WLSB kodovanje i CRC zaštita kod ROHC protokola

Nenad Božić

***Sadržaj* — U radu je dat kratak opis protokola za robusnu kompresiju zaglavlja (Robust Header Compression - ROHC) i opis njegovih funkcija. Dobijeni rezultati ukazuju na potrebu za implementacijom ROHC protokola u mrežama sledeće generacije. Stavljen je poseban akcenat na prozorsko kodovanje najmanje značajnih bita (Window Based Least Significant Bit - WLSB) i na cikličnu proveru redundantnosti (Cycle Redudancy Check - CRC). U okviru poglavlja o WLSB kodovanju dat je opis urađene simulacije i predlog efikasnog hardverskog rešenja kako bi se smanjila upotreba komplikovanih matematičkih operacija.**

***Ključne reči* — CRC, *kodovanje*, *kompresija*, ROHC, *zaglavlje paketa*, WLSB**

I. UVOD

KOMPRESIJA zaglavlja se koristi kako bi se veličina zaglavlja u mrežama svela na svega 2 do 4 bajta, koja za IPv4 pakete zajedno sa UDP (8 bajta) i RTP (12 bajta) nekomprimovana zauzimaju oko 40 bajta.

I pre protokola za robusnu kompresiju zaglavlja (ROHC) postojali su protokoli ove vrste ali se ROHC protokol pokazao kao efikasno rešenje u mrežama sa velikom verovatnoćom greške (Bit Error Rate - BER) i sa velikim vremenom puta između čvorova (Round Trip Time - RTT) zato što dodaje neophodnu robusnost. Opis ROHC protokola, zajedno sa njegovim stanjima i modovima rada dat je u poglavlju II [1]-[6].

Algoritam za prozorsko kodovanje najmanje značajnih bita (WLSB - Window Based Least Significant Bit) posebno je razvijen za ROHC protokol i pogodan je za polja koja se inkrementuju kakvo je polje sekvencijalnog broja. U poglavlju III je dat opis WLSB algoritma [1], [7]-[9]. Takođe data je i ideja za efikasnu implementaciju kako bi se rastereretio procesor i iskoristila pomoć hardverskih komponenti.

U poglavlju IV [1], [10] dat je opis algoritma za cikličnu proveru redundantnosti (CRC - Cycle Redudancy Check) koji se koristi za zaštitu podataka. Poruka se rastavlja na statički i dinamički sadržaj i primenjuje se CRC zaštita i nad jednom i nad drugom grupom polja.

Urađena je i simulacija WLSB algoritma i nad porukom je primenjena CRC zaštita. Simulacija je rađena u programskom jeziku C++ kako bi se prikazala ostvarena kompresija i poboljšanje performansi korišćenjem efikasne implementacije.

Na kraju rada je dat zaključak sa analizom dobijenih rezultata i dati su pravci daljeg mogućeg razvoja kompresije zaglavlja, a samim tim i ROHC protokol.

Nenad Božić, Fakultet tehničkih nauka u Novom Sadu, Srbija (telefon: 381-65-5522979; e-mail: nbozicns@hotmail.com).

II. ROHC PROTOKOL

Kompresija zaglavlja je ostvariva zbog velike redundanse u poljima zaglavlja, kako u samom paketu tako i između paketa koji se šalju jedan za drugim a pripadaju istom toku paketa. U početku se šalje par paketa bez kompresije kako bi se uspostavio kontekst zaglavlja na obe strane veze. Ovih par paketa nose informacije o statičkim poljima i dinamičkim poljima, kao i o pravila njihovog ponavljanja u paketskom zaglavlju. Ove informacije se koriste u kompresoru u cilju najefikasnijeg mogućeg komprimovanja paketa i u dekompresoru radi dekomprimovanja u originalno stanje [1], [8].

Stanje kompresora je diktirano karakteristikama protoka paketa, kao što su reorganizacija ili gubitak pre kompresije, i izmenama pravila ponavljanja polja u zaglavlju zbog ponašanja aplikacije ili zbog operativnog sistema. Slično stanje dekompresora je diktirano stanjem veza (BER i RTT). U slučaju pojavljivanja greške, i ako povratni kanal postoji, dekompresor šalje paket sa povratnim informacijama kompresoru i tako utiče na njegovo stanje rada. Za ROHC su definisani i određeni profili kompresije kao i određena stanja u kojima se mogu naći kompresor i dekompresor.

A. Profili kompresije

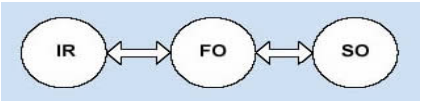
Svaka podržana vrsta paketa za kompresiju zaglavlja je definisana svojim profilom u ROHC specifikaciji. Profil u potpunosti specificira paketski format, logiku prelazaka između stanja, vrste radnji u svakom stanju, kao i metode kodovanja korišćene za svako polje zaglavlja u korišćenom profilu. Razlikujemo četiri profila kompresije:

- **profil 0 (ROHC Uncompressed) –** vrši kompresiju paketa koji ne mogu biti komprimovani ni sa jednim od narednih profila.
- **profil 1 (ROHC RTP) –** vrši kompresiju paketa sa IP/UDP/RTP protokol zaglavljem (**glavni profil**)
- **profil 2 (ROHC UDP) –** vrši kompresiju paketa sa IP/UDP protokol zaglavljem.
- **profil 3 (ROHC ESP) –** vrši kompresiju paketa sa IP/ESP protokol zaglavljem.

B. Stanja kompresora

ROHC kompresor ima tri stanja rada: **Initialization and Refresh (IR), First Order (FO)** i **Second Order**

(SO) [6]**. S**tanja rada ROHC kompresora opisuju nivo sigurnosti ispravanog prijema poruke na strani dekompresora.



Slika 1. Dijagram stanja kompresora

Kompresor uvek počine rad u najnižem stanju (IR). IR stanje nam služi da bi inicijalizovali statički deo sadržaja na strani dekompresora, ili da bi ispravili grešku kada do nje dođe. U IR stanju se šalju nekomprimovani paketi.

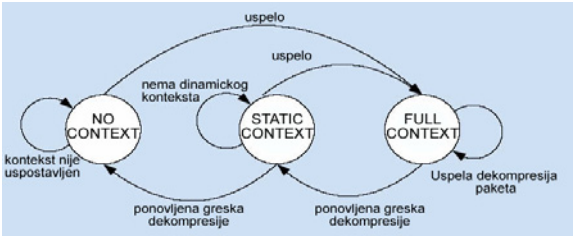
Svrha FO stanja je da efikasno otkloni neregularnosti u paketskom toku. Kompresor ulazi u ovo stanje ili iz IR ili iz SO stanja uvek kada zaglavlja paketskog toka ne odgovaraju njihovom prethodno ustanovljenom načinu promene. Kompresor ostaje u FO stanju dok ne ustanovi da je dekompresor dobio sve parametre za novi način promene.

Sledeće stanje (SO) je stanje u kome je kompresor optimalan. Kompresor ulazi u SO stanje onda kada je zaglavlje koje treba da se komprimuje u celosti predvidivo iz sekvencijalnog broja (SN - RTP Sequence Number) i kada je kompresor dovoljno siguran da dekompresor ima sve potrebne parametre kako bi od SN polja dekomprimovao i ostala polja.

C. *Stanja dekompresora*

ROHC dekompresor ima takođe tri stanja rada: **No Context** (stanje bez konteksta)**, Static Context** i **Full Context.** Ova stanja odgovaraju stanjima kompresora.

Dekompresor počinje rad u najnižem stanju, bez konteksta, s obzirom da na početku nema nikakve informacije o paketskom toku [6]. Uspešna IR dekompresija će stvoriti informacije o kontekstu zaglavlja sa dekompresione strane.



Slika 2. Dijagram stanja dekompresora

U ovom trenutku dekompresor može da se prebaci u Full Context stanje jer je dobio dovoljno informacija o statičkom i dinamičkom sadržaju. Kada je u Full Context stanju, dekompresor se vraća u niže stanje samo u slučaju pojave greške. Kada se greška pojavi prvo se vraća u stanje Static Context u nadi da će grešku ispraviti, a ukoliko to uspe vraća se skroz u prvo stanje, No Context i traži od kompresora ponovno slanje IR paketa.

D. *Modovi rada*

ROHC ima 3 moda rada čiji izbor je definisan parametrima kao što su postojanje povratnog kanala, verovatnoća greške i varjacija veličine zaglavlja [1], [6]. Modovi rada su: **U-mod (jednosmerni mod), O-mod (dvosmerni optimistički mod)** i **R-mod (dvosmerni pouzdani mod).** Kompresija mora početi u jednosmernom modu a prelazak u bilo koji dvosmerni mod se izvršava čim paketi stignu do dekompresora koji šalje paket sa povratnim informacijama za željeni mod rada. ROHC kompresor uvek počinje rad u U-modu i IR stanju.

Kada je ROHC u U-modu rada paketi se šalju samo u jednom smeru, od kompresora do dekompresora. Ovaj mod rada se koristi kod veza gde ne postoji povratni kanal. Nedostatak povratnog kanala se rešava upotrebom brojača i timeout mehanizma. Kompresor koristi optimistički pristup tako što šalje određeni broj informacija u jednom stanju a, kada on postane veći od brojača za to stanje, on prelazi u više stanje. Periodični time-out mehanizam se koristi da bi se kompresor prebacio u niže stanje i poslao FO i IR pakete kako bi se smanjila mogućnost greške.

Dvosmerni optimistički mod je sličan jednosmernom modu. Razlika je u tome što se povratni kanal koristi da bi dekompresor slao zahteve za otklanjanje grešaka i (opciono) da bi potvrđivao bitna osvežavanjanja konteksta na strani dekompresora (ali ne i za svako uspešno primanje sekvencijalnog broja). Potvrda o dobrom osvežavanju konteksta se uvek mora slati za IR pakete dok je za ostale, slanje ove poruke opciono.

Dvosmerni pouzdani mod se u mnogome razlikuje od prethodna dva moda. Za razliku od njih, dvosmerni pouzdani mod se temelji na sledeća tri principa: princip sigurne veze, potreba za osvežavanjem konteksta i princip negativne potvrde. Kompresor dobija sigurnost o dobrom prijemu na dekompresionoj strani samo kroz pakete sa povratnim informacijama (potvrde - ACK i negativne potvrde - NACK). Cilj ovog moda je da obezbedi maksimalnu robusnost kako bi smanjio verovatnoću prostiranja grešaka i prostiranja oštećenja, ali uz optimalnu kompresiju.

III. WLSB KODOVANJE

A. *LSB kodovanje*

Kodovanje najmanje značajnih bita (Least significant bit - LSB) se koristi za polja u zaglavlju koja se malo menjaju [1], [7] (npr. polja koja se inkrementuju). Jednostavno, samo k najmanje značajnih bita za neko polje se prenose. Dekompresor dekoduje originalnu vrednost koristeći referentnu vrednost, v_{ref} koja mora da sadrži ostale bite koji nisu preneseni. Tačnost je zagarantovana ukoliko kompresor i dekompresor koriste iste interpretacione intervale u kojima se originalna vrednost kreće i prenesenih k najmanje značajnih bita jedinstveno predstavljaju originalnu vrednost u tom interpretacionom intervalu.

LSB interpretacioni interval može biti predstavljen kao funkcija:

$$f(v_{ref}\ ;\ k) = [v_{ref} - p\ ;\ v_{ref} + (2^k - 1) - p\ ] \quad (1)$$

Za bilo koje k, k poslednjih bita moraju jedinstveno da predstavljaju vrednost u $f(v_{ref}\ ;\ k)$.

Parametar p je parametar pomeranja intervala i može da se menja za veću efikasnost. Za nas je zanimljivo njegovo prilagođavanje za polja koja se inkrementuju za 1, kao što je polje SN. Tada p uzima vrednost -1.

Kompresor koristi kao v_{ref} poslednju komprimovanu vrednost zaštićenu sa CRC zaštitom. Dekompresor koristi za v_{ref} poslednju dekomprimovanu vrednost koja je prošla CRC proveru. Vrednost za k se bira kao minimalna moguća vrednost tako da vrednost v pada u interpretacioni interval $f(v_{ref}; k)$. Ova funkcija se definiše kao $g(v_{ref}; v)$.

B. WLSB kodovanje

Prozorsko kodovanje najmanje značajnih bita (WLSB) dodaje robusnost LSB kodovanju [1] u slučajevima kada kompresor nije u mogućnosti da donese odluku o pravoj referentnoj vrednosti, v_{ref}, koja se koristi na strani dekompresora. Kompresor čuva klizajući prozor (sliding window) mogućih vrednosti i računa k tako da kandidati za v_{ref} daju korektno dekodovanje vrednosti v. Imamo v_{refmin} i v_{refmax} kao minimalnu i maksimalnu referentnu vrednost u prozoru, i k se bira na sledeći način:

$$k = \max \left(g(v_{refmin}; v) ; g(v_{refmax}; v) \right) \quad (2)$$

Sa veličinom prozora rukujemo na osnovu paketa sa povratnim informacijama od dekompresora kao i na osnovu maksimalnog broja vrednosti u prozoru.

C. Izbor prozora

Parametar dužine prozora je promenljiv parametar i njegovim menjanjem možemo dobiti dobre rezultate kompresije u zavisnosti od karakteristika linka. Stepen kompresije („compression ratio“) je veličina koja označava koliko puta je kraća komprimovana od nekomprimovane kodne reči. Memorijsku uštedu („saving“) ukazuje na to kolika je ušteda memorijskog prostora izražena u procentima u odnosu na korišćenje nekomprimovih reči.

TABELA 1: IZBOR VELIČINE PROZORA

Početna vr. / vel. prozora	Početna vrednost 2 bita		Početna vrednost 8 bita		Početna vrednost 15 bita	
	Ratio	Saving	Ratio	Saving	Ratio	Saving
Prozor 4	2.68	62.65%	3.16	68.3%	5.3	81.13%
Prozor 8	2.01	50.21%	2.37	57.88%	3.99	74.93%
Prozor12	1.61	38.04%	1.9	47.53%	3.2	68.8%

Treba napomenuti da je za potrebe simulacije izabrana sekvenca od 500 ulaznih vrednosti a korištene su početne vrednosti sa predstavom od 2, 8 i 12 bita. Vidimo iz tabele da se bolji rezultati ostvaruju sa većim vrednostima, jer se koduje razlika između ulazne vrednosti i granice prozora, što će kod vrednosti koje se inkrementuju za sva tri slučaja dati istu vrednost. Samim tim je logično da se kod vrednosti koja nekomprimovana predstavlja sa najviše bita ostvari i najbolja kompresija i najveća ušteda.

Prema rezultatima testa, prikazanim u tabeli 1, manja veličina prozora daje bolji stepen komresije i veću uštedu prostora. Treba uzeti u obzir da je glavna područje rada ovog protokola na linkovima koji imaju visok BER i veliko RTT zbog robusnosti koju je ROHC dodao u kompresiju zaglavlja, tako da treba napraviti dobar balans između zadovoljavajuće kompresije i visoke robusnosti kako bi se smanjila verovatnoća gubitaka podataka. Iz tih razloga, rešenje sa prozorom veličine 8 u ovom primeru, bilo bi najbolje jer daje dobar stepen kompresije i mogućnost pojavljivanja 7 uzastopnih grešaka koje ne remete prijem osmog paketa.

D. Tok kompresije i dekompresije

Iz jednačine (2) [7] vidimo da kada kompresor treba da komprimuje vrednost on ustvari računa maksimalnu udaljenost od jedne od granica prozora na sledeći način:

$$r = \max \left(|v - v_{max}|, |v - v_{min}| \right) \quad (3)$$

Sada se na osnovu (3) računa k preko sledeće formule:

$$k = \lceil \log_2 (2 * r + 1) \rceil \quad (4)$$

Kada smo dobili vrednost za k kompresor šalje poslednjih k bita i dodaje vrednost v u prozor. Proverava se da li je ova vrednost novi minimum ili maksimum prozora, i ukoliko jeste čuva se nova vrednost.

Dekompresor, sa druge strane, prima ovih k bita, ima prethodno primljenu vrednost kao referentnu, i na osnovu ova dva podatka dekomprimuje vrednost.

E. Efikasna implementacija

Kao što vidimo iz (4) da bi izračunali k treba da koristimo logaritam, koji je dosta skup za softversku implementaciju zbog potrebe aproksimacione aritmetike i korišćenja lookup tabela [8]. Ovo se može izbeći korišćenjem jednostavnih bitskih operacija (&, ^, <<, >>).

```
int wlsb :: number_of_bits ( int in )
{
  int counter = 0 ;
/*shifts left input number until it becomes
0, in variablee counter it saves number of
cycles. This cycle which gives on the end
position of leftmost 1 in bit representation
  while ( in!=0 )
  {
    in = in >> 1 ;
  counter ++ ;
  }
  return counter ;
}
```

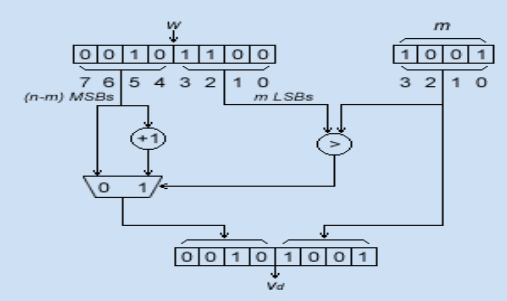
Slika 3. Funkcija number_of_bits

Logaritam sa osnovom 2 u kombinaciji sa najvećim celim od te vrednosti se, korišćenjem bitske logike, može interpretirati kao lociranje pozicije najznačajnije jedinice u bitskoj predstavi broja (prve leve jedinice). Na slici 3 je prikazan deo koda, tačnije funkcija number_of_bits, koja prima neki broj kao ulaznu vrednost a kao rezultat izbacuje broj bita sa kojima predstavljamo taj broj. Ona radi samo pomoću bitskih operacija i na taj način je omogućena hardverska implementacija ove funkcije.

Ova funkcija pomera ulaznu vrednost (*in*) u levo dok ona ne postane 0. U isto vreme u promenljivoj *counter* se beleži broj prođenih ciklusa u while petlji. Na ovaj način ćemo posle određenog broja ciklusa dobiti da je *in* = 0 i to će biti uslov za izlazak iz petlje. Samo ostaje da se kroz povratnu vrednost prosledi pozicija najznačajnije jedinice u broju a to nam govor baš promenljiva *counter* .

Kod dekodera pratimo istu logiku kako bi izbegli korišćenje logaritma. Šema dekodovanja je prikazana na sl. 4 [8], gde je dat i primer osmobitnog broja kao referentne vrednosti, a primili smo 4 bita. Ako sa |w| označimo poslednje dekodovani broj, kada nam stigne nekih m bita, razdvajamo |w| na n-m značajnih bita ($|w|_{n-m}$)

i m najmanje značajnih bita ($|w|_m$) jednostavnim bitskim pomeranjem, pri čemu je n broj bita od $|w|$. Poredimo vrednost $|w|_m$ sa vrednošću m pristiglih bita, i ako je $|w|_m$ veće, uvećavamo $|w|_{n-m}$ za 1. Ostaje nam sada samo da spojimo m pristiglih bita sa bazom poslednje dekodovanog broja ($|w|_{n-m}$) i dobijamo traženi dekodovani broj, korišćenjem samo jednostavnih bitskih operacija.



Slika 4. Šema dekodovanja

Za potrebe ovog rada razvijena je softverska simulacija i jednog i drugog načina implementacije i u tabelu 2 su uneta vremena obrade u jednom i u drugom slučaju. Kao što vidimo, ostvaruje se ubrzanje rada izbegavanjem logaritma, ali ova ušteda ne predstavlja značajnu dobit. Glavna prednost ove efikasne implementacije je korišćenje hardverske pomoći pri računanju kodovanog i dekodovanog broja i izbegavanje komplikovanog softverskog računa ovih operacija. Pošto smo ovaj račun sveli samo na par bitskih operacija (&, ex-or, <<, >>) ovaj algoritam nam omogućava hardversku implementaciju i što je više moguće rasterećenje procesora. Šema se neznatno komplikuje ali se izbegava korišćenje logaritma što je i bila glavna ideja.

TABELA 2: POREĐENJE VREMENA OBRADE

Veličina sekvence / vrsta algoritma	Sekvenca 1000	Sekvenca 5000	Sekvenca 10000
Bitske operacije (&,ex-or,<<)	0.090618 sec	0.457465 sec	0.91643 sec
Matematičke operacije ($\log_2$)	0.104487 sec	0.4631 sec	0.92444 sec

IV. CRC ZAŠTITA

ROHC koristi cikličnu proveru redundantnosti (CRC) za detekciju i otklanjanje grešaka [1], [10]. CRC je algoritam koji vrši deljenje neke sekvence prostim polinomima određenog stepena i sa strane kodera i sa strane dekodera i poredi ove rezultate. Ukoliko se dobije ista vrednost sve je u redu i može se pristupiti daljem dekodovanju.

Kod ROHC se cela poruka rastavlja na statički i dinamički sadržaj i primenjuje se CRC zaštita prvo na jednom a potom na drugom delu. Rezultat deljenja sa strane kodera se upisuje u rezervisano CRC polje. Na strani dekodera se radi isti postupak, samo što se prvo deli pa se potom rezultat poredi sa vrednošću u CRC polju.

Kod ROHC se koristi 3 vrste CRC polinoma, nekomprimovani podaci se štite CRC-8 zaštitom dok se ostali podaci štite ili CRC-3 ili CRC-7 zaštitom, u zavisnosti od dužine kodnih reči.

V. ZAKLJUČAK

S obzirom da mreže sledeće generacije spajaju sve funkcije u jednu jedinstvenu mrežu [5]-[6] sve je veća potreba za ovakvim i sličnim protokolima. Mreža se razvija i postaje sve veća tako da čuvanje mrežnih resursa postaje važna stavka. Takođe sigurna komunikacija je bitan zahtev a ROHC upravo na ovim poljima ostvaruje dobre rezultate zbog svoje robusnosti. Ostavljeno je i prostora za dalji razvoj, profili kompresije su programabilni i samo je pitanje vremena kada će se njihov broj proširiti.

ZAHVALNICA

Zahvaljujem se firmi "Zesium mobile", Novi Sad što mi je dala ideju i temu za rad i što mi je omogućila da uz pomoć zaposlenih u firmi dovedem ovaj rad do kraja i dobijem potrebne rezultate.

Takođe zahvaljujem se univerzitetu "Prince of Songkhla", Hat Yai, Thailand gde sam završio ovaj rad i profesoru Elz Robertu koji mi je u tome dosta pomogao.

LITERATURA

[1] IETF Network working group, "Robust Header Compression (ROHC): Framework and Four Profiles: RTP, UDP, ESP, and uncompressed", (RFC 3095), July 2001.
[2] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links, Request for Comments 1144", February 1990.
[3] IETF Network working group, "ROHC, requirements on TCP/IP header compression", (RFC 4163), August 2005
[4] IETF Network working group, "Requirements for robust IP/UDP/RTP header compression", (RFC 3096), July 2001
[5] EFFNET AB Whitepaper, "An introduction to IP header compression", February 2004, Available: http://www.effnet.com
[6] EFFNET AB Whitepaper, "The concept of robust header compression - ROHC", February 2004, Available: http://www.effnet.com
[7] ROHC Workgroup, "TCP – aware robust header compression (taroc)",July 20, Available draft-ietf-rohc-tcp-taroc-02.txt
[8] D.E. Taylor, E. Herkersdorf, A. Doring, G. Dittman, "Header Compression (ROHC) in NGN processors", WUCSE-204-70, September 13 , 2002
[9] F. Fitzek, S. Rein, P. Seeling, M. Reisslein, "ROHC performance for multimedia transition over 3G/4G wireless network,", Wireless personal communications 2005 32: 23-41.
[10] http://en.wikipedia.org/wiki/Cyclic_redundancy_check (Wikipedia)

ABSTRACT

This paper gives short overview of Robust Header Compression protocol (ROHC) and its functions. It also introduces the need of this protocol in NGN.

Main point of this work is Window Based Least Significant Bit algorithm (WLSB) which is developed specially for this protocol. This paper explains this algorithm, tests different size of window using simulation in C++ developed for this work, and gives an idea for efficient implementation which uses hardware help. On the end we also explain Cycle Redundancy Check (CRC) used in ROHC for detection and recovery after errors.

WLSB CODING AND CRC PROTECTION IN ROHC PROTOCOL

Nenad Bozic