MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Implementace FEC pro UltraGrid

BAKALÁRSKA PRÁCA

Martin Sokol

Brno, podzim 2009

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: RNDr. Petr Holub, Ph.D.

Zhrnutie

Cieľom tejto bakalárskej práce je implementovať podporu pre doprednú opravu chýb (Forward Error Correction – FEC) do platformy pre nízkolatenčné prenosy nekomprimovaného HD videa UltraGrid. Výsledkom je upravená implementácia platformy UltraGrid, ktorá obsahuje modul umožňujúci doprednú opravu chýb na paketovej úrovni.

Kľúčové slová

FEC, UltraGrid, LDPC, HD video, parita, latencia

Obsah

Kapitola 1

Úvod

Komunikácia skupín prostredníctvom počítačovej siete je dnes už realitou. Vzhľadom na to, že ide o vzájomnú komunikáciu viac než dvoch subjektov, je potreba zabezpečiť vhodnú infraštruktúru, ktorá túto komunikáciu umožní. Vhodným prostriedkom, ktorý nám to umožňuje je využitie služby multicast. Avšak multicastové siete sú len podmnožinou celosvetovej siete Internet. Preto je niekedy potrebné nájsť iné prostriedky (hardwarové alebo softwarové). Vhodným prostriedkom môže byť napríklad použitie reflektoru paketov, ktorý môže nahradiť službu multicast tam, kde nie je k dispozícii.

Ak požadujeme komunikáciu vo vyššej kvalite, multicastové siete nestačia. Potrebujeme prenášať veľké množstvo dát medzi jednotlivými účastníkmi – pri nekomprimovanom HD videu je to dátový tok približne 1.5Gbps pre jeden stream – a preto aj prenosová linka musí byť dimenzovaná na takéto vysoké dátové toky. Pre ilustráciu, pri videokonferencii troch strán je potrebné zaistiť jeden výstupný dátový tok a dva vstupné, čo znamená súčasne spracovať 4,5Gb informácií za sekundu. To je značná záťaž na sieť a v súčasnej dobe nie je takáto komunikácia ani zďaleka príliš rozšírená.

Výhodou nekomprimovaného HD videa je nízka latencia a zachovanie maximálnej možnej kvality snímkou. Takisto pri výpadku vnikajú menšie problémy s artefaktami v obraze. Naopak nevýhodou je nutnosť využívania veľkej šírky pásma. Určité podmienky sa kladú aj na systém, ktorý takéto video spracúvajú, či už ide o sieťovú kartu, ale aj na ďalší hardware, ktorý video spracúva. Pri videokonferenciách je potrebné prenášať aj zvukovú stopu, avšak tá nemá také veľké nároky na kapacitu siete ako prenos nekomprimovaného videa vo vysokom rozlíšení.

Pri strate jedného paketu, prípadne skupiny paketov, vznikajú v obraze rôzne fragmenty (chyby obrazu). Nakoľko retransmisia je pri real-time prenose do značnej miery problematická, hľadajú sa iné metódy a postupy na ich eliminovanie. Často však ide len o zakrytie stratenej informácie. Príkladom môže byť použitie metódy nazývanej interleaving. Tieto metódy dokážu viac či menej zakryť obrazové chyby, ale nedokážu ich celkom odstrániť.

Dobrým riešením je použitie samo opravných kódov, ktoré dokážu detekovať stratu paketu, prípadne viacerých paketov, a následne rekonštruovať jeho obsah z určitej redundantnej informácie poslanej spolu s pôvodnými paketmi. Samozrejme tieto kódy majú svoje obmedzenia daných množstvom redundantných dát a použitého algoritmu.

Vhodným riešením je kombinácia oboch prístupov, keď samo opravné kódy opravia väčšinu stratených dát a následné ďalšie zakrytie chýb obrazu je potom jednoduchšie, prípadne nie je vôbec nutné, ak sa FEC kódom podarilo opraviť všetky stratené dáta.

1.1 Cieľ práce

Cieľom tejto práce je implementovať podporu pre doprednú opravu chýb v existujúcej platforme UltraGrid umožňujúcej nízkolatenčné prenosy nekomprimovaného HD videa. Výsledná upravená implementácia platformy UltraGrid bude schopná využiť výhody použitia samo opravných kódov. Bude schopná obnoviť časť alebo všetky chýbajúce pakety stratené počas prenosu sieťou, a tým zvýšiť kvalitu zobrazovaného videa bez nutnosti jeho retransmisie prípadne iných opravných techník.

1.2 Štruktúra práce

V druhej kapitole je stručne zhrnutá problematika kódov s doprednou opravou chýb, možnosti ich využitia, prípadné výhody a nevýhody použitia týchto kódov. Priestor bude venovaný aj základnému rozdeleniu samoopravných kódov a bližší pohľad na najzaujímavejšie z nich. Takisto je priestor venovaný základnému princípu fungovania použitého typu samoopravného kódu.

Tretia kapitola stručne popisuje použitú implementáciu LDPC kódov, porovnanie výkonu s Reed-Solomonovými kódmi, vplyv nastavenia rôznych parametrov na výkon kódovania a dekódovania.

Štvrtá kapitola sa zameriava na upravenú platformu UltraGrid, akým spôsobom bol implementovaný existujúci kód pre doprednú opravu chýb. Obsahuje aj načrtnutie možností ďalšieho vývoja či prípadného zvýšenia výkonu.

Piata kapitola predstavuje aplikácie, ktoré sa buď využívajú na prenos videa vo vysokom rozlíšení (a taktiež implementujú doprednú opravu chýb), alebo sú nejakým spôsobom využívané pri prenose multimediálneho obsahu a implementujú rôzne techniky odstránenia alebo zakrývania dátových strát vzniknutých pri prenose.

1.3 Použité technológie

Do platformy UltraGrid je doimplementovaná podpora existujúcej knižnice samoopravných kódov. Použité kódy sa vyznačujú relatívne dobrou komplexnosťou a efektivitou a tým umožňujú aj dosiahnutie vyšších prenosových rýchlostí, ktoré sú potrebné pri prenose nekomprimovaného videa vo vysokom rozlíšení. Použitie upravenej platformy UltraGrid je možné pod systémom Linux. Použitý kodek však umožňuje jeho skompilovanie aj pod inými operačnými systémami, ako sú napríklad MS Windows, Solaris a FreeBSD. Použitý kodek je naďalej vyvíjaný a tak sa nevylučuje rozšírenie aj na platformu MacOS X, ktorú platforma UltraGrid takisto podporuje.

1.4 Programovacie jazyky

Práca bola napísaná v jazykoch C a C++, pričom platforma UltraGrid je z prevažnej časti napísaná v jazyku C, ale použitá implementácia FEC kódov je napísaná v jazyku C++.

Bolo preto potrebné vhodne upraviť niektoré časti platformy UltraGrid kvôli kompatibilite s implementáciou LDPC kódov.

1.5 UltraGrid

Ide o software používaný na real-time prenos nekomprimovaného videa vo vysokom rozlíšení. Umožňuje uskutočňovanie videokonferencií pri zachovaní maximálnej možnej kvality obrazu. Implementácia využívaná na Masarykovej univerzite je úpravou pôvodnej implementácie UltraGrid-u od Colina Perkinsa[1]. Umožňuje nízkolatenčný prenos nekomprimovaného HD videa (1920 x 1080 @ 30i/25i) pri 8- alebo 10-bitovej farebnej hĺbke. Ide o multiplatformnú aplikáciu, ktorá je schopná pracovať pod operačným systémom Linux, ale aj pod systémom MacOS X. Takisto podporuje rôzne ďalšie technológie ako sú DXT komprimované dátové toky, pripadne podporu iHDTV[2].

1.6 LDPC (Low-density parity-check) kódy

Vlastná implementácia samoopravných kódov je distribuovaná pod GNU/LGPL licenciou[3], čo umožňuje ich využitie pre účely doprednej opravy chýb pre platformu UltraGrid. Príslušný kodek obsahuje rôzne optimalizácie a veľkou výhodou je jeho komplexnosť. Hoci je daná implementácia napísaná v jazyku C++, je veľmi dobre vyladená a to umožňuje na príslušne výkonnom stroji dosahovať potrebné dátové toky, aké sú potrebné pre prenos nekomprimovaného HD videa. Využitím samo opravných kódov sa nároky na priepustnosť siete síce zvýšia, avšak takisto sa zvýši aj kvalita prenášaného videa a menšiu náchylnosť na vznik rôznorodých fragmentov vo videu vzniknutých stratou jednotlivého paketu, prípadne viacerých paketov.

Využitá implementácia ponúka relatívne bohaté možnosti pri nastavovaní jednotlivých atribútov, ako sú napríklad aj typ kodeku[4], pomer dátových a opravných paketov a ďalšie. Bol zvolený kodek, ktorý podľa porovnania dosahoval najvyššie prenosové rýchlosti. Výpočtová náročnosť je vyššia na strane príjemcu, kde je potrebné zaistiť znovuzostavenie stratených paketov. Jednak treba rozhodnúť, ktorý paket chýba a prípadne dopočítať jeho hodnotu z informácie obsiahnutej v paketoch, ktoré obsahujú opravné dáta a sú posielané príjemcovi spolu s pôvodnou informáciou. Množstvo informácie, ktorú je možné nahradiť je okrem iného závislé najmä na množstve pridaných dát k pôvodným. Preto je treba nájsť vhodný kompromis medzi vymedzenou šírkou pásma a množstve strát, ktoré je možné opraviť. Avšak je treba uvažovať aj nad zaťažením celého systému z pohľadu zvýšenia latencie prenosu.

1. <http://csperkins.org/ultragrid/>
2. <http://www.washington.edu/ihdtv/>
3. <http://www.gnu.org/copyleft/lesser.html>
4. LDPC Staircase alebo LDPC Triangle.

Kapitola 2

Detekcia a oprava chýb

Detekcia vzniknutých chýb je schopnosť určiť prítomnosť nesprávne prijatej informácie počas prenosu. Naproti tomu oprava chýb je schopnosť zrekonštruovať pôvodné dáta bez chýb. Existujú 2 základné prístupy pri korekcii chýb:

- **Automatic Repeat Request (ARQ)** – odosielateľ pri zasielaní posiela pôvodné dáta a s nimi takisto aj určitý kód, ktorý umožní príjemcovi detekovať prítomnosť vzniku chýb. V mnohých prípadoch je žiadosť o retransmisiu nastavená implicitne – odosielateľ zasiela dáta automaticky, ak od príjemcu nedostane potvrdenie o bezchybnom prijatí dát do určitej doby.

  Existuje niekoľko prístupov pri detekcii chýb. Všetky takéto kódy zasielajú určitú redundantnú informáciu spolu s vlastnými dátami. Väčšina týchto kódov pracuje na princípe, že po zaslaní pevne daného počtu bitov dát, zašle pevne daný počet kontrolných bitov, ktoré sú odvodené z originálnych dát deterministickým algoritmom. Príjemca po prijatí dát používa rovnaký algoritmus ako odosielateľ, z prijatých dát spočíta kontrolnú informáciu a tú porovnáva s príslušnými dátami od odosielateľa. V prípade zhody prebehol prenos v poriadku a retransmisia nie je potrebná. Naopak rozdielny výsledok indikuje výskyt chyby počas prenosu.

- **Forward Error Correction (FEC)** – odosielateľ zakóduje dáta zvoleným algoritmom na opravu chýb a zasiela takto upravenú správu. Príjemca nezasiela odosielateľovi žiadne správy, pokúsi sa opraviť čo najviac chýb, ideálne všetky. Množstvo zrekonštruovaných dát závisí na použitom algoritme a množstve chýb, ktoré sa pri prenose komunikačným médiom vyskytnú.

2.1 Spôsoby detekcie chýb

- **Opakovanie bitov** – existuje množstvo variácií tejto schémy. Tento typ rozdeľuje dátový tok na skupiny pevne daného počtu bitov. Pri zasielaní každej jednotlivej skupiny bitov sa ale posiela niekoľko ďalších kópií tej istej skupiny bezprostredne za sebou. Príjemca dokáže ľahko detekovať jednotlivé chyby porovnávaním príslušných bitov z každej kópie. Problém však môže nastať, ak sa chyba vyskytne presne na tej istej pozícii (v každom, alebo aspoň vo väčšine prijatých..). V tom prípade príjemca nedokáže rozpoznať chybu a chybné dáta sú označené ako korektne prijaté.

- **Paritný bit** – princíp tohto opravného mechanizmu je založený na rozdelení posielaných dát do skupín rovnakej dĺžky, pričom každému bloku sa priradí na koniec 1 bit navyše. Tento paritný bit je nastavený podľa počtu jednotkových bitov v danom bloku dát. Existujú 2 prístupy podľa toho, aký nastavujú paritný bit: buď nastavia paritný bit na 1, vtedy ide o tzv. párnu paritu alebo opačne a vtedy sa jedná o nepárnu paritu. V prípade, že sa testované bloky určitým spôsobom prekrývajú, je možné tento mechanizmus využiť aj pri opravovaní izolovaných chýb (na podobnom princípe pracujú Hammingove kódy).

  Nevýhodou je, že tieto schémy dokážu opraviť len párny resp. nepárny počet chýb (podľa zvoleného postupu určovania paritného bitu). Teda ak je pri prenose poškodených párny(nepárny) počet bitov, je chybne prijatá správa nesprávne vyhodnotená ako korektne prijatá.

- **Kontrolný súčet** – je výsledkom nejakej vopred určenej operácie vykonanej nad vlastnými dátami. Príkladom takéhoto princípu je číselné vyjadrenie jednotlivých kódových slov zo správy určitej dĺžky. Výsledkom je súčet jednotlivých zložiek. Príjemca si po prijatí spočíta vlastný kontrolný súčet a porovnáva ho s prijatým. V prípade, že súčet nesúhlasí s tým od odosielateľa, je detekovaný vznik chyby.

- **Cyclic redundancy checks (CRC)** – ako vstup berie dáta ľubovoľnej dĺžky, pričom výstupom sú dáta pevnej dĺžky. Princíp je podobný ako v prípade kontrolných súčtov. Príjemca po prijatí dát spočíta výsledok pre prijaté dáta a ten porovnáva s prijatým výsledkom. V určitých prípadoch je možné chybnú informáciu aj opraviť. Tento typ detekcie chýb je často využívaný pri zisťovaní chýb vzniknutých napríklad prenosom paketov po sieti a podobne.

- **Kontrola založená na hammingovej vzdialenosti** – tento typ kontroly prevedie každé n-bitové kódové slovo do n+k+1 bitov, pričom k+1 musí byť minimálna hammingova vzdialenosť[1]. Toto umožňuje príjemcovi detekovať d-chýb a pri kóde s hammingovou vzdialenosťou 2.k+1 dokonca tieto chyby aj opraviť.

- **Hashovacia funkcia** – princípom funkcie je prevod vstupu ľubovoľnej dĺžky na výrazne kratší výstup, ktorý tvorí jednonačnú charakteristiku dát. Využíva sa pri kontrole integrity prijatých dát.

2.2 Forward Error Correction (FEC)

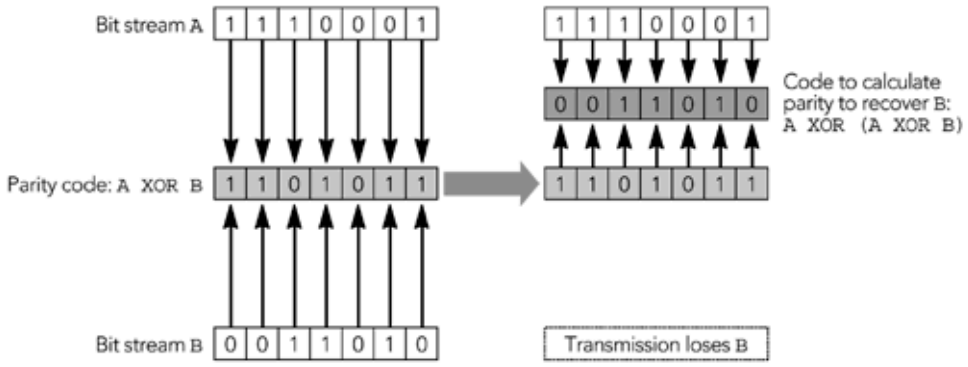
Dopredná oprava chýb je systém kontroly chýb a prenosu dát, kde odosielateľ pridáva k vlastný dátam určité množstvo redundantných informácii. Tie slúžia prijímateľovi na de-

1. Hammingova vzdialenosť dvoch binárnych reťazcov rovnakej dĺžky je počet príslušných pozícií, na ktorých sa navzájom líšia. Hammingova vzdialenosť kódu (množiny kódových slov) je minimálna vzdialenosť medzi ľubovoľnými dvoma slovami v kóde.

tekciu a prípadnú opravu chýb v dátach vzniknutých prenosom cez chybové médium. Výhodou týchto kódov je to, že prijímateľ sa už nemusí dožadovať žiadnych ďalších dát od odosielateľa pri výpadku, odpadá teda nutnosť retransmisie. Určitou nevýhodou je nutnosť využitia o trochu vyššej šírky pásma pri prenose, než je šírka potrebná pre prenos požadovaných dát. FEC kódy sa preto uplatňujú hlavne tam, kde opätovné zasielanie dát je relatívne nákladné pripadne nemožné. FEC kódy sa ale takisto využívajú v úložných médiách (CD a DVD médiá) kvôli zabráneniu straty uložených dát.

2.2.1 Princípy FEC

Jedným z najjednoduchších princípov samoopravných kódov je použitie paritného bitu. Operácia parita môže byť matematicky popísaná ako exkluzívna disjunkcia (XOR) bitového prúdu dát. Táto operácia je binárna, ale je ju možno použiť aj ako n-árnu operáciu, lebo je asociatívna: A XOR B XOR C = (A XOR B) XOR C = A XOR (B XOR C)

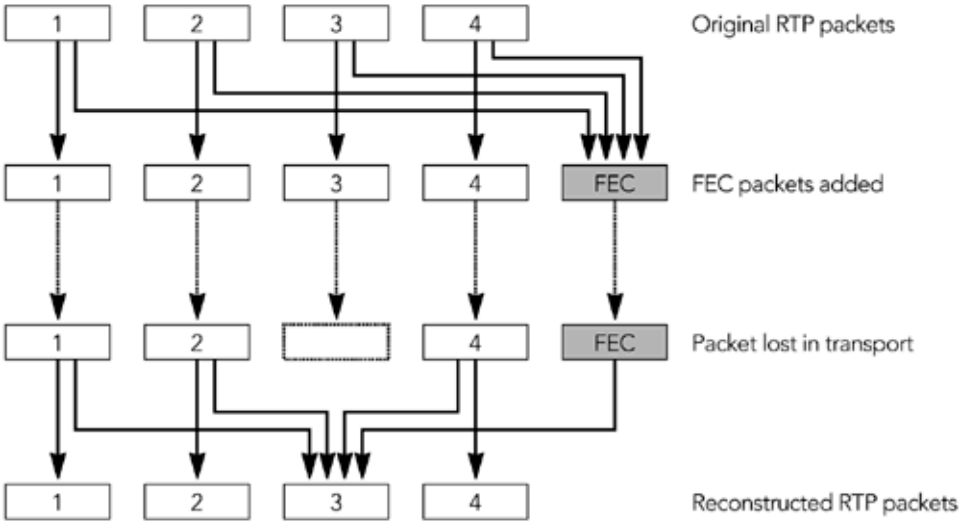


Obrázok 2.1: Použitie parity na obnovu stratených dát. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet*. Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 9.1, Use of Parity between Bit Streams to Recover Lost Data, s. 255.)

Typické použitie FEC paketu ukazuje nasledujúci obrázok. FEC paket obsahuje klasickú hlavičku, špeciálnu FEC hlavičku a samotné dáta, ktoré sú generované na základe dát, ktoré má daný FEC paket chrániť pred poškodením.

Ak jednotlivé zdrojové pakety nemajú rovnakú veľkosť, sú najskôr upravené tak, aby všetky boli rovnako dlhé ako najdlhší z nich. Použitím operácie XOR sa vytvorí FEC paket, ktorý sa zasiela spolu s pôvodnými dátami. Ak dôjde k výpadku niektorého z paketov, je možné informáciu v ňom obsiahnutú dopočítať zo zvyšných prijatých paketov. Množstvo informácie, ktorú je možné takto obnoviť je obmedzené použitím konkrétneho samoopravného kódu. Ak je takýto FEC kód schopný opraviť napríklad 5% stratu, ale dôjde k 10% strate paketov, tento kód nie je schopný obnoviť pôvodnú informáciu.

Nevýhoda pri posielaní paritných FEC paketov je tá, že zvyšujú celkové zaťaženie siete,



Obrázok 2.2: Obnova dát použitím paritného FEC paketu. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet.* Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 9.2, Repair Using Parity FEC, s. 256.)

a tak zasielanie neúmerne veľkého množstva týchto paketov môže viesť k zahlcovaniu siete, a tým spôsobeným stratám paketov.

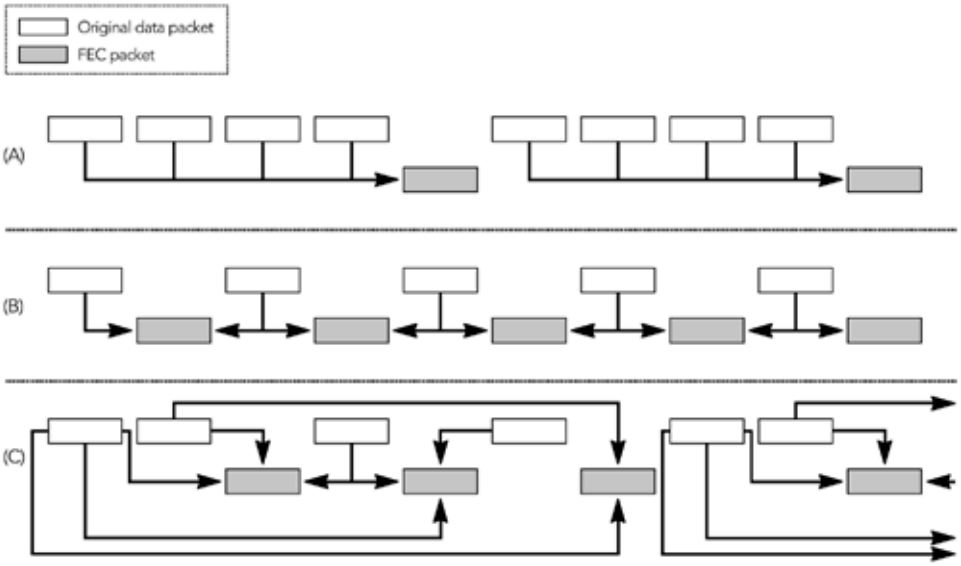
Existuje preto niekoľko typických možností prístupu:

- Najjednoduchším riešením je zasielanie FEC paketu po každom n-1 dátovom pakete (obrázok 2.3A), ktoré umožňuje obnovu dát, ak je nanajvýš jedna strata na každých n paketov. Tento prístup má malú réžiu, je ľahko odhadnuteľný a ľahko adaptovateľný, pretože percento FEC paketov je to isté, ako percento stratených paketov pri prenose.

  Ak je strata paketov počas prenosu rovnomerná, tak je tento prístup veľmi vhodný. Avšak ak dochádza k náhlym výpadkom väčších rozmerov, strata nie je opraviteľná. Ak sú tieto náhle výpadky relatívne časté, ako napríklad v sieti Internet, je možné počítať paritu nad väčším množstvom počtom čo vedie vo viac robustné riešenie. Tento prístup potom dobre funguje pre streaming, avšak má vysoké oneskorenie, ktoré nie je vhodné pre interaktívne aplikácie.

- Robustnejšou schémou, ale zároveň jednou z najviac plytvajúcich šírkou pásma, je posielanie FEC paketu medzi každou dvojicou dátových paketov (obrázok 2.3B). Tento prístup umožňuje príjemcovi opraviť každú jednu stratu paketu a množstvo dvojitých strát. Ako už bolo spomenuté, dochádza síce k veľkému plytvaniu šírky pásma, avšak oneskorenie je relatívne malé, čo je vhodné pre interaktívne aplikácie.

- Zložitejšie schémy dovoľujú viacero strát paketov nasledujúcich bezprostredne za sebou. Postup popísaný na obrázku 2.3C umožňuje obnovu až troch po sebe stratených paketov. Nutnosť spočítať FEC nad viacerými paketmi vytvára relatívne vysoké oneskorenie, preto tento postup nie je celkom vhodný pre interaktívne aplikácie, avšak je dobre použiteľný pri streamovaní.



Obrázok 2.3: Ukážka niekoľkých FEC schém. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet*. Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 9.4, Some Possible FEC Schemes, s. 260.)

2.2.2 Typy FEC

Samoopravné kódy sa rozdeľujú do dvoch hlavných kategórií podľa prístupu, akým spracúvajú dáta.

- **Konvolučné** - pracujú s dátovým tokom ľubovoľnej dĺžky, pričom rozlišujú jednotlivé bity alebo celé symboly. Každý m-bitový symbol [2] transformujú na n-bitový symbol, kde sa pomer m/n nazýva code rate (pričom n $\geq$ m). Konvolučné kódy môžu byť zmenené na blokové kódy.

  Využívajú najmä na zvýšenie kvality digitálneho vysielania, mobilnej komunikácie prípadne satelitných prenosov. Tie zvyčajne nemôžu tolerovať oneskorenie, ale tolerujú určitú poruchu v prenose.

2. String, paket.

- **Blokové** - pracujú s blokmi dát, kde každá informácia spracovávaná kodérom musí byť rozdelená na bloky určitej dĺžky[3]. Tieto kódy sú tiež označované ako (n,k)-kódy, kde n reprezentuje dĺžku slova po zakódovaní a k reprezentuje dĺžku jedného bloku vstupných dát, pričom sú obe tieto dĺžky pre daný kód konštantné. Týmto sa líšia napríklad od Huffmanovej metódy kódovania alebo od konvolučných kódov.

2.3 Reed-Solomon kódy

Tieto kódy patria do skupiny blokových kódov. To znamená, že vstupný blok konštantnej dĺžky je spracovaný do výstupného bloku takisto konštantnej dĺžky. Najbežnejšou implementáciou Reed-Solomon kódov je typ (255, 223), kde 223 vstupných symbolov (každý dĺžky 8 bitov) je zakódovaný do 255 výstupných symbolov. Tento typ je schopný opraviť až 16 symbolov v každom kódovom slove.

Tento druh samoopravných kódov je široko využívaný v mnohých aplikáciách, pričom najväčšie uplatnenie nachádza v úložných médiách (CD, DVD a Blu-Ray diskoch), v technológiách na prenos dát ako sú DSL a WiMAX, pri šírení digitálneho vysielania (štandardy DVB a ATSC) ale takisto aj v technológiách RAID6.

2.4 Turbo kódy

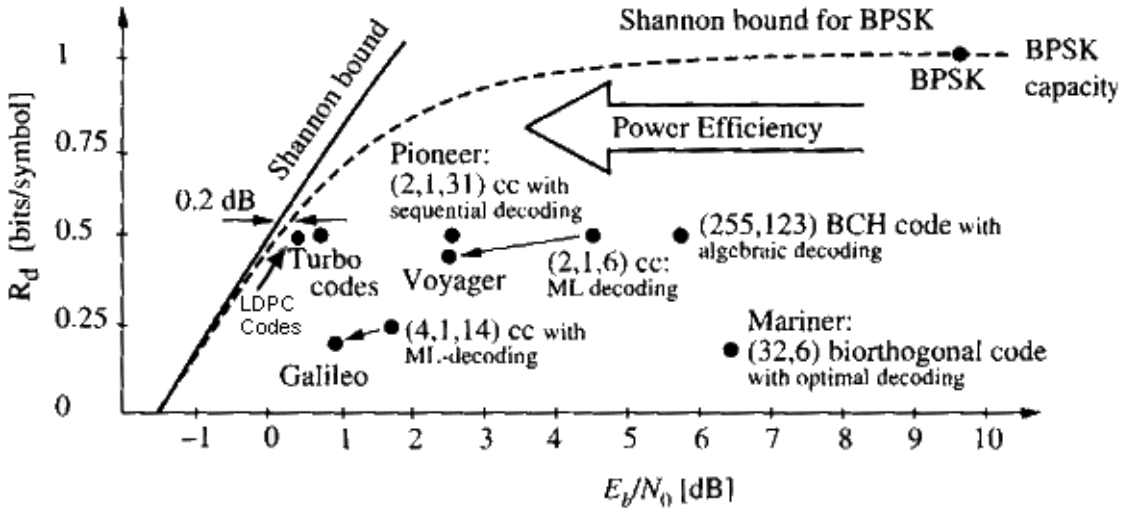
Ide o vysoko výkonnú skupinu samoopravných kódov vyvinutých v roku 1993. Uplatnenie našli najmä v satelitoch preskúmavajúcich okolitých vesmír a v iných aplikáciách, ktoré vyžadujú čo najväčší možný objem prenesených dát pri obmedzenej šírke pásma chybového komunikačného média.

2.5 Low-density parity-check (LDPC) large block Forward Error Correction codes

Tieto kódy využívajú princíp opravy chýb objavený v šesťdesiatych rokoch dvadsiateho storočia Robertom G. Gallagerom. V dobe objavenia však neboli vhodné na praktickú implementáciu, a to najmä kvôli ich komplexnosti, výkonovej náročnosti a objaveniu sa Reed-Solomon kódov. Tento princíp samoopravných kódov bol „znovuobjavený“ až v roku 1996.

LDPC kódy sú triedou takzvaných lineárnych blokových kódov. Názov pochádza z, pre nich typickej, parity-kontrolujúcej matice, kde sa nachádza iba malé množstvo jednotiek v porovnaní s množstvom núl. Medzi ich hlavné výhody patrí, že poskytujú výkon, ktorý je veľmi blízko kapacite média a lineárna časová náročnosť dekódovania. Takisto sú vhodné pre implementácie, ktoré využívajú paralelizmus.

3. Tá je daná použitým typom kódu



Obrázok 2.4: Porovnanie výkonnosti jednotlivých kódov. (zdroj: <http://cmrr-star.ucsd.edu/psiegel/pubs/07/ldpc_tutorial.ppt>)

2.6 Reprezentácia LDPC kódov

V podstate existujú 2 rozdielne možnosti reprezentácie týchto kódov, prvou je reprezentácia pomocou matíc, druhou je grafická reprezentácia.

2.6.1 Maticová reprezentácia

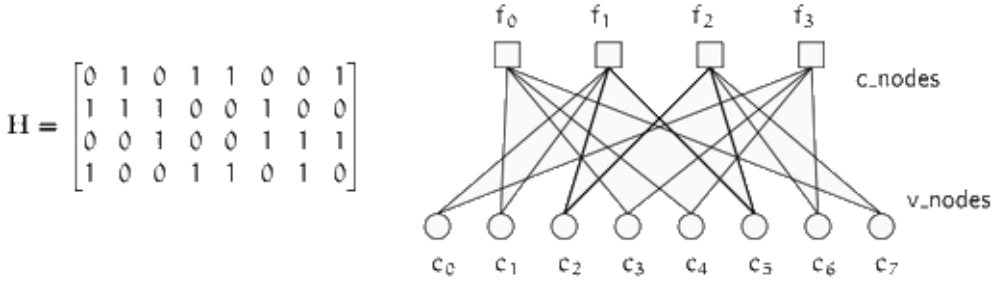
Nech je daná matica veľkosti $m \times n$. LDPC kódy kladú určité podmienky na tieto matice. Matice typu low-density sú len také, ktoré spĺňajú nasledujúcu vlastnosť: $w_c \ll n \& w_r \ll m$, pričom w_r je celkový počet jednotiek v každom riadku matice a w_c je celkový počet jednotiek v každom stĺpci. Pri budovaní dobrého kódu zároveň platí, že $w_c \geq 3$, preto sú tieto paritu-kontrolujúce matice obvyklé veľmi rozsiahle. Nižšie uvedená matica je príkladom, ktorý popisuje Hammingov kód typu (8,4), avšak nie je veľmi dobre použiteľná, lebo nie je riedkou (low-density matrix).

2.6.2 Grafická reprezentácia

Na tento účel sa využívajú Tannerove grafy, ktoré predstavujú veľmi dobrú grafickú reprezentáciu pre LDPC kódy. Nielenže umožňujú kompletnú reprezentáciu týchto kódov, ale takisto môžu byť využité pri popisovaní algoritmu.

Tannerove grafy patria do skupiny bipartitných grafov[4], teda takých, kde sú jednotlivé uzly rozdelené do dvoch osobitných množín a hrany spájajú len uzly navzájom rozdielnych

4. <http://en.wikipedia.org/wiki/Bipartite_graph>



Obrázok 2.5: Kód reprezentovaný maticou a príslušným Tannerovým grafom. (zdroj: <http://www.engr.uvic.ca/~masoudg/upload/ldpc-a%20brief%20tutorial.pdf>)

typov. Tieto skupiny uzlov sa nazývajú premenné uzly (v-nodes) a kontrolné uzly (c-nodes).

Na obrázku 2.5 je daný príklad takejto reprezentácie a zároveň uvedený Tannerov graf popisuje rovnaký kód ako matica nad ním. Obsahuje m kontrolných uzlov (počet paritných bitov) a n premenných uzlov (počet bitov v kódovom slove). Kontrolný uzol f_i je spojený s premenným uzlom c_j práve vtedy, ak element h_{ij} matice H je 1.

V Tannerových grafoch môžu vznikať krátke cykly[5], avšak prítomnosť týchto cyklov negatívne ovplyvňuje výkonnosť dekódovania, preto by sa v danom kóde nachádzať nemali.

2.7 Pravidelné a nepravidelné LDPC kódy

Pravidelné LDPC kódy splňujú podmienku, kde parameter w_c je konštantný pre každý stĺpec a zároveň $w_r = w_c \times (n/m)$ je takisto konštantný pre každý riadok. Príklad takejto matice je na obrázku 2.5, pričom platí, že $w_c = 2$ a $w_r = 4$. Pravidelnosť je taktiež možné vidieť na grafickej reprezentácii kódu, kde na každý v-uzol pripadá rovnaký počet príchodzích hrán, podobne aj pre c-uzly. Ak je daná riedka matica, ale počet jednotiek v jednotlivých riadkoch a stĺpcoch nie je konštantný, ide o nepravidelný LDPC kód.

2.8 Princíp LDPC kódov

LDPC kódy sú definované riedkou paritu-kontrolujúcou maticou H, ktorá je zvyčajne generovaná náhodne alebo generujúcou maticou G. Riedku maticu H môžeme pomocou Gaussovej eliminačnej metódy upraviť do tvaru $[P^T \mid I]$. Generujúcu maticu je možné vypočítať podľa vzťahu $G = [I \mid P]$. Kódovanie jednotlivých kódových slov c ďalej prebieha aplikovaním nasledujúceho vzťahu: $c = xG = [x \mid xP]$.

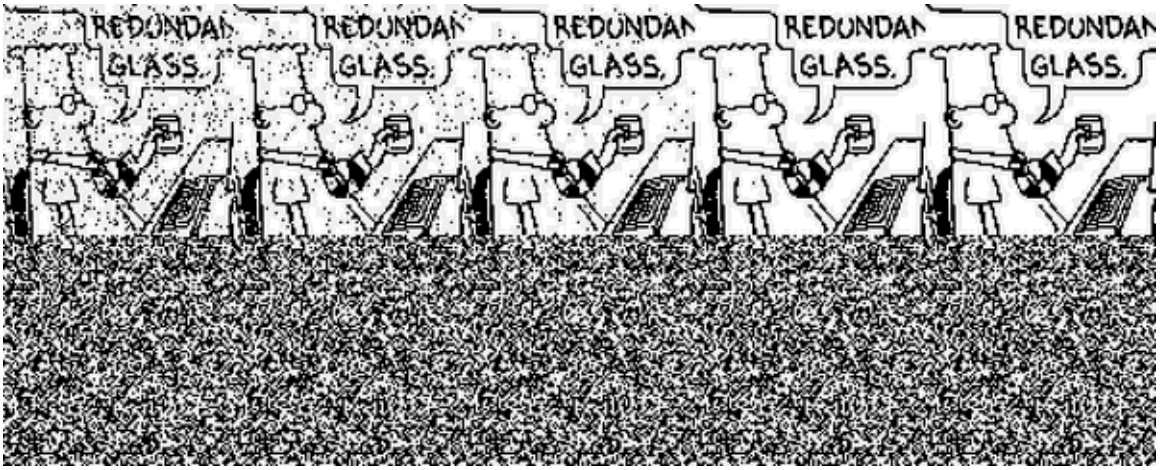
Vlastnosťou LDPC kódov je, že dokážu pracovať na hranici Shannonovho limitu[6], avšak iba za podmienky, že pracujú s veľkými blokmi dát. Použitie veľkých blokov implikuje ge-

5. Príkladom takéhoto krátkeho cyklu je hrubo vyznačená cesta $(c2 \rightarrow f1 \rightarrow c5 \rightarrow f2 \rightarrow c2)$.
6. Viď obrázok 2.4

nerovanie veľkej paritu-kontrolujúce matice, pričom náročnosť násobenia kódového slova s maticou závisí od počtu jednotiek v danej matici. Problémom teda je, že matica G je veľmi veľká a obecne nie je riedka, tak ani submatica P nie je obecne riedkou a to zvyšuje výpočtovú náročnosť. Pretože takýto prístup vedie k zložitosti $O(n^2)$ aj pri použití riedkych matíc, čo neúmerne zvyšuje výpočtovú náročnosť, volia sa iné metódy prístupu. Existujú rôzne iteratívne algoritmy, prípadne algoritmy založené na rôznych algebrických či geometrických metódach (kódovanie s využitím posúvania hodnôt v registroch a podobne), ktoré majú zložitosť $\sim O(n)$.

2.9 Dekódovanie

V súvislosti s LDPC kódmi bolo objavených viacero algoritmov, ktorými je možné dekódovať dáta. Obecný princíp dekódovania LDPC kódov spočíva na pricípe, že kódové slovo c je platné iba vtedy, ak platí vzťah $cH^T=0$. Avšak prenosové médium pridáva ku kódovému slovu istý chybový vektor e, ktorý zmení niektoré bity kódového slova. Dekódovanie je založené na lineárnej algebre, existujú však aj rôzne grafové algoritmy.



Obrázok 2.6: Demonštrácia princípu fungovania LDPC kódov na dátach s pridaným šumom (code rate = 1/2). Sekvencia reprezentuje približne každú tretiu iteráciu dekódovania. (zdroj: <http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/10000.075.seq.large.10.gif>)

2.9.1 Peeling decoder

Princíp algoritmu:

- Inicializácia: Zaslanie známych hodnôt v-uzlov pozdĺž hrán, zhromaždiť tieto hodnoty na kontrolných uzloch a uložiť paritu. Vymazať známe v-uzly a všetky ich hrany, ktoré sú s nimi spojené.

- Dekódovanie: Ak je možné, vybrať kontrolný uzol s iba jednou hranou, zaslať uloženú paritu a tým určiť hodnotu príslušného v-uzlu.
- Ukončenie: Ak je výsledný graf prázdny, kódové slovo bolo úspešne určené. V opačnom prípade ohlásiť neúspech dekódovania.

2.9.2 Message-Passing decoding

Pri tomto druhu algoritmu všetky variabilné aj kontrolné uzly zasielajú správy paralelne pozdĺž priľahlých hrán. Jednotlivé hodnoty bitov kódového slova sú obnovované taktiež. Algoritmus pokračuje kým nie sú všetky chýbajúce miesta vyplnené alebo po určenom počte iterácií.

Zasielanie správy od variabilného uzlu ku kontrolnému:

- Ak všetky príchodzie správy sú ?, poslať správu v=?
- Ak nejaká príchodzia správa u obsahuje 0 alebo 1, zaslať správu v=u. V prípade, že bit bol vymazávací, vyplniť s u.

Poznámka: Prijatá správa 0 alebo 1 musí byť správna, nemôže byť nekonzistentná.

Zasielanie správy od kontrolného uzla k variabilnému po hrane e:

- Ak je niektorá príchodzia správa ?, zaslať u=?
- Ak sú všetky príchodzie správy 0 alebo 1, zaslať výsledok funkcie XOR aplikovanú na prijaté správy, u=v1+v2+v3

2.9.3 Belief propagation

Tento algoritmus je istou podmnožinou predošlého algoritmu, je niekedy takisto nazývaný sum-product decoding algoritm, prvýkrát predstavený Gallagerom v roku 1962 v jeho dizertačnej práci. Princíp algoritmu je podobný ako v predošlom prípade, avšak v tomto prípade správy reprezentujú pravdepodobnostné hodnoty (beliefs) podľa logaritmického pravdepodobnostného pomeru.

Kapitola 3

Použitá implementácia LDPC FEC kódov

Kľúčovou vlastnosťou týchto kódov je, že pracujú s veľkými blokmi dát. Tieto zdrojové dáta môžu byť väčšie ako umožňujú spracovať napríklad Reed-Solomonove alebo Crauchyho kódy. Napríklad typický Reed-Solomonom kód dokáže spracovať blok dát s veľkosťou do 255 paketov, avšak LDPC kódy dokážu spracovať bloky dát o veľkosti aj niekoľko stoviek či tisícok paketov. Limitujúcim faktorom sa stáva množstvo dostupnej fyzickej pamäte, či už na strane odosielajúceho, alebo prijímajúceho. Pre predstavu, ak uvažujeme veľkosť jedného paketu na 1kB, tak tieto kódy dokážu spracovať dáta o veľkosti niekoľko desiatok až stoviek megabajtov.

Hoci tieto kódy sú určené najmä pre prácu s veľkými blokmi dát, s určitými obmedzeniami dokážu pracovať aj s malými blokmi. Tieto jednotlivé symboly sú zoskupované do väčších celkov, čo má veľmi pozitívny vplyv na výkonnosť týchto kódov aj nad takýmito blokmi.

Bola využitá posledná dostupná verzia[1], ktorá oproti starším verziám priniesla niekoľko vylepšení, medzi ktoré patrí mierne zvýšenie efektivity algoritmov, bola odstránená nutnosť používať symboly s dĺžkou deliteľnou 4 a takisto rôzne ďalšie optimalizácie.

3.1 Výkonnostné porovnanie Reed-Solomon a LDPC kódov

Použitá implementácia v sebe obsahuje aplikácie na testovanie výkonnosti jednotlivých kodekov. V oficiálnych materiáloch[2] je porovnanie LDPC kódov, pričom sa testovali 2 implementované kodeky, s Reed-Solomon kódmi. Testovacia zostava bola zakaždým rovnaká, šlo o PentiumIV@3GHz, objekt veľkosti 20 000 paketov, každý o veľkosti 1kB, pričom sa vytváralo 10 000 paritných paketov (code rate = 2/3). Namerané výsledky boli nasledujúce:

1. LDPC v2.1
2. <http://planete-bcast.inrialpes.fr/IMG/txt/LDPC_v2.1_FAQ.txt>

LDPC STAIRCASE	
Encoding (considering FEC packets only):	734.0 Mbps
Encoding (considering source+FEC packets):	2,205.0 Mbps
Decoding:	816.5 Mbps
LDPC TRIANGLE	
Encoding (considering FEC packets only):	443.8 Mbps
Encoding (considering source+FEC packets):	1,331.4 Mbps
Decoding:	434.9 Mbps
REED-SOLOMON, n=255 (vyššia schopnosť opravy pri výpadku)	
Encoding (considering FEC packets only):	21.5 Mbps
Encoding (considering source+FEC packets):	64.4 Mbps
Decoding:	52.3 Mbps
REED-SOLOMON, k=51 (vyššia rýchlosť)	
Encoding (considering FEC packets only):	86.0 Mbps
Encoding (considering source+FEC packets):	258.0 Mbps
Decoding:	242.0 Mbps

Z výsledkov je vidieť menšiu výpočtovú náročnosť LDPC kódov a z toho vyplývajúce aj teoreticky vyššie možné prenosové rýchlosti. LDPC kódy dokážu oproti Reed-Solomovým kódom spracovať naraz aj väčší zdrojový objem dát, pričom teoretické maximum tohto kodeku siaha podľa jeho nastavenia od 2TB do 128TB (dáta sa spracúvajú po jednotlivých blokoch osobitne).

3.2 Integrácia s programovými celkami

Implementácia LDPC kódov umožňuje sama zasielať dáta, ale takisto je schopná spolupracovať s inými aplikáciami, hoci použitý LDGM/LDPC kód má určité obmedzenia čo sa týka dĺžky zdrojového bloku dát, počet symbolov v bloku a podobne. LDPC kódy pracujú nad veľkými blokmi dát, avšak súbory presahujúce schopnosti kodeku musia byť rozdelené do viacerých blokov, čo však robí riešenie problému s prenosom a kódovaním/dekódovaním viac komplexným.

Odosielateľ aj príjemca sa musia takisto dohodnúť na dôležitých parametroch kódu, a takisto aj na zvolenom kodeku. Parametre môžu byť zvolené pevne, alebo je možné zasielať požadované informácie v hlavičke paketu. Medzi takéto údaje patrí napríklad poradové číslo symbolu, celkový počet symbolov, dĺžka symbolu a iné.

Ak aplikácia produkuje pakety rôznej dĺžky, je takisto možné použiť danú implementáciu LDPC kódov, avšak každý jednotlivý paket je potrebné doplniť do konštantnej dĺžky (napríklad na veľkosť najdlhšieho paketu). Takto upravené pakety je potom možné spracovať týmito kódmi, pričom je ale potrebné príjemcovi spolu s takto upravenými paketmi zasielať aj informáciu o dĺžke pôvodných dát.

3.3 Možné spôsoby použitia implementácie LDPC kódov

Je možné pevne určiť niektoré parametre kodeku, prípadne využiť ponúkané triedy na získanie daných parametrov a tým istým spôsobom optimalizovať výpočet[3]. Nasledujúci príklad demonštruje možnosti tohto kodeku:

1.) Použitie bez ponúkanej optimalizácie, objekt zložený zo 100 symbolov (rovnaký počet paketov), každý veľkosti 1024B a rovnaké množstvo príslušných opravných paketov:

```
$ ./perf_tool2 -k100 -r100 -ss1024 -ps1024
init_time=0.000245

data_symbols=100 fec_symbols=100 symbol_size=1024 nb_symbol_per_pkt=1
total_nb_pkts=200 pkt_size=1024 object_size=102400 left_degree=3

build_fec_time=0.000314
decoding_time=0.000695 decoding_steps=122
inefficiency_ratio=1.220
```

Výsledkom je nutnosť použitia 22% dodatočných opravných symbolov pre dekódovanie.

2.) Využitie optimalizácie na určenie počet symbolov v pakete, teda združovanie symbolov, testovaný bol objekt o veľkosti 102400B, rovnaké počty a veľkosti paketov ako v predošlom príklade:

```
$ ./perf_tool2 -os102400 -fec2.0 -ps1024
init_time=0.002352

data_symbols=1600 fec_symbols=1600 symbol_size=64 nb_symbol_per_pkt=16
total_nb_pkts=200 pkt_size=1024 object_size=102400 left_degree=3

build_fec_time=0.000711
decoding_time=0.002540 decoding_steps=110
inefficiency_ratio=1.100
```

V tomto prípade bolo potrebných iba 10% dodatočných paketov na dekódovanie. Bolo to dosiahnuté umelým zvýšením počtu symbolov (16 na paket), avšak výpočtová náročnosť sa naopak zvýšila (decoding_time=0.002540 sekúnd namiesto 0.000695) a takisto došlo ak zvýšeniu množstva pamäte, ktorá bola pri výpočtoch potrebná.

3. Napríklad počet symbolov pripadajúcich na jeden paket.

3.4 Predpokladané zvýšenie latencie

Vytváranie FEC paketov so sebou nutne prináša vyššie systémové nároky (vytváranie paketov pre doprednú opravu chýb a zvýšenie celkového počtu paketov na prenos), a tým aj isté zvýšenie latencie. Na odhadnutie času, ktorý je potrebný na toto spracovanie môže poslúžiť obsiahnutá aplikácia[4].

Pre veľkosť jedného paketu 8500B (jumbo frames) vychádza približný počet produkovaných paketov na 740, pri pomere 0,2 opravných paketov sa takto ďalej vytvára dodatočných 148 FEC paketov. Do uvažovaného modelu bola takisto zahrnutá aj náhodná strata paketov.

```
$ ./perf_tool2 -k740 -r148 -ps8500 -ss8500 -loss1
init_time=0.000479

data_symbols=740 fec_symbols=148 symbol_size=8500 nb_symbol_per_pkt=1
total_nb_pkts=888 pkt_size=8500 object_size=6290000 left_degree=3

build_fec_time=0.009827
decoding_time=0.018622 decoding_steps=776
inefficiency_ratio=1.049
```

Tento príklad je ukážkou na stroji, ktorý je využívaný ako vysielacia stanica UltraGridu a dokumentuje, že čas potrebný na vytvorenie FEC paketov (build_fec_time) a čas potrebný na ich dekódovanie (decoding_time) zvýši latenciu asi o 27-35ms. Do tohto času nie je započítaný čas potrebný na prenos FEC paketov ani prípadná réžia spojená s úpravou dát produkovaných Ultragridom a ich následnú úpravu na formu vhodnú pre LDPC kodek. Štandardne sa pakety v tomto nástroji posielajú v náhodnom poradí, avšak je možné nastaviť sekvenčné posielanie paketov[5]:

```
$ ./perf_tool2 -k740 -r148 -ps8500 -ss8500 -t1 -loss1
init_time=0.000492

data_symbols=740 fec_symbols=148 symbol_size=8500 nb_symbol_per_pkt=1
total_nb_pkts=888 pkt_size=8500 object_size=6290000 left_degree=3

build_fec_time=0.009827
decoding_time=0.017749 decoding_steps=752
inefficiency_ratio=1.016
```

Parametre prenosu sú rovnaké ako v predošlom prípade, čas potrebný na vytvorenie opravných paketov sa stále pohybuje okolo 10ms. Podľa počtu strát paketov sa čas potrebný

4. Základná testovacia aplikácia `perf_tool` alebo pokročilejšia `perf_tool2`.
5. Najskôr dáta a následne FEC pakety.

na dekódovanie pohybuje okolo 17-23ms, čo sú podobné výsledky ako v predošlom prípade, bolo teda pre jednoduchosť zvolené sekvenčné posielanie paketov.

3.5 Rozdiel medzi kodekmi LDPC-Staircase a LDPC-Triangle

Jednotlivé kodeky sa medzi sebou líšia spôsobom, akým je generovaná matica na kontrolu parity a z časti rozdielnym algoritmom pri kódovaní/dekódovaní. Vyššie uvedené príklady využívajú pre svoju činnosť kodek Staircase, ktorý sa mierne líši od kodeku Triangle.

```
$ ./perf_tool2 -k740 -r148 -ps8500 -ss8500 -loss1 -c2
init_time=0.000635

data_symbols=740 fec_symbols=148 symbol_size=8500 nb_symbol_per_pkt=1
total_nb_pkts=888 pkt_size=8500 object_size=6290000 left_degree=3

build_fec_time=0.014464
decoding_time=0.025461 decoding_steps=772
inefficiency_ratio=1.043
```

V tomto meraní boli zachované pôvodné hodnoty, avšak bol použitý kodek Triangle. Všetky namerané hodnoty boli vyššie oproti Staircase zhruba o 3-5ms na oboch stranách, čo zvyšuje celkovú latenciu o ďalších 6-10ms. Bol preto vybraný kodek Staircase, ktorý dosiahol lepšie výsledky v meraniach.

3.6 Spotreba pamäte a výkonnostné optimalizácie

Implementácia LDPC kódov potrebuje ukladať zdrojové a FEC pakety, ale aj maticu na kontrolu parity, dekóder aj kontrolné uzly. Pri spracovaní objektu pozostávajúceho z 20 000 paketov, každý o veľkosti 1kB, pričom sa vytvára ďalších 10 000 paritných paketov (code rate = 2/3, resp. FEC ratio = 1,5) a pri použití LDPC Staircase (LDPC v1.4) bolo potrebné nasledujúce množstvo pamäte:

Matica na kontrolu parity:	3MB (vďaka využitiu riedkych matíc)
Ďalšia pamäť pre kóder:	30MB (základné nastavenie)
	20MB (program kóduje a posiela FEC pakety za behu za behu, následne ich hneď uvoľňuje zo zásobníka)
Dodatočná pamäť pre dekóder:	25MB

Ak vlastná aplikácia nepotrebuje uchovávať všetky vyprodukované FEC pakety, je možné na strane kodéra používať jednotlivé opravné pakety akonáhle sú vyprodukované kodekom. Pri použití LDPC-Staircase kodeku je potrebné uchovávať iba aktuálne vyprodukovaný i-tý a bezprostredne predošlý i-1 FEC paket. Akonáhle je i-tý paket zaslaný, je možné

uvoľniť predošlý i-1, pretože pre kodek už viac nie je potrebný. Uchovaný paket bude použitý ďalej pre vyprodukovanie i+1 paketu, následne uvoľnený atď.

Na strane dekodéra je možné uvoľňovať prijaté FEC pakety akonáhle sú predané a spracované funkciou `LDPCFecSession::DecodingStepWithSymbol()`, pretože už nebudú kodekom nijako využívané ani spracúvané.

Pri zvolen9 kodeku LDPC-Staircase, je možné aktivovať optimalizáciu správy pamäte `OPTIMIZEFORMEMORY`[6]. Pri použití LDPC-Triangle by však došlo k niekoľkým výrazným výkonovým degradáciám. Využitím tejto možnosti sa zníži veľkosť spotrebovanej pamäte o 1/6.

3.7 Kľúčové vlastnosti LDPC kodeku

Využitá implementácia LDPC kódov má relatívne široké možnosti nastavení, čo značne zvyšuje teoretickú účinnosť a možnosti nasadenia, prípadne upravenie špecifickým nárokom. Výkonnosť daného kódu závisí najmä od nastavenia nasledujúcich faktorov:

- veľkosť bloku zdrojových dát
- pomer FEC paketov vzhľadom k pôvodným
- použitie kodeku LDPC-Staircase alebo LDPC-Triangle
- poradie zasielania jednotlivých dátových a opravných paketov
- stratovosť daného média
- algoritmus použitý na dekódovanie

Nie je vopred dané univerzálne nastavenie týchto kódov, ide o experimentálne nastavenie jednotlivých parametrov v snahe o čo najlepšie zefektívnenie algoritmu v konkrétnom prípade (väčší počet FEC paketov potrebných na opravu oproti zvýšenej výpočtovej náročnosti a podobne). Veľkosť bloku dát je daný veľkosťou jedného snímku nekomprimovaného HD videa, pomer FEC paketov k pôvodným dátam je vhodné zvoliť podľa strát paketov v sieti. V istých prípadoch môže výkonnosť kodeku ovplyvniť aj spôsob posielania paketov (sekvenčne vs náhodné poradie).

6. Nachádza sa v súbore `ldpc_matrix_sparse.h`

Kapitola 4

Implementácia LDCP v platforme UltraGrid

Súčasná implementácia vychádza z pôvodného konceptu navrhnutého Colinom Perkinsom, avšak táto je doplnená o možnosť o možnosť spracovania videa vo vysokom rozlíšení. Veľkosť rozlíšenia vychádza z HDTV štandardu, pričom 1080i je mód s 1920x1080 obrazovými bodmi a prekladanými riadkami. Takýto nekomprimovaný HD video stream so 60 prekladanými riadkami za sekundu , 10-bitovou farebnou hĺbkou a použitým 4:2:2 samplingom vyžaduje šírku pásma 1,485Gbps.

Do tejto platformy bola pridaná možnosť využívať doprednú opravu chýb, pričom bola použitá hotová implementácia LDPC kódov, ktoré ponúkajú vysoké teoretické prenosové rýchlosti a takisto sú schopné pracovať blízko Shannonovho limitu. Ďalšou výhodou oproti Reed-Solomonovým kódom sa javí možnosť pracovať nad väčšími blokmi dát, nižšia výpočtová náročnosť a tým vyššia teoretická prenosová rýchlosť. Dopredná oprava chýb bola implementovaná iba pri prenose nekomprimovaného videa, teda nie je možné použiť FEC pri súčasnom zapnutí napríklad DXT kompresie.

4.1 Zvolené parametre LDPC kodeku

Použitá implementácia ponúka na výber niekoľko rôznych kodekov, ktoré sa líšia v určitých parametroch, bol zvolený LDPC-Staircase, ktorý podľa výkonových charakteristík dosahoval najlepšie parametre, ale takisto je možné uplatniť optimalizáciu využívania pamäte. Bola zvolená veľkosť kódového symbolu na veľkosť paketu – jedna vyprodukovaná dátová časť paketu s príslušnými hlavičkami indikujúcimi rozloženie pripojených dát. Bolo vytváraných dodatočných 20% opravných paketov vzhľadom k získanému počtu zdrojových paketov. Takisto bol parameter PRNG[1] SEED zvolený na pevne danú hodnotu, ktorá bola zároveň rovnakou pre kóder aj pre dekóder. Nakoľko nebolo zaručené, že pakety produkované UltraGridom budú rovnakej veľkosti, bolo potrebné pred samotným procesom kódovania doplniť všetky pakety, ktoré mali menšiu veľkosť ako určenú[2] na túto jednotnú dĺžku doplnením núl na koniec. Kvôli zvýšenej výpočtovej a režijnej náročnosti bola zvolená veľkosť symbolu rovnajúca sa veľkosti paketu, bol teda ukladaný jeden symbol na paket za cenu prípadného menej efektívneho spracovania dát dekóderom.

1. Pseudo-Random Number Generator, musí spĺňovať vlastnosť, že pri počiatočnej hodnote 1 vracia po 10000 iteráciách hodnotu 1043618065.
2. Veľkosťou najväčšieho z nich.

4.2 Úprava potrebná na oboch prenosových stranách

Ultragrid s doprednou opravou chýb sa spúšta klasickým spôsobom[3], takto upravená implementácia umožňuje použiť FEC iba pri prenose nekomprimovaného videa, neumožňuje súčasné využitie FEC s DXT kompresiou prenášaných dát a taktiež neumožňuje odosielanie nekomprimovaného videa bez FEC. Boli upravené niektoré z procedúr, ktoré sa starali o spracovanie, prenos a následné dekódovanie dát. Funkcionalita spojená so spracovaním dát bola umiestnená do príslušných lokálnych procedúr starajúcich sa o samotné rozdeľovanie snímku do jednotlivých paketov na strane odosielateľa a reverzný proces na strane príjemcu.

4.3 Kódovanie dát

V pôvodnej platforme Ultragrid na prípravu paketov z jednotlivých snímkov slúži procedúra `tx_send()` v prípade prenosu nekomprimovaného videa vo vysokom rozlíšení, alebo procedúra `dxt_tx_send()` pri použití DXT kompresie dát[4]. Na účel doprednej opravy chýb bola upravená procedúra `tx_send()`, zachované boli časti kódu, ktoré sa starajú o samotné rozloženie snímku do jednotlivých paketov, avšak niektoré časti museli byť upravené kvôli implementácii samoopravného kódu a z toho plynúcej zmenenej réžii pri spracovaní a následnom posielaní dát. Pôvodná procedúra berie predložený snímok, ktorý rozdeľuje do paketov. Veľkosť týchto paketov je určená parametrom MTU[5], ktorý je možné definovať pri spúšťaní UltraGridu. Počas paketizácie snímku sa ukladajú aj metainformácie o rozložení obrazových dát. Po zozbieraní dostatku dát sa odošlú a procedúra pokračuje ďalšou iteráciou, pokiaľ takto nespracuje celý snímok.

Keďže FEC kodek vyžaduje pre svoju činnosť kódové slova rovnakej dĺžky a nebolo zaručené takéto vytváranie dát, bolo potrebné v každej iterácii, keď malo prebehnút odoslanie dát, poznačenie ich pôvodnej veľkosti. Tá pozostávala z veľkosti samotných video dát, ale aj z veľkosti príslušných vyprodukovaných hlavičiek, ktoré indikovali rozloženie dát v snímku. Ako kódové slovo slúžia obrazové dáta, ale taktiež aj príslušné metadáta. Problémom bolo vytvorenie celistvého kódového slova, pretože údaje boli v dvoch osobitných štruktúrach. Pre tieto účely bola využitá štruktúra `struct iovec`.

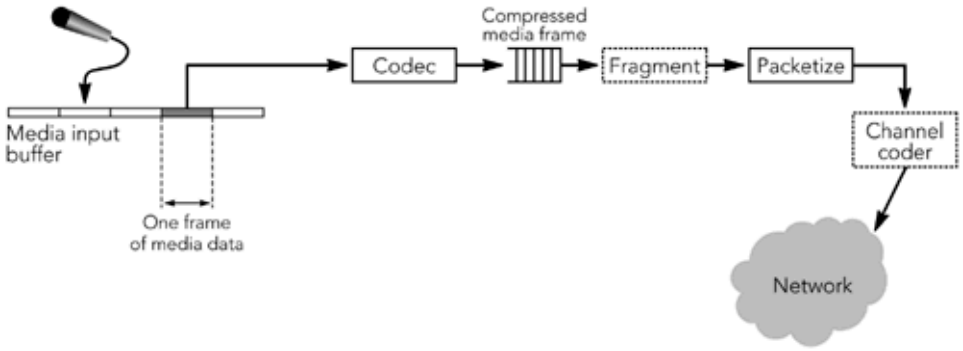
```
struct iovec {
      void  *iov_base;    /* Starting address */
      size_t iov_len;     /* Number of bytes to transfer */
   };
```

Do dvojrozmerného poľa, ktoré vychádzalo z tejto štruktúry, sa ukladali ukazovatele na jednotlivé časti dát ako aj ich veľkosť v bajtoch. Procedúra `read_iovec()` sa starala o nakopírovanie obsiahnutých dát do celistvej štruktúry patričnej dĺžky, ktorá následne reprezentovala jedno kódové slovo. Zasielanie paketov však muselo byť odložené, pretože použitie

3. <https://www.sitola.cz/igrid/index.php/UltraGrid#Running>
4. Nachádzajú v súbore `transmit.c`
5. Maximum Transmission Unit

FEC vyžadovalo spropagovanie ďalších dát dekóderu. Medzi tieto informácie patrí poradie symbolu v bloku, jeho veľkosť, pôvodná dĺžka dát a v neposlednom rade celkový počet paketov. Nakoľko sú všetky tieto údaje známe až po celkovej paketizácii snímku, muselo dôjsť k oneskoreniu zasielania paketov.

Následne muselo dôjsť k doplneniu dát na zhodnú veľkosť pridaním núl na koniec každého symbolu. Nakoľko bola veľkosť jednotlivých paketov známa, vybrala sa veľkosť najväčšieho kódového symbolu a do tejto veľkosti sa doplnili aj všetky ostatné pakety. Vzhľadom na fakt, že bola využitá posledná verzia FEC kodekov, nebolo potrebné, aby mali jednotlivé symboly špecifickú veľkosť [6]. Avšak to bolo umožnené aj zvoleným spôsobom kódovania, keďže nedochádzalo k umelému zväčšovaniu počtu symbolov na paket ich zoskupovaním, teda v tomto prípade delením získaného paketu na viacero častí. Tento kodek totiž umožňuje umelo zvýšiť počet zdrojových symbolov rozdelením každého paketu na niekoľko častí, avšak potom je potrebné zaistiť určitú veľkosť paketu, aby ho bolo možné takto rozdeliť bezozbytku.



Obrázok 4.1: Diagram spracovania dát. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet.* Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 1.2, Block Diagram of an RTP Sender, s. 12.)

Takto upravené zdrojové dáta už mohli byť predložené kodeku na spracovanie volaním procedúry `ldpc_encode()`, pričom sa vytvárali paritné symboly. Samotná inicializácia FEC kodeku a následné spracovanie dát si však vyžiadala vytvorenie nového súboru `ldpc_coder.cxx`, ktorý implementoval danú procedúru, kedže implementácia LDPC kódov je v jazyku C++ a platforma UltraGrid je napísaná v jazyku C. Táto procedúra berie ako argument pole s ukazovateľmi na kódové symboly, ich počet, ako aj počet paritných symbolov, ktoré sa majú vytvoriť a veľkosť kódového symbolu. Procedúra inicializuje príslušný kodek s dohodnutými parametrami, následne vytvorí paritné symboly, ukazovatele na nich uloží na koniec poľa, v ktorom sú uložené zdrojové symboly a uvoľní prostriedky.

Následne sa vytvorí FEC hlavička, ktorá obsahuje potrebné údaje pre dekódovanie. Tá-

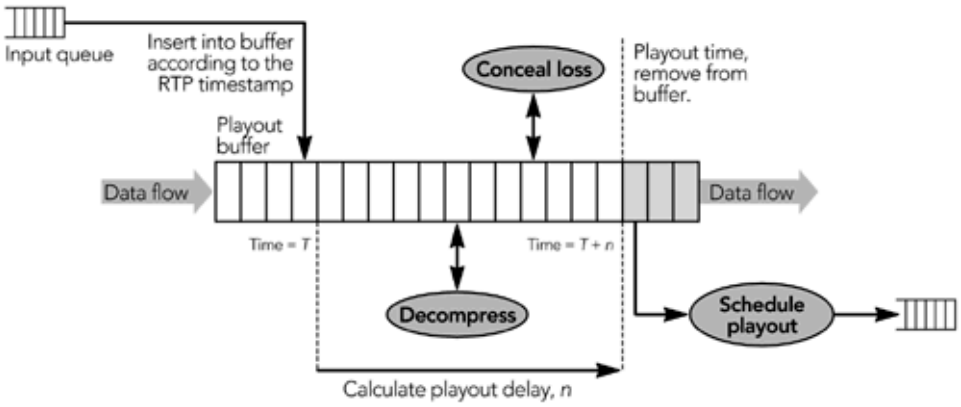
6. Predošlé verzie vyžadovali deliteľnosť 4

to hlavička sa spolu v kódovým slovom odosiela príjemcovi. Následne je už potrebné iba uvoľniť systémové prostriedky a procedúra končí.

4.4 Dekódovanie

Príjem a dekódovanie dát je náročnejší proces ako ich zakódovanie a odoslanie, pretože príjemca musí riešiť viacero problémov spojených s prenosom dát po sieti ako je napríklad výpadok niekoľkých paketov, resynchronizácia obrazu a zvuku a podobne.

Príjem paketov začína ich prijatím a umiestnením do zásobníka pre ich ďalšie spracovanie. Avšak pakety môžu byť prijímané s rôznymi časovými rozostupmi, keď môže na jednej strane dôjsť k prijatiu niekoľkých paketov súčasne, prípadne k oneskoreniu paketu a jeho prijatiu mimo požadovaného poradia. Po prijatí paketov sú skontrolované na výskyt chýb, je poznačený čas ich prijatia, umiestnené do zásobníka a usporiadané podľa RTP časovej značky.



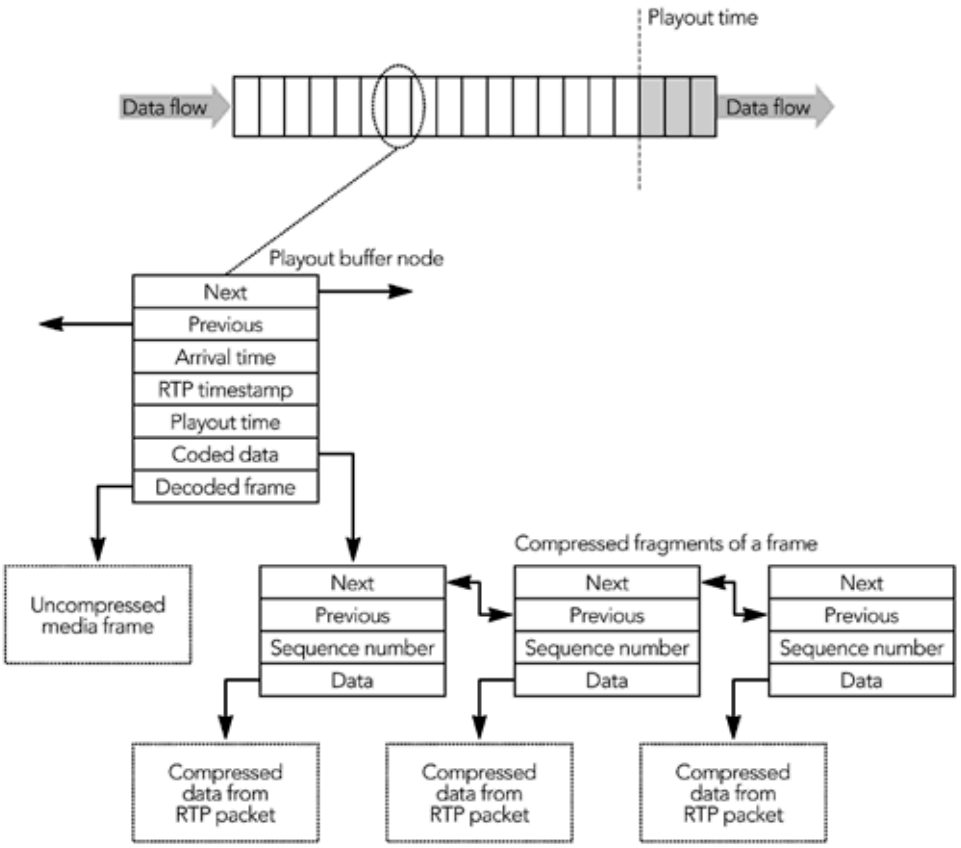
Obrázok 4.2: Schéma spracovania dát na strane príjemcu. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet.* Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 6.7, The Playout Buffer.)

Jednotlivé snímky sú držané držané v zásobníku po dobu potrebnú pre vykompenzovanie rôznych nežiadúcich elementov vzniknutých pri prenose sieťou. Držanie týchto dát vo vyrovnávacej pamäti dovoľuje takisto prijať fragmentované snímky a zlúčiť jednotlivé fragmenty dokopy. Takisto to dovoľuje vykonať rôzne opravy chýb na týchto dátach. Snímky sú potom dekomprimované, neopraviteľné chyby sú maskované pomocou rôznych na to určených techník a následne poskytnuté na zobrazovanie.

Buffer počas spracovania obsahuje už usporiadaný prepojený zoznam uzlov, pričom každý takýto uzol predstavuje rámec obrazových dát s príslušnou časovou značkou. Dátová štruktúra uzlu pozostáva s ukazovateľov na susedné uzly, čas príchodu, RTP[7] časovú

7. Real-time Transport Protocol je transportný protokol, ktorý zabezpečuje doručovanie audio a video dát.

značku, čas, kedy sa má daný snímok prehrať a zároveň komprimovaný aj nekomprimovaný obsah daného snímku.



Obrázok 4.3: Zobrazenie štruktúry dát, ktoré obsahujú obrazovú informáciu potrebnú pre dekomprimovanie jedného snímku. (zdroj: PERKINS, Colin. *RTP: Audio and Video for the Internet.* Boston: Addison-Wesley, 2003. 414 s. ISBN 0672322498. Figure 6.8, The Playout Buffer Data Structures.)

O spracovanie prijatých paketov sa stará procedúra `decode_frame()`, ktorá posiela prijímané dáta na ich ďalšie spracovanie príslušnej procedúre. Tu bola potrebná prvá úprava, pretože nestačí iba vytvoriť novú procedúru na spracovanie dát obsiahnutých v jednom pakete. Je najskôr potrebné tieto dáta[8] spracovať ako celok. Po prijatí prvého paketu sa zistia z FEC hlavičky obsiahnuté informácie, najmä o veľkosti paketov a ich celkovom množstve. Po prijatí každého paketu je potrebné určiť jeho poradové číslo v danom bloku dát[9], čo umožní jeho uloženie do bufferu na príslušnú pozíciu. Buffer po prijatí všetkých paketov obsahuje ukazovateľe na tieto dáta, pričom poradie každého symbolu je určené poradím

8. Jeden celý blok dát.
9. Tento blok je vymedzený jedným snímkom.

v tomto poli. Prvých k symbolov reprezentuje obrazovú informáciu, zvyšných n-k symbolov reprezentuje paritnú informáciu potrebnú pre prípadné znovuzostavenie chýbajúcich symbolov a hodnota `NULL` indikuje stratu daného paketu. Takto pripravené dáta je možné predložiť dekóderu. O implementáciu dekódera sa stará funkcia `ldpc_decode()` v súbore `ldpc_decoder.cxx` kvôli potrebe jej implementácie v jazyku C++. Procedúra sa pokúsi obnoviť stratené dáta, skončí v prípade úspešného dekódovania a prípadnej obnovy dát alebo po danom počte iterácií.

Takto pripravené dáta je ešte potrebné dekomprimovať. Bola upravená príslušná procedúra na spracovanie nekomprimovaných dát `copy_p2f()`, avšak musela byť mierne pozmenená. Úprava sa týkala vstupných parametrov, keďže sa už neposkytoval ukazovateľ na príslušný uzol, ale už iba predpripravené dáta (zbavené FEC hlavičiek) v podobe ukazovateľa `char*`. Ďalšie zmeny z toho vyplývajúce boli už len v správnom upravení typových parametrov vo vnútri funkcie.

Po zostavení snímku pripraveného na prehrávanie je možné pridelené prostriedky uvoľniť – všetky zdrojové a takisto aj opravné symboly. Tento postup je potrebný z dôvodu, lebo v zásobníku sa nachádzajú ukazatele na jednotlivé uzly, v ktorých sa nachádzajú síce prijaté dáta, avšak prirodzene neexistuje žiaden uzol, ktorý obsahuje obnovené dáta. Komprimované dáta už nad'alej nebudú potrebné a časom budú rovnako zahodené po zobrazení snímku. Je možné pamätať si poradové čísla obnovených symbolov, a uvoľňovať iba nové, vytvorené dekóderom.

4.5 Integrácia LDPC kodeku do platformy UltraGrid

Implementácia LDPC kódov obsahuje vlastný súbor `makefile`, ten bol však dekomponovaný a včlenený do už existujúceho `makefile` platformy UltraGrid. Pri tejto úprave boli zároveň odstránené niektoré časti kódu obsiahnutých v tomto súbore, nakoľko sa uvažuje iba s ich implementáciou na platforme Linux – skripty na určovanie platformy a z toho vyplývajúce špecifické nastavenia prekladača pre konkrétnu platformu boli vypustené a nahradené pevne určenými nastaveniami, ktoré by sa inak nastavili počas behu skriptu. Obsiahnutý kód bol takisto zjednodušený o vytváranie samostatnej knižnice použitých LDPC kódov, namiesto toho je kodek priamo včlenený do platformy UltraGrid. Úprava súboru sa týkala definovania pravidiel pre preklad - nastavenie príslušných flagov kompilátora a určení závislostí (podľa pôvodne obsiahnutého súboru v implementácii LDPC kódov).

4.6 Návrhy na zlepšenie

Hlavným obmedzením použitej implementácie LDPC kódov je istá nekompatibilita v podpore jednotlivých operačných systémov. Platforma UltraGrid pracuje pod operačným systémom Linux a MacOS X, avšak implementácia LDPC kódov nepodporuje MacOS X platformu, hoci počet podporovaných platforiem je relatívne široký. Ďalší vývoj tohto kodeku môže priniesť podporu aj pre tento operačný systém. Súčasná upravená platforma je teda schopná pracovať iba pod systémom Linux.

Ako už bolo spomínané vyššie, výkonnosť LDPC kódov, ako aj väčšiny ostatných, závisí od zvolených parametrov, ktoré sú závislé na konkrétnom prípade použitia. Preto je možné experimentovať s rôznymi nastaveniami kódov – typ kodeku, rozdeľovanie jednotlivých paketov na viacero kódových slov, pomer zdrojových a opravných symbolov – a následného zisťovania výkonových charakteristík LDPC kódov s takto upravenými parametrami.

S vyššie uvedeným úzko súvisí aj podpora nastavovať rôzne parametre cez príkazový riadok pri inicializovaní UltraGridu, pretože momentálne je potrebné znovu a znovu rekompilovať celú platformu UltraGrid, nakoľko sú niektoré hodnoty pevne určené. Takisto je možné viac integrovať FEC do platformy Ultragrid, nakoľko táto implementácia prepisuje jednotlivé procedúry a teda je možné použiť iba prenos nekomprimovaného videa s FEC.

Možnosťou na zvýšenie výkonu v budúcnosti by mohlo byť efektívnejšie spracúvanie dát – spraviť jednu iteráciu dekódovania už pri samotnom prijatí dát, čo by sa mohlo mierne odraziť na menšom oneskorení spôsobené spracúvaním, avšak za cenu väčšej réžie.

Výkon samotnej implementácie LDPC kódov sa zdá byť už relatívne optimalizovaný, v tomto smere nepredpokladám výrazné navýšenie. Zvyšovanie výkonu je možné zapínaním rôznych optimalizácií, ktoré sú však často použiteľné len pre istý vybraný kodek a pri použití iného, dochádza naopak k výkonnostnej degradácii.

Isté navýšenie výkonu by sa dalo dosiahnuť implementovaním multithreadingu pri spracovaní a súčasnom posielaní dát, kedy by sa jedno vlákno staralo o vytváranie opravných dát a ďalšie vlákno by v tom čase mohlo zdrojové dáta, ktoré sa počas kódovania nemenia, odovzdávať príslušným procedúram starajúcim sa o prenos paketov. Avšak vyžadovalo by si to kontrolu, aby i-tý zdrojový symbol nebol odoslaný a následne uvoľnený ešte pred tým, ako by bol spracovaný FEC kodekom.

Ďalším možným vylepšením by mohlo byť implementovanie podobnej utility ako obsahujú použité LDPC kódy na meranie výkonových charakteristík priamo do platformy UltraGrid a tým meranie výkonu s nastavenými parametrami kódu, čo ale úzko súvisí s podporou možnosti nastavovania príslušných parametrov kódu cez príkazový riadok.

Kapitola 5

Podobné práce

Existuje viacero aplikácii na prenos audia a videa, ktoré využívajú doprednú opravu chýb na zlepšenie kvality prenosu. Takisto platforma UltraGrid nie je jedinou aplikáciou umožňujúcou prenos videa vo vysokej kvalite. Tieto platformy obvykle obsahujú podporu pre FEC, keď sa snažia čo najviac chýb opraviť, aby bolo zakrývanie neopraviteľných výpadkov o to jednoduchšie.

5.1 Aplikácia RAT v MBone Tools

Robust Audio Tool (RAT) je open-source aplikácia, ktorú je možné využívať na účely audio konferencií alebo na streamovanie audia. Tento prenos je možný medzi dvoma účastníkmi priamo, prípadne medzi viacerými užívateľmi v multicastovej skupine. RAT je založený na IETF štandardoch, používa RTP nad UDP/IP protokolom na šírenie dát. Umožňuje nastavenie rôznych dátových tokov a nastavenie kvality kodekov, obsahuje algoritmy na odstraňovanie a maskovanie vzniknutých chýb počas prenosu pri strate paketov.

RAT implementuje 2 schémy opravy chýb po výpadku: zasielanie redundantných dát a ich prekladanie medzi paketmi.

Zasielanie redundantnej informácie (viac komprimovaných dát) spoločne s nasledujúcim vysielaným paketom. V prípade straty pôvodného paketu sa použije redundantná kópia, hoci kvôli zvýšenej kompresii trpí zvuková kvalita. Je však lepšie prehrať aj takto menej kvalitné dáta, akoby malo dôjsť k úplnému výpadku v prehrávaní zvukovej stopy. Je možné nastaviť množstvo redundantnej informácie na paket podľa požadovanej kvality a dostupnej šírky pásma.

Alternatívou je zasielanie prekladaných dát, keď pôvodné audio pakety sú rozdelené na menšie časti, tieto fragmenty sú následne rozložené do viacerých paketov. Každý takýto novovytvorený paket potom obsahuje časť audio dát z viacerých pôvodných paketov. Pri strate jedného paketu sa síce stratí malá časť informácie z viacerých paketov, avšak je možné zrekonštruovať väčšinu dát každého z nich a maskovanie výpadkov je jednoduchšie. Nevýhodou toho prístupu je zvýšenie latencie prenosu, kvôli nutnosti spracúvať pakety po skupinách, avšak nedochádza k zvýšenému zaťaženiu siete.

Oprava dát na strane príjemcu je závislá od spôsobu zasielania popísaných vyššie. Ako už bolo spomenuté, prekladanie dát má za úlohu čo najviac eliminovať vznik rozsiahlejších chýb, ale po tejto oprave ostáva ešte niekoľko malých izolovaných chýbajúcich častí z pôvodných dát. Tieto chyby nie je možné opraviť, preto musia byť použité rôzne techniky na

ich zakrytie. Medzi najjednoduchšiu patrí vloženie ticha na miesto výpadku, čo je účinné iba pri veľmi krátkych výpadkoch a malej veľkosti paketov. Ďalšou možnosťou je vloženie kópie dát, ktoré dorazili tesne pred výpadkom. Ide o istý kompromis medzi slabou efektivitou vkladania ticha a vysokou komplexnosťou vkladania určitých vzorkov dát. Posledný menovaný algoritmus pracuje na princípe vyplnenia straty vhodným signálom, ktorý je závislý na okolitých dátach a snaží sa čo najvernejšie zakryť výpadok.[17]

5.2 Demonštrácia prenosu 4K videa na konferencii iGrid 2005

Bolo uvedených viacero technológií, medzi ktoré patria JPEG 2000 Flexcast, Soundscape a schémy synchronizácie audia a videa prenášaných IP sieťami. Nekomprimovaný 4K snímok veľký zhruba 8 megapixelov vyžaduje šírku pásma väčšiu než 6Gbps. Kvôli takto vysokej dátovej náročnosti bol vytvorený real-time JPEG 2000 kodek, ktorý kompresiou snímkov znižuje bit rate na 200-450 Mbps.

Obe prenosové strany implementujú doprednú opravu chýb. Na strane servera sú komprimované dáta jednotlivých snímkov rozdelené do rovnako veľkých paketov špecifikovanej dĺžky. Blok dát je tvorený daným počtom paketov, posledný paket je doplnený tak, aby naplnil veľkosť tohto bloku. Pre každý blok je spočítaná horizontálna parita a z nej vytvorený paritný paket. Server následne pridá hlavičky každému paketu a následne ich odosiela v prekladanom poradí. JPEG 2000 dekóder prijíma tieto pakety a ukladá ich v adekvátne veľkom zásobníku. Pred samotným spracovaním dát dekóder skontroluje príchod jednotlivých paketov a v prípade straty sa pokúsi o obnovu týchto dát použitím paritného paketu. Pri strate jediného paketu je dekóder schopný obnoviť celú informáciu obsiahnutú v danom bloku dát. V opačnom prípade je potrebné použiť rôzne techniky[1] na zakrytie vzniknutých prenosových strát.

Počas demonštrácie tohto kodeku dochádzalo k stratám paketov, avšak pomocou FEC bolo možné väčšinu z nich opraviť. Pridaním 14% paritných FEC dát k jednému 4K komprimovanému streamu (250-400Mbps) bolo zistených 243 obnovených a 18 stratených snímkov z celkového počtu 35 777. Použitím FEC bol zlepšený pomer stratených snímkov z 0,73% na 0,05%. Niektoré snímky nemohli byť obnovené kvôli zvýšenému výskytu nárazových výpadkov jednotlivých paketov.[1]

5.3 iHDTV™

Ide o software umožňujúci zachytávať a prenášať HD video v rôznorodých formátoch prostredníctvom siete Internet. Na vývoji sa podieľa ResearchChannel[2] spoločne s University of Washington[3], táto technológia bola prvýkrát predstavená v roku 1999. Tento software je

1. Skopírovanie informácie z predošlého snímku. Takisto prekladanie poradia paketov znižuje pravdepodobnosť výskytu násobných strát.
2. <http://www.researchchannel.org/prog/>
3. <http://www.washington.edu/>

schopný pracovať s rýchlosťami do 1,5Gbps, čo je dostatočné pre prenos nekomprimovaného 1080i videa vo vysokom rozlíšení pri zachovaní čo najmenšieho oneskorenia.[21]

5.4 i-Visto

i-Visto je technológia vyvinutá laboratóriami NTT[4][5] umožňujúca prenos videa vo vysokej kvalite prostredníctvom IP sietí. Boli vyvinuté mechanizmy na prenos nízkolatenčného videa vo vysokom rozlíšení, architektúra vysokorýchlostných serverov umožňujúcich nahrávanie a prehrávanie vysokokvalitného videa, akým je DVCproHD, nekomprimovaného HD videa ale aj nekomprimovaného 4K videa.[23]

4. <http://www.ntt.co.jp/islab/e/org/ns.html>
5. <http://www.onlab.ntt.co.jp/en/index.html>

Kapitola 6

Záver

Dopredná oprava chýb je veľmi vhodným riešením pri odstraňovaní chýb prenosu a to najmä v prípade, že retransmisia nie je možná. Typickým príkladom sú videokonferencie, ktoré neumožňujú opätovné zasielanie dát z dvoch dôvodov. Prvým je použitie protokolu UDP, ktorý sa nijako nestará o potvrdzovanie prijatých dát, preto ani nie je možná detekcia výpadku na strane odosielajúceho. Druhým dôvodom je vyžadovanie real-time prenosu a čo najnižšej latencie, ktorá by sa pri retransmisii neúmerne zväčšila. Preto je výhodné použiť FEC za cenu nutnosti použiť väčšiu šírku pásma.

Možností implementácií FEC kódov je viacero, jedným z najjednoduchších by mohlo byť použitie paritných bitov získaných operáciou XOR nad jednotlivými zdrojovými dátami a z takto získanej paritnej informácie vytvoriť paket, ktorý by bol odoslaný s ostatnými dátami. Ďalšou možnosťou je použiť rôzne známe blokové kódové algoritmy ako sú napríklad Reed-Solomon kódy, ktoré nachádzajú široké uplatnenie vo výpočtovej technike. Hlavnou nevýhodou týchto kódov je fakt, že by nedokázali spracovať celý snímok v jednom bloku. Pri použití jumbo frames a zasielaní nekomprimovaného HD videa vzniká okolo 700-800 dátových paketov. Keďže Reed-Solomon kódy nie sú schopné spracovať takýto počet kódových symbolov, bolo by buď potrebné upraviť spôsob, akým sú dáta produkované, čo nie je najvhodnejšie riešenie, pretože by to zvýšilo réžiu, alebo rozdeliť každý snímok na viacero blokov a tie spracovať akoby samostatne. Druhý spôsob obnáša isté problémy pri dekódovaní, keď je najskôr potrebné určiť, do ktorého bloku daný paket patrí a spracovať všetky takéto bloky prijatých dát.

Odpoveď pri riešení týchto problémov, by však mohli poskytnúť iné kódy, ktoré sa dostávajú do popredia. Ide o LDPC kódy, ktoré ponúkajú veľmi dobré vlastnosti a ich už hotová implementácia je voľne použiteľná[1]. Hlavnými výhodami týchto kódov je, že dokážu pracovať blízko hranice Shannonovho limitu[2], ale taktiež odpadá nutnosť využiť viacero blokov pri spracovaní jediného snímku, keďže je možné ich použiť aj nad takto veľkým blokom. Použitá implementácia je multiplatformná, avšak neobsahuje podporu pre MacOS X, preto zatiaľ nie je možné využívať UltraGrid s FEC na tejto platforme. Výhodou je naopak možnosť prispôsobiť si všetky parametre podľa danej situácie, prípadne výber viacerých implementovaných prístupov kódovania a dekódovania[3].

Výsledkom práce je pokus o implementovanie LDPC kódov do platformy UltraGrid,

1. Šírené pod GNU Lesser General Public License.
2. Pri použití veľkého bloku dát.
3. Kodeky Staircase a Triangle.

avšak kvôli relatívnej komplexnosti tejto aplikácie ide o nahradenie, prípadne rozšírenie obsiahnutej funkcionality inou. Preto nie je možné v súčasnej podobe používať všetky typy prenosu[4], je možné použit iba jeden z nich a to práve prenos nekomprimovaného videa s podporou FEC.

4. Nekomprimované video bez použitia FEC, DXT kompresia dát a pod.

Literatúra

[1] SHIMIZUA, T. a SHIRAIA, D. a TAKAHASHIA, H. a MUROOKAA, T. a OBANAA, K. a TONOMURAA, Y. a INOUEA, T. a YAMAGUCHIA, T. a FUJIIA, T. a OHTAB, N. a ONOB, S. a AOYAMAC, T. a HERRD, L. a VAN OSDOLD, N. a WANGE, X. a BROWNE, M. a DEFANTIE, T. a FELDF, R. a BALSERF, J. a MORRISF, S. a HENTHORNG, T. a DAWEG, G. a OTTOG, P.: *ScienceDirect - Future Generation Computer Systems : International real-time streaming of 4K digital cinema*, 30 May 2006 [cit. 30. decembra 2009], URL: <http://www.sciencedirect.com/science/article/B6V06-4K2T517-2/2/33607c77705f23327605331d0963d118>. 5.2

[2] *Coding Concepts and Block Coding*, 2001 [cit. 15. novembra 2009], URL: <http://www.complextoreal.com/chapters/block.pdf>.

[3] *Error Correctionand Detection*, Apr 29, 2009 [cit. 21. decembra 2009], URL: <http://www.cs.nmsu.edu/~pfeiffer/classes/573/notes/ecc.html>.

[4] HOLUB, P.: *Nekomprimované HD video přes IP @ 1,5 Gbps*, 7. decembra 2005 [cit. 4. decembra 2009], URL: <http://www.cesnet.cz/akce/20051207/pr/holub.pdf>.

[5] SUN, J.: *An Introduction to Low Density Parity Check (LDPC) Codes*, June 3, 2003 [cit. 11. decembra 2009], URL: <http://www.csee.wvu.edu/wcrl/public/slideldpc.pdf>.

[6] *http://complextoreal.com/chapters/convo.pdf*, July, 1999 [cit. 15. novembra 2009], URL: <http://complextoreal.com/chapters/convo.pdf>.

[7] ROCA, V. a NEUMANN, C.: *Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec*, June 2004 [cit. 12. decembra 2009], URL: <http://planete.inrialpes.fr/people/roca/doc/rr04_fec_comparison.pdf>.

[8] JOHNSON, S. a WELLER, S.: *Low-density parity-check codes: Design and decoding*, June 20, 2002 [cit. 18. novembra 2009], URL: <http://sigpromu.org/reports/EE02041.pdf>.

[9] ROCA, V. a KHALLOUF, Z. a LABOURE, J.: *Design and Evaluation of a Low Density Generator Matrix (LDGM) Large Block FEC Codec*, June 20, 2002 [cit. 28. októbra 2009], URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.7863&rep=rep1&type=pdf>.

[10] ROCA, V. a NEUMANN, C. a FURODET, D.: *Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes*, June 2008 [cit. 12. decembra 2009], URL: <http://planete.inrialpes.fr/people/roca/doc/rfc5170.txt>.

[11] ROCA, V.: *LDPC Codec Frequently Asked Questions (FAQ)*, 6. septembra 2006 [cit. 4. decembra 2009], URL: <http://planete-bcast.inrialpes.fr/IMG/txt/LDPC_v2.1_FAQ.txt>.

[12] RYAN, W.: *An Introduction to LDPC codes*, August 19, 2003 [cit. 14. decembra 2009], URL: <http://www.ece.arizona.edu/~ryan/New%20Folder/ryan-crc-ldpc-chap.pdf>.

[13] SHOKROLLAHI, A.: *LDPC Codes: An Introduction*, April 2, 2003 [cit. 4. decembra 2009], URL: <http://ipm.ac.ir/swc/2002/amin/Amin2.pdf>.

[14] SIEGEL, P.: *An Introduction to Low-Density Parity-Check Codes*, May 31, 2007 [cit. 6. novembra 2009], URL: <http://cmrr-star.ucsd.edu/psiegel/pubs/07/ldpc_tutorial.ppt>.

[15] LEINER, B.: *LDPC Codes - a brief Tutorial*, April 8, 2005 [cit. 28. novembra 2009], URL: <http://www.engr.uvic.ca/~masoudg/upload/ldpc-a%20brief%20tutorial.pdf>.

[16] PERKINS, C.: *RTP: Audio and Video for the Internet*, Boston: Addison-Wesley, 2003, ISBN: 0672322498, s. 444.

[17] *Robust Audio Tool*, [cit. 30. decembra 2009], URL: <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/features.html>. 5.1

[18] *RatWiki - Media Tools*, 08/14/09 [cit. 30. decembra 2009], URL: <http://mediatools.cs.ucl.ac.uk/nets/mmedia/wiki/RatWiki>.

[19] *Self-correcting codes conquer noise Part 2: Reed-Solomon codecs*, March 15, 2001 [cit. 21. októbra 2009], URL: <http://www.edn.com/contents/images/315013.pdf>.

[20] HOLUB, P. a MATYSKA, L. a LIŠKA, M. a HEJTMÁNEK, L. a DENEMARK, J. a REBOK, T. a HUTANU, A. a PARUCHURI, R. a RADIL, J. a HLADKÁ, E.: *High-definition multimedia for multiparty low-latency interactive communication*, Amsterdam, The Netherlands, Elsevier Science (NLD), 2006, ISSN: 0167-739X, s. 856-861.

[21] *iHDTV^{TM}*, 2007 [cit. 30. decembra 2009], URL: <http://www.washington.edu/ihdtv/>. 5.3

[22] *ResearchChannel - Technology Projects Listed by Category*, 2009 [cit. 30. decembra 2009], URL: <http://www.researchchannel.org/tech/ihdtv.asp>.

[23] *i-Visto*, 2009 [cit. 30. decembra 2009], URL: <http://www.i-visto.com/>. 5.4

Dodatok A

Odkazy na použitý software:

Pri tejto práci bol využitý nasledujúci software:

UltraGrid <https://www.sitola.cz/igrid/index.php/UltraGrid>
Platforma UltraGrid

LDPC FEC Codes <http://planete-bcast.inrialpes.fr/article.php3?id_article=16>
Knižnica implementujúca LDPC-Stiarcase a LDPC-Triangle FEC kódy

Dodatok B

Obsah priloženého CD

Súčasťou tejto práce je aj CD. Jeho obsah:

- Zdrojové kódy tejto práce vo formáte XML(DocBook) a práca samotná vo formáte PDF.
- Zdrojové kódy upravenej platformy ultraGrid.