7	14	21	28	31	42	56	62	69	81	84	87	93	112	124	First byte in error
	131	143	162	168	171	174	186	193	199	213	224	227	234	241	248	
	9	18	27	33	36	39	54	57	66	72	78	99	108	111	123	Second byte in error
	132	144	147	156	177	183	189	198	201	216	219	222	228	237	246	

$e_{\overline{w}}$	1	2	3	4	6	8	12	16	24	32	48	63	64	96	127	Third byte in error
	128	159	191	192	207	223	231	239	243	247	249	251	252	253	254	
	254	253	252	251	32	249	247	64	63	207	243	24	96	239	128	First byte in error
	127	16	159	231	12	48	192	191	8	6	223	4	3	2	1	
	254	253	252	223	251	24	249	192	191	247	48	159	243	16	128	Second byte in error
	127	239	12	96	207	8	64	63	6	231	4	32	3	2	1	

Sl.3. Luk-ap tabela za intidžerski $(B_2EC)_8$ kod: struktura (levo) i odgovarajući elementi (desno)

Važno je napomenuti da luk-ap tabele za potrebe intidžerskih $(B_lEC)_b$ kodova zahtevaju memorijske resurse izmedju 32KB i 155KB, što je zanemarljivo imajući u vidu veličinu keš memorija kojima raspolažu moderni procesori opšte namene (nekoliko MB).

Softver za simulaciju (24, 16) intidžerskog $(B_2EC)_8$ koda realizovan u MATLAB-u

```
function [] = Integer

koef=[7 9];
b=8;
L=length(koef); %broj informacionih bajtova;
dataword=L*b; %duzina podataka (u bitima);
send=randint(1,dataword); %generisanje slucajnog binarnog niza duzine dataword;
Q=[];
CB=0;
for i=1:L %procedura kodovanja;
   BB=fliplr(send(1+(i-1)*b:i*b));
   B=bi2de(BB); %aritmeticka vrednost bajta;
   CB=mod(CB+B*koef(i),2^b-1); %kumulativno izracunavanje vrednosti CB (po modulu 2^b-1);
end

fprintf('\nKODOVANJE:\nAritmeticka vrednost cek-bajta CB iznosi:\nCB = %i\n\n',CB)
fprintf ('Sada unesite vrstu greske i to:\n1)e = 0 za prenos bez greske\n2)e = 1 za prenos sa greskom\n')
e=input('\nUnesena vrednost je e = ');
if (e==0) %prenos bez greske;
   E=randerr(1,length(send),0);
elseif (e==1) %prenos sa  greskom;
   E=randerr(1,length(send),1);
end

receive=xor(send,E); %XOR operacija izmedju podataka i vektora greske (dodavanje greske);
```

```
CBprim=0;  %vrednost CB na prijemu;
for i=1:L %procedura kodovanja;
   BB1=fliplr(receive(1+(i-1)*b:i*b));
   B1=bi2de(BB1); %aritmeticka vrednost bajta;
   CBprim=mod(CBprim+B1*koef(i),2^b-1); %kumulativno izracunavanje vrednosti CB (po modulu 2^b-1);
end
S=mod(CBprim-CB,2^b-1); % aritmeticka vrednost sindroma S (po modulu 2^b-1);

e=[1 2 3 4 6 8 12 16 24 32 48 63 64 96 127 128 159 191 192 207 223 231 239 243 247 249 251 252 253 254 7 14 21 28 42 56
84 112 168 224 81 186 193 162 124 131 93 62 69 174 31 87 143 171 199 213 227 234 241 248 9 18 27 36 54 72 108 144 216
33 177 57 66 99 123 132 156 189 198 78 222 39 111 147 183 201 219 228 237 246]; %pretpostavlja se da se ove vrednosti
nalaze u luk-ap tabeli;

fprintf('\nDEKODOVANJE:\nAritmeticka vrednost cek-bajta CBprim iznosi:\nCBprim = %i\n',CBprim)
fprintf('\nNa osnovu dobijenih vrednosti dekoder izracunava da je:\nS = %i\n',S)

if (S==0)
   fprintf('Iz vrednosti sindroma S dekoder zakljucuje da nema gresaka.\n\n')
elseif (S~=0)
   for i=1:length(e)
      if (S==e(i))&(i<31)
         error=e(i);
         fprintf('\nGreska desila na cekbajtu\n')
         fprintf('\nAritmeticka vrednost greske iznosi:\ne = %i\n',error)
         fprintf('\nGreska se ispravlja koriscenjem izraza (%i + %i) mod(%i) = %i.\n\n',CBprim,error,2^b-1,CB)
      elseif (S==e(i))&(i>30)&(i<61)
         error=e(i-30);
         fprintf('\nGreska desila na bajtu broj 1.\n')
         fprintf('\nAritmeticka vrednost greske iznosi:\ne = %i\n',error)
         fprintf('\nGreska se ispravlja koriscenjem izraza (%i + %i) mod(%i) = %i.\n\n',bi2de(fliplr(send(1:b))),error,2^b-
1,bi2de(fliplr(receive(1:b))))
      elseif (S==e(i))&(i>60)
         error=e(i-60);
         fprintf('\nGreska desila na bajtu broj 2.\n')
         fprintf('\nAritmeticka vrednost greske iznosi:\ne = %i\n',error)
         fprintf('\nGreska se ispravlja koriscenjem izraza (%i + %i) mod(%i) = %i.\n\n',bi2de(fliplr(send(1+b:2*b))),error,2^b-
1,bi2de(fliplr(receive(1+b:2*b))))
      end
   end
end
```

Softver za simulaciju koda koji koriguje snop grešaka u bajtu razvijen je na Fakultetu tehničkih nauka u Novom Sadu, u okviru tekućeg projekta br. TR-32019 kod Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije.

Štampano – Decembar 2011.