Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling met

Titel: Beschermen van Multimedia Netwerktrafiek tegen Transmissiefouten
Richting: 2de masterjaar in de informatica - multimedia Jaar: 2009

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

CYPERS, Kevin

Datum: 14.12.2009

Beschermen van Multimedia Netwerktrafiek tegen Transmissiefouten

Kevin Cypers

promotor :
Prof. dr. Wim LAMOTTE

Universiteit Maastricht

universiteit hasselt

Eindverhandeling voorgedragen tot het bekomen van de graad master in de informatica multimedia

Samenvatting

Door de sterke groei van het Internet is er steeds meer interesse in real-time multimedia applicaties. Deze applicaties hebben de eigenschap dat ze tijdsgebonden zijn. Videoframes moeten tijdig aankomen bij de ontvanger om een vloeiende beeldsequentie te kunnen afspelen. In interactieve applicaties moeten vertragingen die worden geïntroduceerd door het netwerk korter blijven dan wat menselijke perceptie kan waarnemen. Indien aan deze beperkingen geen gevolg wordt gegeven, zal de gebruiker een lage Quality-of-Experience (QoE) ervaren.

Netwerken, en in het bijzonder het Internet, kunnen data verliezen of beschadigen. Standaard Internetprotocollen zoals TCP vangen deze transmissiefouten op door beschadigde of verloren pakketten opnieuw op te vragen bij de zender. Deze methode is niet geschikt voor real-time mediastromen omdat het aanvragen en opnieuw verzenden van een pakket teveel tijd kost en hierdoor niet aan de tijdsbeperkingen van deze applicaties kan worden voldaan.

Dit werk stelt het gebruik van Forward Error Correction (FEC) technieken voor. Deze technieken voegen extra informatie toe aan een datastroom, zodat de ontvanger ontbrekende of incorrecte informatie kan detecteren en corrigeren zodat hertransmissie overbodig wordt. Verschillende technieken, met verschillende achterliggende ideeën, correctiecapaciteit en efficiëntie worden naast elkaar geplaatst.

Vervolgens wordt een overzicht gegeven van methodes die toelaten om FEC te gebruiken ter bescherming van mediastromen, met een bijzondere focus op toepassing bij het Real-time Transport Protocol (RTP). Deze technieken variëren van methodes om zowel pakketverlies als datacorruptie door het netwerk te voorkomen, tot geavanceerde Unequal Error Protection (UEP) methodes, die de belangrijkste informatie van extra bescherming voorzien, en adaptieve technieken, die zich aanpassen aan de netwerkomstandigheden. Enkele van deze technieken worden in dit werk in de praktijk uitgeprobeerd en geëvalueerd. Tot slot wordt er nagedacht over hoe het netwerk zelf intelligent bescherming kan toevoegen aan de mediastromen met behulp van een proxyserver.

Woord vooraf

Van kindsbeen af ben ik steeds geïnteresseerd geweest in hoe dingen werken. Vandaar heb ik binnen de informatica een bijzondere interesse ontwikkeld in netwerktechnologieën, omdat netwerken vaak de basis vormen voor verbluffende applicaties. Multimedia toepassingen genieten een enorme populariteit. Het is voor mij een eer en een uitdaging om deze toepassingen te bestuderen en te zoeken naar oplossingen voor niet-triviale problemen.

Dit werk zou niet tot stand gekomen zijn zonder de steun en de hulp vanuit verschillende hoeken. Via deze weg wil ik deze mensen ook ten zeerste bedanken. Eerst en vooral denk ik aan de professoren en de assistenten van Universiteit Hasselt, die mij gedurende mijn opleiding de kennis hebben onderwezen om dit werk tot een goed einde te brengen. In het bijzonder denk ik aan mijn promotor Prof. Dr. Wim Lamotte en mijn begeleider Maarten Wijnants die mij bij dit werk steeds met tips en suggesties wisten bij te staan. Ook wil ik graag mijn ouders bedanken die mij de mogelijkheid hebben gegeven om de opleiding te volgen en die mij steeds de steun hebben gegeven om te volharden in mijn studies.

Inhoudsopgave

Samenvatting **2**

Woord vooraf **3**

Lijst van figuren **8**

Lijst van tabellen **10**

Lijst van afkortingen **11**

1 Situering **13**

I Literatuurstudie **15**

2 Computernetwerken **16**

2.1 Wat is een netwerk . . . 16
2.2 Verlies van data . . . 16
2.3 Foutdetectie . . . 18
2.3.1 Pariteitscontrole . . . 19
2.3.2 Controlesom . . . 20
2.3.3 Cyclic Redundancy Checks . . . 21
2.4 Klassieke foutafhandeling . . . 23
2.4.1 Automatic Repeat-reQuest . . . 23
2.4.2 Foutafhandeling bij de Internetprotocollen . . . 25
2.5 Kenmerken van multimediastromen . . . 25

3 Forward Error Correction **27**

3.1 Het foutcorrectie model . . . 27
3.2 Terminologie . . . 28
3.3 Repetition code . . . 30
3.4 Hamming codes . . . 32
3.4.1 (7,4) Hamming code . . . 32

3.4.2 Syndroom decoderen . . . 36
3.4.3 Generalisatie . . . 38
3.5 Gegarandeerde foutcorrectie . . . 38
3.6 Reed-Solomon codes . . . 39
3.6.1 Linear Feedback Shift Registers . . . 39
3.6.2 Galois velden . . . 40
3.6.3 RS encoder . . . 43
3.6.4 Encoderen met een LFSR . . . 44
3.6.5 RS decoder . . . 45
3.6.6 Eigenschappen van RS codes . . . 47
3.7 Andere codes en technieken . . . 49
3.7.1 BCH codes . . . 50
3.7.2 LDPC codes . . . 50
3.7.3 Convolutionele codes . . . 52
3.7.4 Interleaving . . . 53
3.7.5 Concatenated codes . . . 54
3.7.6 Product codes . . . 55
3.7.7 Turbo codes . . . 55
3.8 Shannon's Theorema . . . 56

4 Beschermen van multimediastromen 58
4.1 Mediatransport via RTP . . . 58
4.2 Foutcorrectie voor RTP . . . 60
4.3 Reconstructie van volledige pakketten . . . 61
4.3.1 Parity code . . . 61
4.3.2 Gebruik van andere codes . . . 62
4.3.3 Redundante mediastroom . . . 62
4.3.4 Keuze van de pakketten . . . 63
4.3.5 Gevolgen van extra pakketten . . . 63
4.4 Vervangen van volledige pakketten . . . 64
4.4.1 Redundant Audio Coding . . . 64
4.4.2 Transport via piggybacking . . . 64
4.5 Bescherming van de inhoud van pakketten . . . 64
4.6 Unequal Error Protection . . . 65
4.6.1 UDP Lite . . . 66
4.6.2 UEP van payload data . . . 66
4.7 UEP met foutcorrectie . . . 67
4.7.1 Uneven Level Protection in RTP . . . 67
4.7.2 Het Unequal Loss Protection framework . . . 69
4.7.3 Object-gebaseerde UEP . . . 71
4.8 Media-specifieke technieken . . . 71
4.9 Adaptieve FEC . . . 72
4.9.1 Adaptieve algoritmes . . . 72
4.9.2 Gezamenlijke media/FEC ratio selectie . . . 73

II Implementatie 74

5 Applicatie 75
5.1 Doelstellingen . . . 75
5.2 Libraries . . . 75
5.3 Opbouw . . . 77
5.3.1 Netwerk . . . 78
5.3.2 Media . . . 79
5.3.3 RTP encapsulatie . . . 80
5.3.4 Graphical User Interface . . . 81

6 Bescherming in de praktijk 82
6.1 Architectuur . . . 82
6.1.1 Onderscheppen van RTP pakketten . . . 82
6.1.2 Een generieke interface . . . 83
6.1.3 Verwerking van pakketstromen . . . 84
6.2 Pakketbescherming met parity code . . . 85
6.3 Redundant Audio Coding . . . 88
6.3.1 Pakketformaat . . . 88
6.3.2 Beperkingen van Speex . . . 90
6.4 Reed-Solomon bitbescherming . . . 91
6.4.1 RS implementatie . . . 92
6.4.2 Selectie RS codes . . . 92
6.4.3 Pakketformaat . . . 94
6.4.4 Decoderen . . . 94

7 Integratie in NIProxy 96
7.1 Overzicht . . . 96
7.1.1 Bandbreedte beheer . . . 97
7.1.2 Services . . . 97
7.1.3 Toepassingen . . . 98
7.2 FEC service . . . 98
7.2.1 Testapplicatie . . . 99
7.2.2 Implementatie . . . 99
7.2.3 mmfec library . . . 100

III Evaluatie 101

8 Simulatie 102
8.1 Emulators en simulators . . . 102
8.2 NCTUns . . . 102
8.2.1 Emulatie . . . 103
8.2.2 Voor- en nadelen . . . 103

8.3 Opstelling 104
8.3.1 Simulatiemodel 104
8.3.2 Rechtstreekse verbinding 104
8.3.3 Opstelling met NIProxy 105

9 Resultaten 107
9.1 IEEE 802.3 en 802.11 beperkingen 107
9.2 Bandbreedte vereisten 108
9.2.1 Vergelijking van technieken 108
9.2.2 Vergelijking audio en video 109
9.3 Foutcorrectie capaciteit 109
9.3.1 Parity code 110
9.3.2 Redundant Audio Coding 112
9.4 Systeemvereisten 113

10 Uitbreidingen 115
10.1 Geavanceerde media codecs 115
10.1.1 Unequal Error Protection 115
10.1.2 Verbeterde Redundant Audio Coding 116
10.2 Adaptiviteit 116
10.2.1 Meten van de netwerkomstandigheden 116
10.2.2 Feedback kanaal 117
10.2.3 Implementatie van adaptieve algoritmes 117
10.2.4 Adaptiviteit voor de NIProxy 117

11 Conclusie 118

Bibliografie 120

Bijlagen 126

A Het Galois veld $GF(2^3)$ 127

B GUI screenshots 129

C EMIPLIB componentkettingen 131
C.1 `AudioInputChain` 131
C.2 `AudioOutputChain` 131
C.3 `VideoInputChain` 131
C.4 `VideoOutputChain` 133

Lijst van figuren

2.1 Een uitgebreid LAN . . . 17
2.2 Een afbeelding met 10% beschadiging . . . 18

3.1 Het model voor FEC . . . 28
3.2 Een afbeelding beschermd met een R_3 repetition code . . . 31
3.3 Het Venn diagram van de (7,3) Hamming code . . . 33
3.4 Venn diagrammen van de Hamming code met bitfouten . . . 35
3.5 Een afbeelding beschermd met de (7,4) Hamming code . . . 36
3.6 Eenvoudig shift register . . . 40
3.7 Een Linear Feedback Shift Register . . . 40
3.8 Het LFSR voor een (7,3) RS encoder . . . 45
3.9 Een afbeelding beschermd met een (7,3) RS code . . . 48
3.10 Resultaat van een RS code bij niet-willekeurige ruis . . . 49
3.11 Een bigraaf voor een LDPC decoder . . . 52
3.12 Een 4×4 interleaver . . . 53
3.13 Het interleaving proces . . . 54
3.14 Een concatenated code . . . 55
3.15 Schematische voorstelling van een product code . . . 55

4.1 Taken van een RTP zender . . . 59
4.2 Taken van een RTP ontvanger . . . 59
4.3 FEC pakket uit vijf originele pakketten . . . 62
4.4 Single level ULP . . . 68
4.5 Multi-level ULP . . . 69
4.6 Verdeling van bytes in het ULP framework . . . 70

5.1 Technische lagen van de applicatie . . . 76
5.2 Structuur van de applicatie. . . . 77
5.3 Gebruik van de Multicast Tunnel Server . . . 79
5.4 Eenvoudige EMIPLIB componentketting . . . 79

6.1 Interface voor implementatie van bescherming . . . 84
6.2 Onderscheppen van RTP pakketten . . . 86
6.3 Het formaat van een pakket met XOR informatie . . . 86

6.4 Het formaat van een pakket met piggybacked informatie . . . 89
6.5 Het formaat van een RS geëncodeerd pakket 94

7.1 Integratie van de NIProxy in een netwerk 97

8.1 Het simulatiemodel in NCTUns 104
8.2 Opstelling met rechtstreekse verbinding 105
8.3 Opstelling met een verbinding via de NIProxy 106

9.1 Bandbreedte gebruik 108
9.2 Informatie ratio van de parity code voor audio en video. . . . 109
9.3 Efficiëntie evaluatie van de parity code 111
9.4 Details van de efficiëntie evaluatie van de parity code 112
9.5 Efficiëntie evaluatie van Redundant Audio Coding 113
9.6 CPU gebruik van de RS code 113

B.1 Screenshot van de applicatie 129
B.2 Vensters voor instellingen 130

C.1 Invoerketting voor audio 132
C.2 Uitvoerketting voor audio 132
C.3 Invoerketting voor video 134
C.4 Uitvoerketting voor video 134

Lijst van tabellen

3.1 De codewoorden van de (7,4) Hamming code 34
3.2 Syndromen van de (7,4) Hamming code 36
3.3 Status van een LFSR bij elke kloktik 41
3.4 Optelling en vermenigvuldiging in $GF(2)$ 41
3.5 LFSR status bij het encoderen met een (7,3) RS code 45

6.1 Bandbreedte voor RS codewoorden 93

A.1 Verschillende voorstellingen van de symbolen in $GF(2^3)$ . . . 127
A.2 Optellingstabel van $GF(2^3)$ 127
A.3 Vermenigvuldigingstabel van $GF(2^3)$ 128

Lijst van afkortingen

ADSL .... Asymmetric Digital Subscriber Line
AMR codec ...... Adaptive Multi-Rate codec
ARQ .... Automatic Repeat-reQuest
ASCII .... American Standard Code for Information Interchange
BCH code .... Bose-Chaudhuri-Hocquenghem code
BER .... Bit Error Rate
CD .... Compact Disc
CRC .... Cyclic Redundancy Check
DVD .... Digital Versatile Disc
ECC .... Error Correction Coding
ECC RAM ...... Error Correcting Code Random Access Memory
EDM .... Expertisecentrum voor Digitale Media
EMIPLIB .... EDM Media over IP library
ENUt .... EDM Network Utilities
EPB .... Error Protection Block
FEC .... Forward Error Correction
FIR .... Finite Impulse Response
GF .... Galois veld (Galois Field)
GUI .... Graphical User Interface
IIR .... Infinite Impulse Response
iLBC .... Internet Low Bitrate Codec
IP .... Internet Protocol
ISBN .... International Standard Book Number
JPEG .... Joint Photographic Experts Group
LAN .... Local Area Network
LDPC code ...... Low-Density Parity-Check
LFSR .... Linear Feedback Shift Register
ML decoder ...... Maximum Likelihood decoder
mmfec library .... Multimedia FEC library
MPEG .... Moving Picture Experts Group
NAB .... Network Abstraction library
NASA .... National Aeronautics and Space Administration
NAT .... Network Address Translation
NIProxy .... Network Intelligence Proxy

NP .... Nondeterministic Polynomial time
NVE .... Networked Virtual Environment
OSI model .... Open Systems Interconnection Basic Reference Model
PLR .... Packet Loss Rate
PT .... Payload Type
QoE .... Quality-of-Experience
RFC .... Request for Comments
RS code .... Reed-Solomon code
RTCP .... RTP Control Protocol
RTP .... Real-time Transport Protocol
SECDED .... Single Error Correction Double Error Detection
SSRC .... Synchronization Source
TCP .... Transmission Control Protocol
UDP .... User Datagram Protocol
UEP .... Unequal Error Protection
ULP .... Uneven Level Protection
WAN .... Wide Area Network
WLAN .... Wireless LAN

Hoofdstuk 1

Situering

Voor mensen is communicatie één van de meest essentiële zaken. Door met elkaar te praten, zijn we in staat ideeën uit te wisselen en over zaken te discussiëren. We verwachten dat onze communicatie correct verloopt. Anders leidt dit tot misverstanden of ontbrekende informatie. Dit is uiteraard ongewenst. In sommige omgevingen is het echter moeilijk communiceren. Onze communicatie kan door allerlei omgevingsinvloeden worden verstoord. Wie bracht in een discotheek nog nooit het verkeerde drankje van de bar mee voor zijn geliefde, gewoon omdat het moeilijk is om elkaar duidelijk te verstaan door de luide muziek? Als de omstandigheden niet ideaal zijn om via spraak te communiceren, zijn we toch in staat om met wat extra inspanning een boodschap over te brengen. Als iedereen in een discotheek zijn bestelling op een stukje papier noteert, kunnen er geen misverstanden meer zijn. Het kost gewoon wat meer moeite.

Met de opkomst in de jaren '40 van computersystemen in wetenschappelijke onderzoeksgroepen ontstond vanzelf de nood om informatie uit te wisselen tussen verschillende computers. Al snel werden computers met netwerktechnieken met elkaar verbonden. Enkele jaren later werd het met het ontstaan van de voorgangers van het Internet mogelijk om via computerverbindingen documenten uit te wisselen over de hele wereld. Netwerkverbindingen vormen een heel belangrijk aspect van de informatica en een wereld zonder netwerken is bijna niet meer voor te stellen.

Net als bij spraakcommunicatie is het belangrijk dat ook communicatie tussen computers foutloos verloopt. Computers vervullen nog altijd een taak voor mensen en we verwachten dat informatie die we ontvangen ook de informatie is die we wilden ontvangen. Netwerken hebben echter hun eigen stoorzenders. Net als in het echte leven zullen computers meer moeite moeten doen om stoorzenders te slim af te zijn.

Sinds de laatste decennia hebben mensen duidelijk de weg naar digitale media gevonden. Een modern gezin beschikt tegenwoordig minstens over een computer met Internetaansluiting, enkele mobiele telefoons en een

abonnement op digitale televisie. Met deze digitalisering blijft de interesse in multimedia toepassingen, waaronder audio- en videotoepassingen en telecommunicatie worden verstaan, stijgen. Om te kunnen praten met iemand die niet in de buurt is, kan men tegenwoordig gebruik maken van audio- en videogesprekken via Internet. Daar deze toepassingen in essentie transport van informatie via allerlei wegen tussen computers vereisen, is het niet zo onzinnig om te onderzoeken hoe deze toepassingen kunnen beschermd worden tegen de invloed van stoorzenders. Dit werk bestudeert de mogelijkheden om communicatie tussen computers te beschermen en doet dit in het bijzonder gericht op multimedia toepassingen. Deze bescherming zorgt ervoor dat toepassingen beter kunnen functioneren, wat de gebruikers maar al te graag hebben.

De rest van dit werk is als volgt georganiseerd. In deel I wordt een overzicht gegeven van wat er in de literatuur al geschreven werd betreffende dit onderwerp. Hoofdstuk 2 overloopt enkele manieren die toelaten om te detecteren wanneer er iets mis gaat tijdens het versturen van informatie tussen computers. Detectie alleen lost het probleem van transmissiefouten echter niet op. Hoofdstuk 3 beschrijft daarom technieken die toelaten om gedeeltelijk foutief ontvangen informatie toch nog te herstellen. Hoofdstuk 4 legt uit hoe deze technieken kunnen worden aangewend om multimedia toepassingen te verbeteren.

Deel II beschrijft een praktische toepassing van enkele technieken uit de literatuurstudie. In hoofdstuk 5 wordt de ontwikkeling van een applicatie besproken die gebruik maakt van multimediale communicatie. Hoe deze communicatie kan worden beschermd tegen transmissiefouten, wordt uitgelegd in hoofdstukken 6 en 7. In deel III worden de gebruikte technieken geëvalueerd. De testomgeving wordt beschreven in hoofdstuk 8. De resultaten van de evaluatie staan vermeld in hoofdstuk 9. Hoofdstuk 10 geeft een overzicht van eventuele uitbreidingen voor de ontwikkelde software. Hoofdstuk 11 concludeert dit werk.

Deel I

Literatuurstudie

Hoofdstuk 2

Computernetwerken

2.1 Wat is een netwerk

Een netwerk maakt een fysieke verbinding tussen verschillende computers. Via deze verbinding kunnen de verbonden toestellen data met elkaar uitwisselen. De eenvoudigste vorm van een computernetwerk verbindt twee computers rechtstreeks met elkaar.

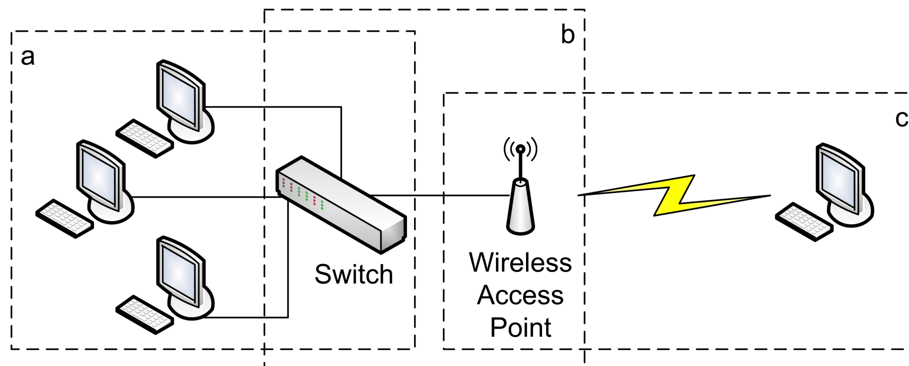
De fysieke verbinding kan via verschillende technologieën tot stand worden gebracht: bij een bedraad netwerk kan dit bijvoorbeeld gebeuren via elektrische signalen die door koperdraad gestuurd worden. Een draadloze verbinding kan gebruik maken van een waaier van signaaldragers, zoals infraroodverbindingen en radiogolfverbindingen.

Naast fysieke verbindingslijnen bevat een netwerk vaak ook netwerkapparaten met verschillende functies. Een eenvoudig voorbeeld hiervan is een netwerkswitch, die gebruikt wordt in Local Area Networks (LANs). Dit toestel vormt een centraal punt in het netwerk. Elke computer is fysiek verbonden met het toestel. Als een computer een datablok naar een andere computer wil versturen, verstuurt hij de data naar de netwerkswitch, die de data op zijn beurt verderstuurt naar de ontvanger. Aan de hand van figuur 2.1 worden enkele concepten verduidelijkt.

Het is niet de bedoeling om in deze tekst een verdere uiteenzetting te geven over welke netwerktechnologieën er zoal bestaan, en hoe deze werken. [54] is een goede inleiding in deze materie.

2.2 Verlies van data

Netwerkverbindingen zijn niet perfect. Wanneer een datablok over een netwerk wordt verstuurd, moet er rekening mee worden gehouden dat dit datablok mogelijk beschadigd raakt, of zelfs verloren gaat. Deze transmissiefouten zijn het gevolg van de fysieke opbouw van netwerken. Men onderscheidt twee bronnen van dataverlies:



Figuur 2.1: *Een uitgebreid LAN. (a) een bedraad netwerkgedeelte verbindt drie computers met een netwerkswitch; (b) netwerkapparaten; (c) draadloos netwerkgedeelte: een computer die via radiosignalen met het Wireless Access Point verbonden is.*

- Storing van de fysieke verbindingen
- Problemen bij de netwerkapparatuur

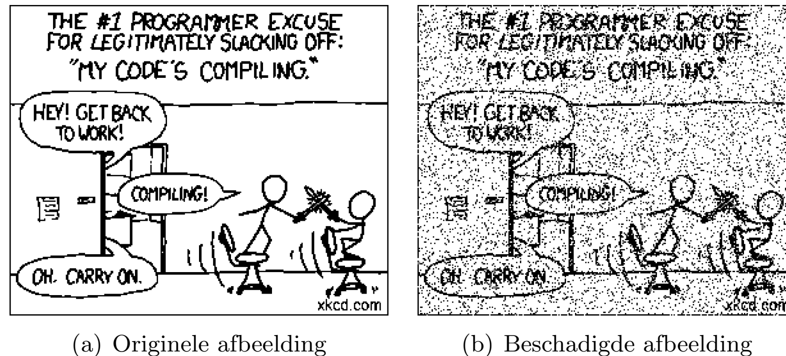
Fysieke verbindingen zijn onderhevig aan vele invloeden. Een elektrisch signaal kan in de war gebracht worden door magnetische straling. Radioverbindingen worden blootgesteld aan stoorzenders die op dezelfde radiofrequentie uitzenden en kunnen slechts een beperkte afstand overbruggen. Deze storingen zijn in het dagelijkse leven zeer frequent: de microgolfoven en het draadloze LAN van een buurman genereren allebei signalen op dezelfde radiofrequentie als het eigen draadloze LAN, waardoor dit gestoord wordt. De kanalen voor breedband Internetverbinding, waarover meer dan twee miljoen[1] Belgen beschikken, zijn dezelfde kanalen die gebruikt worden voor het publieke telefoonnetwerk (in het geval van ADSL aansluitingen) of het televisiedistributienetwerk (voor kabelaansluitingen), waardoor verschillende signalen elkaar kunnen beïnvloeden.

Zoals beschreven staat in [54], bevat een netwerk vaak apparaten met geheugen, zodat inkomende datapakketten tijdelijk bewaard kunnen worden vooraleer ze worden verwerkt door het apparaat. Als het dataverkeer echter hoog is, kan er netwerkcongestie optreden. In deze situatie zit het geheugen van een netwerkapparaat vol, waardoor datapakketten noodgedwongen moeten worden weggegooid. Bijgevolg gaan volledige datapakketten verloren. Door defecte apparaten, ten gevolge van bijvoorbeeld beschadiging van het geheugen, kan data ook beschadigd worden.

Figuur 2.2 toont een afbeelding waarbij 10% van de bits werd beschadigd. Omdat dit een zwart-witafbeelding is, kan elke pixel worden voorgesteld met

[1] Bron: Nationaal Instituut voor de Statistiek (`http://www.statbel.fgov.be`)

één bit. Elke bitfout resulteert bijgevolg in de foutieve weergave van een pixel.



(a) Originele afbeelding (b) Beschadigde afbeelding

Figuur 2.2: *Een afbeelding waarbij 10% van de bits wordt beschadigd.*

Beschadigde of verloren data is uiteraard ongewenst. Het is belangrijk voor computers die communiceren over een netwerk, dat ze transmissiefouten kunnen herkennen en hiervoor een oplossing kunnen bieden, zodat ze toch data kunnen uitwisselen. In de volgende paragrafen wordt een overzicht gegeven van technieken die gebruikt worden om databeschadiging te detecteren. Vervolgens worden enkele eenvoudige technieken besproken om de data die beschadigd werd opnieuw te laten versturen door de zender.

2.3 Foutdetectie

Het vinden van transmissiefouten is de eerste stap tot het corrigeren ervan. Bij het ontvangen van een aantal bits, kan een computer echter niet zomaar bepalen of er al dan niet fouten in de ontvangen data aanwezig zijn. Om dit mogelijk te maken, moet de zender extra informatie bij de te versturen bits toevoegen. Deze informatie wordt *redundante data* genoemd, omdat ze geen relevante informatie bevat betreffende de verzonden data. Redundante bits hebben enkel nut wanneer er transmissiefouten optreden. In dat geval laten ze toe om de fouten te detecteren en eventueel te corrigeren, zoals zal worden aangetoond in hoofdstuk 3.

Redundante informatie wordt ook buiten het informatica domein gebruikt om correctheid te controleren. ISBN nummers zijn hier een typisch voorbeeld van [22]. Het laatste cijfer van deze nummers is redundant. Om de correctheid van een ISBN nummer te controleren, wordt elk cijfer van het nummer afwisselend vermenigvuldigd met 1 of 3. Het redundante cijfer wordt zo gekozen dat de som van deze producten voor een geldig ISBN nummer deelbaar is door 10. Voorbeeld 1 toont de berekeningen voor het nummer 978-0-321-48100-9. Deze berekening is enkel correct voor de nieuwe ISBN-13 standaard, die sinds 2007 in gebruik is. Bij oude ISBN nummers

met 10 cijfers is het laatste cijfer ook redundant, maar om de correctheid te controleren moet een andere berekening gebruikt worden. Deze staat beschreven als voorbeeld in de inleiding van [37].

Voorbeeld 1 *Het laatste cijfer van het ISBN nummer 978-0-321-48100-9 is redundant. De som van de producten is 80. Omdat 80 deelbaar is door 10, is het een geldig nummer.*

ISBN	*9*	*7*	*8*	*0*	*3*	*2*	*1*	*4*	*8*	*1*	*0*	*0*	***9***	
×	*1*	*3*	*1*	*3*	*1*	*3*	*1*	*3*	*1*	*3*	*1*	*3*	*1*	
product	*9*	*21*	*8*	*0*	*3*	*6*	*1*	*12*	*8*	*3*	*0*	*0*	*9*	***80***

Het vervolg van deze sectie bespreekt enkele populaire foutdetectie methoden die worden gebruikt in computersystemen.

2.3.1 Pariteitscontrole

De eenvoudigste methode om te controleren of een ontvangen reeks bits dezelfde is als degene die werd verstuurd, is de pariteitscontrole [61]. De te versturen data wordt opgesplitst in blokken van n bits. Bij elk blok wordt één extra bit toegevoegd, die de *pariteitsbit* wordt genoemd. Bij deze methode wordt het aantal bits van een datablok die als waarde 1 hebben, geteld. Er zijn twee varianten van deze methode, en de variant bepaalt welke waarde de pariteitsbit krijgt:

even pariteit Er wordt een 1 toegevoegd als er een oneven aantal 1-bits in het datablok aanwezig is. Dit maakt het totaal aantal 1-bits *even*. Als het aantal 1-bits al even was, wordt een 0 toegevoegd, zodat er nog steeds een even aantal 1-bits is.

oneven pariteit Hier geldt het omgekeerde: een 1 wordt toegevoegd als dit nodig is om het aantal 1-bits *oneven* te maken.

De ontvanger moet voor elke ontvangen sequentie van $n + 1$ bits enkel tellen hoeveel bits een waarde 1 hebben. Bij even pariteit moet dit aantal even zijn, bij oneven pariteit juist niet. Als de ontvanger merkt dat deze aanname niet klopt, is er een transmissiefout opgetreden.

Voorbeeld 2 *Stel* 1001101 *is een te versturen datablok. Dit blok bevat een even aantal 1-bits. Als even pariteit wordt verondersteld, wordt een 0 toegevoegd. De verstuurde sequentie is* 10011010*. Als dit bericht wordt ontvangen, bevat het nog steeds een even aantal 1-bits, en wordt het dus aanvaard als een correct bericht. Als echter het datablok beschadigd raakt, en* 01001**0**010 *wordt ontvangen, dan zal het aantal 1-bits oneven zijn. De voorwaarde van even pariteit voldoet niet langer, en de transmissiefout wordt ontdekt.*

Deze methode is echter weinig betrouwbaar voor netwerktransmissie. De aandachtige lezer heeft ongetwijfeld wel gemerkt dat, wanneer in het datablok van voorbeeld 2 twee bits worden omgewisseld, de pariteit weer klopt, terwijl het bericht toch foutief is.

Pariteitscontrole kent echter wel nuttige toepassingen in situaties waar een bitfout zeer zelden voorkomt, en de kans op twee of meer fouten verwaarloosbaar is, zoals in verschillende computer hardware componenten. Een andere toepassing vindt men bij de opslag van ASCII tekst [49]: daar een ASCII symbool wordt voorgesteld door zeven bits, kan extra bescherming worden toegevoegd met een pariteitsbit, waardoor een beschermd symbool wordt voorgesteld door één byte. Deze voorstelling leent zich uitstekend voor efficiënte opslag.

2.3.2 Controlesom

Pariteitscontrole is weinig betrouwbaar: enkel een oneven aantal bitfouten wordt gedetecteerd. Een controlesom, *checksum* in het Engels, laat toe om meer transmissiefouten te detecteren. Er bestaan verschillende varianten, maar het basisidee is steeds hetzelfde: de te versturen data wordt ingedeeld in blokken van gelijke grootte. De blokken worden bitgewijs bij elkaar opgeteld. Het resultaat van deze bewerking is de controlesom. Deze som kan na transmissie opnieuw worden uitgerekend. Indien de som niet overeenkomt met de oorspronkelijke som, zijn er bits gewijzigd.

De controlesom wordt als redundante data toegevoegd aan de te verzenden data. Vaak wordt ervoor gekozen om de controlesom zo te berekenen, dat ze kan worden voorgesteld met evenveel bits als de grootte van de datablokken. Dit kan door de *one's complement* [42] som te gebruiken. Het gebruik van *one's complement* is beter dan het gebruik van *two's complement* [42], omdat de controlesom dan even gevoelig is voor fouten in alle bit posities [7]. Als vervolgens niet de berekende som zelf, maar zijn complement wordt toegevoegd aan de data, laat dit efficiëntere controle van de correctheid van de ontvangen data toe: als een ontvanger de controlesom berekent van de volledige ontvangen datasequentie, inclusief de toegevoegde controlesom, dan zal het resultaat 0 zijn bij een bericht zonder fouten. *One's complement* operaties met 16- of 32-bit getallen lenen zich uitstekend voor een eenvoudige en snelle hardware implementatie. Voorbeeld 3 berekent de controlesom voor de tekst “hello”.

Voorbeeld 3 *In dit voorbeeld wordt de tekst “hello” verzonden. Er wordt gebruik gemaakt van 8 bits per datablok, wat overeenkomt met één letter per datablok. De datablokken worden opgeteld volgens de standaardregels voor de binaire som: van rechts naar links wordt elke kolom bits opgeteld. Elke som van twee 1-bits resulteert in een 0 en een carry bit. Een carry bit wordt aan de links aangrenzende kolom toegevoegd. De optelling gebeurt met de*

one's complement *som. Dit betekent dat de carry bits die uiteindelijk niet binnen het 8 bits grote datablok passen, geschrapt worden uit het resultaat en vervolgens bij de som worden opgeteld door ze naar de meest rechtse posities te verschuiven.*

De zender voegt het complement van de som toe aan het bericht. Aan de ontvangstkant wordt opnieuw een controlesom berekend. Het resultaat is 1111 1111*, wat in een* one's complement *representatie gelijk is aan 0. De data werd dus correct ontvangen.*

Zender		*Ontvanger*	
0110 1000	*'h'*	0110 1000	
0110 0101	*'e'*	0110 0101	
0110 1100	*'l'*	0110 1100	
0110 1100	*'l'*	0110 1100	
+ 0110 1111	*'o'*	0110 1111	
10 0001 0100	*som*	+ 1110 1001	*complement*
+ 10	*one's complement*	10 1111 1101	
0001 0110	*controlesom*	+ 10	
1110 1001	*complement*	1111 1111	*= 0*

Theoretisch gezien bestaat de kans dat een aantal bits zo wordt beschadigd, dat de controlesom toch nog klopt. Als in het bovenstaand voorbeeld de laatste bit van zowel de letter h als de letter e wordt omgewisseld, dan is de som toch nog gelijk. De controlesom kan dus, net als de pariteitscontrole, slechts gegarandeerd één bitfout detecteren. Maar de kans dat exact die bits worden omgewisseld waardoor de controlesom toch klopt, is zo klein, zodat de techniek toch veelgebruikte toepassingen kent, zoals in de Internetprotocollen (sectie 2.4.2).

2.3.3 Cyclic Redundancy Checks

De Cyclic Redundancy Check (CRC) is een meer geavanceerde methode om een controlesom te berekenen [62]. Deze methode steunt op veelterm algebra. Het te versturen bericht wordt voorgesteld als een veelterm. Deze wordt gedeeld door een *generator veelterm.* De restterm wordt gebruikt als controlesom voor het bericht.

Om een bitsequentie voor te stellen als een veelterm, en hierop delingen uit te voeren, moet de lezer vertrouwd zijn met Galois velden (GF) [8]. In dit wiskundig gebied wordt de som en de vermenigvuldiging gedefinieerd op een eigen set van elementen. Met *GF(2)* wordt een Galois veld aangeduid dat twee elementen bevat. Op bit niveau zijn slechts twee elementen, met name 0 en 1. *GF(2)* is bijgevolg geschikt om operaties te definiëren voor het bit niveau. *GF(2)* is een eenvoudig veld: de som wordt uitgerekend als de gewone som met gehele getallen, maar *modulo 2.* Voor de vermenigvuldiging wijzigt er niets ten opzichte van gewone algebra.

Om een bitsequentie als een veelterm voor te stellen, wordt elke bit de coëfficiënt van een term. Als voorbeeld: de sequentie 1101 heeft als veelterm $1 * x^3 + 1 * x^2 + 0 * x + 1$, of korter, $x^3 + x^2 + 1$. Met dergelijke veeltermen kunnen vervolgens bewerkingen worden uitgevoerd, rekening houdend met de regels van *GF(2)*. De som van $x^3 + x^2 + 1$ met een andere veelterm, bijvoorbeeld $x^2 + x$, is $x^3 + x + 1$. Merk op dat de gewone som van x^2 met x^2 een term $2x^2$ zou geven, maar wegens de modulo 2, is dit in *GF(2)* de term $0x^2$, die kan worden weggelaten.

Een uitgebreidere discussie over Galois velden zou ons voorlopig te ver leiden. Dit onderwerp wordt later met meer detail behandeld bij de bespreking van de Reed-Solomon code in sectie 3.6.

Veelterm algebra [39] leert dat de restterm na deling altijd minstens één graad lager is dan de graad van de deler. Om een controlesom te berekenen van maximaal r bits, mag de graad van de restterm maximaal $r - 1$ bedragen. De gebruikte generator veelterm moet dus van graad r zijn. Voor een bitsequentie van m bits voegt het CRC algoritme eerst r 0-bits achteraan de sequentie toe. De veelterm die de sequentie voorstelt, heeft nu een graad van $m + r - 1$. Door de uitbreiding met 0-bits, is de graad steeds hoger of gelijk aan de graad van de generator veelterm, waardoor de deling steeds uitvoerbaar is. Na deling door de generator veelterm, met graad r, vormen de coëfficiënten van de restterm de r bits lange controlesom.

De ontvanger heeft, net als bij de methode met de gewone controlesom uit sectie 2.3.2, twee mogelijkheden om te controleren of een ontvangen bericht transmissiefouten bevat of niet. De meest voor de hand liggende methode berekent opnieuw de controlesom, en vergelijkt deze vervolgens met de door de zender berekende versie. Uit de wiskunde is echter geweten dat, wanneer de rest van een deling bij het deeltal wordt opgeteld, en wanneer deze som vervolgens opnieuw wordt gedeeld, de rest van de nieuwe deling 0 is. Dit kan worden toegepast: de controlesom wordt bij het oorspronkelijke bericht opgeteld. Als vervolgens het geheel gedeeld wordt door de generator veelterm, dan is de rest van de nieuwe deling 0 indien het bericht correct ontvangen werd. De uitbreiding van het oorspronkelijke bericht met r 0-bits zorgt ervoor dat de restterm, die r bits lang is, bij de optelling enkel de toegevoegde 0-bits overschrijft, waardoor het mogelijk wordt om het oorspronkelijke bericht en de restterm uit elkaar te houden.

Voorbeeld 4 *Ter demonstratie wordt voor de korte sequentie* 1101000 *een 4-bit controlesom berekend. Als generator veelterm wordt* $x^4 + x + 1$ *gebruikt. Na uitbreiding met vier 0-bits, wordt de sequentie voorgesteld door de veelterm* $x^{10} + x^9 + x^7$. *Onderstaande staartdeling bewijst dat de restterm* $x^2 + 1$ *is. Bijgevolg is de controlesom* 0101.

$$
\begin{array}{llllll|l}
 & x^{10} & +x^9 & +x^7 & & & x^4+x+1 \\
\cline{7-7}
-(& x^{10} & & +x^7 & +x^6) & & x^6+x^5+x \\
\cline{1-5}
 & & x^9 & & +x^6 & & \\
 & -(& x^9 & & +x^6 & +x^5) & \\
\cline{2-6}
 & & & & & x^5 & \\
 & & & & -(& x^5+x^2+1) & \\
\cline{5-6}
 & & & & & x^2+1 & \\
\end{array}
$$

De keuze van de generator veelterm is belangrijk om te bepalen hoe goed een CRC transmissiefouten kan detecteren [27]. Het zoeken naar geschikte veeltermen is niet eenvoudig, en wordt in deze tekst niet verder besproken. Er bestaan algemeen gebruikte veeltermen, die door internationale organisaties werden gestandaardiseerd. Zo is er bijvoorbeeld de veelterm $x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$, die gestandaardiseerd werd in de IEEE 802.3 specificatie (Ethernet). Volgens [27] kan deze veelterm slechts tot vier willekeurige bitfouten gegarandeerd terugvinden in berichten met de maximale lengte van een Ethernet frame (12112 bits). In dezelfde paper wordt echter bewezen dat andere veeltermen tot zes bitfouten gegarandeerd kunnen terugvinden in een bericht van die grootte.

Opmerkelijk is de veelterm $x + 1$. De graad van deze veelterm is 1. De controlesom heeft dus een lengte van 1 bit. Een CRC met deze veelterm gebruiken heeft hetzelfde resultaat als het gebruik van de methode met de pariteitsbit uit sectie 2.3.1, en kan dus slechts één bitfout gegarandeerd terugvinden.

2.4 Klassieke foutafhandeling

In de vorige sectie werd beschreven hoe transmissiefouten kunnen worden gedetecteerd met behulp van redundante data. Uiteraard heeft detectie enkel nut als er vervolgens handelingen worden ondernomen om de transmissiefouten te verbergen voor de applicaties waarvoor de informatie uiteindelijk bedoeld is. Het opnieuw verzenden van de beschadigde data is één van de mogelijkheden. Hertransmissie kan worden geïmplementeerd in de protocollen van de transport- of datalinklaag van het OSI netwerkmodel [54], zodat processen in de applicatielaag niet merken dat data beschadigd wordt door het netwerk.

2.4.1 Automatic Repeat-reQuest

Om hertransmissie mogelijk te maken, moet de ontvanger aan de zender laten weten wat ontvangen wordt en wat niet. Indien gebruik wordt gemaakt van *Positive Acknowledgement*, stuurt de ontvanger voor elk correct ontvangen netwerkframe een bevestiging naar de zender. Zo weet de zender welke

frames opnieuw verzonden moeten worden: wanneer een frame niet wordt bevestigd binnen een bepaald tijdsinterval, wordt het opnieuw verstuurd. Deze algemene methode wordt *Automatic Repeat-reQuest* (ARQ) genoemd. Er zijn drie ARQ modellen [54]:

Stop-and-Wait In het eenvoudigste model verstuurt de zender het volgende frame pas wanneer het vorige bevestigd is. Wordt het vorige niet bevestigd, dan wordt dit opnieuw verstuurd van zodra een timer afloopt, totdat een bevestiging wordt ontvangen.

Go-back-n Het *Stop-and-Wait* model heeft als nadeel dat datatransport zeer traag verloopt op netwerken met een grote *delay* (de tijd die een bit nodig heeft om de ontvanger te bereiken), omdat het lang duurt vooraleer een frame op zo'n netwerk bevestigd kan worden. Bijgevolg wordt slechts een klein deel van de netwerkcapaciteit benut. In het *Go-back-n* model kan de zender tot n frames versturen vooraleer een bevestiging wordt ontvangen. De ontvanger verstuurt een bevestiging van zodra hij een frame ontvangt. De zender mag maximaal $n - 1$ volgende frames versturen in de periode tussen het versturen van het eerste frame en het ontvangen van een bevestiging hiervoor. In het ideale geval, waarbij het versturen van n frames zo lang duurt dat het eerste frame bevestigd wordt net nadat het laatste werd verstuurd, kan de zender onmiddellijk beginnen met het verzenden van het eerste frame van de volgende n frames. Zo wordt de volledige netwerkcapaciteit benut. Van zodra de ontvanger een frame niet bevestigt, zal de zender stoppen na het versturen van n frames. Indien hij aan de hand van een timeout concludeert dat de ontvanger het frame niet meer zal bevestigen, worden dit frame en alle volgende frames die al verzonden waren, opnieuw verstuurd. Op deze manier zal de ontvanger alle frames steeds in de juiste volgorde kunnen ontvangen. De zender moet n frames in een buffer kunnen bijhouden, omdat deze mogelijk opnieuw zullen moeten worden verstuurd.

Selective Repeat Op netwerken waar regelmatig frames verloren gaan of beschadigd worden, is het *Go-back-n* model niet efficiënt: het verlies van één frame leidt tot hertransmissie van dit frame en alle daarop volgende verzonden frames. Bij *Selective Repeat* worden enkel die frames opnieuw verzonden die niet correct werden ontvangen. De ontvanger moet in dit model een buffer voorzien waarin alle correct ontvangen frames, die aankomen na een verloren frame, worden bewaard. Pas als het verloren frame via hertransmissie correct werd ontvangen, kan de inhoud van de frames in de juiste volgorde aan de applicatielaag worden doorgegeven.

2.4.2 Foutafhandeling bij de Internetprotocollen

Het Internet is een wereldwijd netwerk waarbij data vaak over enorme afstanden en door ontelbare netwerkapparaten wordt getransporteerd om zijn doel te bereiken. Het is dus niet verwonderlijk dat hierbij data beschadigd of verloren geraakt. De Internetprotocollen voorzien dan ook in enkele technieken voor foutafhandeling.

UDP, het onbetrouwbare datagram protocol, controleert de correctheid van de data met behulp van een 16-bit controlesom (sectie 2.3.2) [7]. Wanneer de data echter niet correct is, wordt het datagram weggegooid. UDP voorziet geen hertransmissie. Vandaar het onbetrouwbare karakter van het protocol: de data die wordt afgeleverd, is gegarandeerd correct, maar het protocol verzekert niet dat alle data wordt afgeleverd, noch dat de data in de juiste volgorde wordt afgeleverd.

TCP maakt net als UDP gebruik van een 16-bit controlesom om de correctheid van een datapakket te verifiëren. TCP is echter een betrouwbaar protocol, en voorziet hertransmissie van beschadigde of verloren data. Hiervoor maakt het protocol gebruik van een variant op de Go-back-n ARQ methode [54].

2.5 Kenmerken van multimediastromen

Correctie van foutief ontvangen datastromen met behulp van hertransmissie is eenvoudig en laat toe om informatie betrouwbaar te transporteren over een onbetrouwbaar netwerk. Voor veel niet real-time toepassingen is hertransmissie een goede keuze voor foutafhandeling.

Op het moderne Internet wordt echter steeds vaker gebruik gemaakt van multimedia toepassingen, zoals onder andere voice-over-IP, videotelefonie, teleconferencing en live streaming van sportevenementen. Deze toepassingen maken gebruik van real-time audio- en videostromen. Real-time stromen zijn in essentie multimedia bestanden die een gebruiker kan afspelen terwijl ze op hetzelfde moment worden gedownload.

Real-time datastromen hebben als bijkomende vereiste ten opzichte van niet real-time stromen dat informatie tijdig moet arriveren bij de ontvanger. Neem als voorbeeld een videostroom die wordt bekeken via het Internet. De zender stuurt voortdurend netwerkpakketten met beelden naar de ontvanger. De ontvanger toont deze beelden vervolgens via een monitor en zo kan de gebruiker de videostroom waarnemen. Een beeld moet echter op het juiste tijdstip worden weergegeven, opdat de gebruiker de videostroom optimaal kan ervaren. Dit wordt omschreven als een hoge Quality-of-Experience (QoE) voor de gebruiker. Indien beelden niet tijdig arriveren, en bijgevolg enkel te laat of niet kunnen worden getoond, zal de gebruiker een haperende videosequentie te zien krijgen. In dit geval is de QoE van de applicatie laag. Uiteraard is een goede QoE cruciaal voor een geslaagde applicatie.

De QoE van een real-time multimedia toepassing wordt op twee manieren beïnvloed door het netwerk:

- Het netwerk moet de datapakketten tijdig afleveren, zodat de inhoud op het juiste moment kan worden weergegeven aan de gebruiker;
- Het netwerk moet de datapakketten correct afleveren, zodat de correcte informatie wordt weergegeven aan de gebruiker.

Het tijdig afleveren wordt in de war gestuurd door de *delay* en *jitter* van een netwerk. De effecten van deze problemen kunnen worden opgevangen door gebruik te maken van buffers. Deze technieken behoren echter niet tot de kern van dit werk. Hoe dit wordt afgehandeld in het Real-time Transport Protocol (RTP) [48], dat speciaal werd ontwikkeld voor het transport van real-time stromen, staat beschreven in hoofdstuk 6 van [43].

Dit werk richt zich op het correct afleveren van pakketten in multimedia toepassingen. Correctheid is minder belangrijk voor de QoE bij real-time stromen dan de timing [43]. Eén of enkele pixelfouten in een videobeeld zal de gebruiker minder snel merken dan een beeld dat enkele milliseconden te laat wordt getoond en daardoor de continuïteit verbreekt. Toch mag het belang van de correctheid van informatie niet onderschat worden: indien fouten zichtbaar worden, zal de gebruiker de videostroom minder goed ervaren en zal de QoE dalen. Om dit te vermijden is foutafhandeling ook in multimedia toepassingen belangrijk.

Real-time stromen kunnen echter moeilijker gebruik maken van de hertransmissie technieken die werden beschreven in sectie 2.4. Als een netwerkpakket opnieuw moet worden verzonden, gaat kostbare tijd verloren en zal dit opnieuw verzonden pakket mogelijk te laat arriveren om nog op het juiste tijdstip getoond te kunnen worden. In het volgende hoofdstuk worden technieken behandeld die toelaten om beschadigde pakketten bij de ontvanger te herstellen. Deze technieken, die Forward Error Correction (FEC) technieken worden genoemd, verhogen bijgevolg de QoE, zonder gebruik te moeten maken van hertransmissie.

Hoofdstuk 3

Forward Error Correction

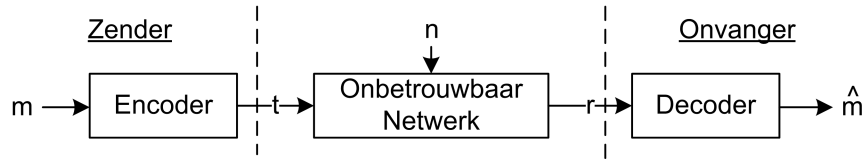
In het vorige hoofdstuk werd vermeld dat communicatiekanalen niet altijd even betrouwbaar zijn. Data kan betrouwbaar worden uitgewisseld door fouten op te sporen en de beschadigde informatie opnieuw te verzenden. Hertransmissie is echter te traag voor bepaalde toepassingen. Er is bijgevolg nood aan technieken die toelaten om fouten te detecteren *en* te herstellen in een omgeving waar hertransmissie geen optie is. Dit hoofdstuk beschrijft dergelijke technieken. Deze technieken worden Forward Error Correction (FEC) technieken genoemd. Men spreekt ook wel van Error Correction Coding (ECC).

3.1 Het foutcorrectie model

Net als bij de foutdetectie technieken uit het vorige hoofdstuk, wordt bij FEC technieken tijdens de uitwisseling van een bericht redundante informatie toegevoegd. Aan de hand van deze informatie kan de ontvanger transmissiefouten opsporen en herstellen. Dit proces kan schematisch worden afgebeeld zoals in het blokdiagram in figuur 3.1: de zender wil bericht m versturen. Een encoder transformeert dit bericht naar een geschikt formaat met redundante informatie, voorgesteld als t. Deze bitsequentie wordt vervolgens over een onbetrouwbaar kanaal verstuurd naar de ontvanger. Deze ontvangt bitsequentie r, die gelijk is aan t, maar waarbij een ruisterm n werd toegevoegd, die de datacorruptie voorstelt. Een decoder bij de ontvanger probeert tot slot uit de ontvangen bitsequentie r het oorspronkelijke bericht m te reconstrueren, wat resulteert in bericht $\hat{m}$. m en $\hat{m}$ kunnen verschillen indien de FEC techniek niet in staat was om alle fouten, die door n werden geïntroduceerd, te corrigeren. Bitsequenties worden in de literatuur soms als bitvectoren voorgesteld. Dit laat wiskundige uitdrukkingen toe zoals

$$r = t + n, \tag{3.1}$$

waarbij de *modulo 2* optelling geldt. Een FEC techniek wordt ook een FEC code genoemd, omdat ze gebruik maakt van een encoder/decoder paar.



Figuur 3.1: *Het model voor Forward Error Correction. De zender encodeert bericht m tot bitvector t. t wordt beschadigd door het netwerk en resulteert in vector $r = t + n$. Uit r probeert de decoder m te reconstrueren, wat resulteert in $\hat{m}$.*

3.2 Terminologie

In het domein van FEC worden termen gebruikt die een introductie vragen. De termen en definities die volgen worden ook gebruikt in de literatuur [12, 33, 37].

Symbolen

Een te versturen datasequentie wordt voorgesteld als een sequentie of een vector van *symbolen*. In de meest voor de hand liggende situatie wordt een datasequentie voorgesteld met symbolen uit de verzameling $\{0, 1\}$, dus op bit-niveau. Meer complexe codes kunnen gebruik maken van een andere verzameling symbolen die door de code wordt gedefinieerd. Uiteraard bestaat de data nog steeds uit bits, maar de encoder/decoder groepeert in dit geval bits om symbolen te vormen, waarop vervolgens de berekeningen worden uitgevoerd.

Binaire/niet-binaire code

Bij een binaire code wordt gebruik gemaakt van symbolen uit de verzameling $\{0, 1\}$. Een niet-binaire code gebruikt een andere verzameling in zijn berekeningen.

Bericht en codewoord

Een bericht bestaat uit de symbolen die door de encoder van de zender in één keer worden verwerkt. Het codewoord is het resultaat van de encoder. In figuur 3.1 wordt het bericht voorgesteld als m, en het codewoord als t. Voor de ontvanger is r het codewoord. Dit kan echter beschadigd zijn tijdens het transport over het netwerk.

Gewicht van een bitsequentie

Het gewicht van een bitsequentie is een zeer eenvoudig begrip: het gewicht komt overeen met het aantal 1-bits in de sequentie. Een codewoord *100101* heeft bijgevolg een gewicht 3.

Hamming afstand

De Hamming afstand bepaalt in hoeverre twee even lange bitsequenties van elkaar verschillen. De waarde voor deze afstand is gelijk aan het aantal bitposities waarin beide sequenties een andere waarde voor de bit hebben. Sequenties *100101* en *101111* hebben verschillende waardes voor twee bitposities, dus de Hamming afstand tussen deze sequenties is 2.

Systematische code

Indien de symbolen van het bericht m na het encoderen integraal terug te vinden zijn in het codewoord, dan spreekt men van een systematische code. Een niet-systematische code transformeert de symbolen van het bericht naar een volledig andere sequentie.

(n,k) code

Een (n,k) code is een code waarbij de encoder een bericht van k symbolen omvormt tot een codewoord van n symbolen. Merk op dat k en n niet het aantal bits zijn, maar het aantal symbolen. Voor een binaire code is het aantal symbolen uiteraard gelijk aan het aantal bits. Een (7,4) code encodeert vier symbolen tot een codewoord met lengte 7.

Informatie ratio

Het informatie ratio duidt het percentage bandbreedte aan dat effectief wordt benut voor de uitwisseling van informatie. Dit wordt verlaagd naar mate er meer redundantie wordt toegevoegd door de code. Een (7,4) code heeft een ratio $R = \frac{4}{7}$, omdat vier symbolen informatie worden verstuurd met behulp van zeven symbolen. De code gebruikt drie symbolen voor redundante informatie.

Bit Error Rate

Het BER (Bit Error Rate) is het percentage van de informatie dat foutief bij de ontvanger arriveert. Slechte netwerkomstandigheden doen het BER stijgen. Een FEC code moet het BER doen dalen door fouten te corrigeren. Een goede FEC code zorgt voor een lage BER waarde terwijl er toch een hoog informatie ratio is. Dit wil zeggen dat veel informatie wordt verzonden met weinig redundantie, terwijl toch zoveel mogelijk fouten worden gecorrigeerd.

Lineaire codes

Wanneer codewoorden lineaire functies zijn van berichten, dan spreekt men van een *lineaire code*. Als berichten en codewoorden worden voorgesteld als vectoren van symbolen, dan leert de lineaire algebra [39] dat het encoderen en decoderen lineaire operaties zijn, die kunnen worden geschreven met behulp van matrices. De encoding operatie wordt genoteerd als:

$$t = mG, \tag{3.2}$$

waarbij G de *generator matrix* van de code wordt genoemd.

Blok code

Een *blok code* verwerkt elk blok van k symbolen — een bericht — afzonderlijk, en genereert hiervoor een codewoord. Er is geen verband met informatie uit een vorig of volgend bericht. Zoals zal blijken bij convolutionele codes (sectie 3.7.3) zijn er ook codes die wel rekening houden met andere berichten.

3.3 Repetition code

De repetition code is de meest eenvoudige manier om een bericht te beschermen tegen datacorruptie. Het idee bestaat eruit om een bericht meermaals te versturen. De duplicaten worden door de ontvanger bit per bit met elkaar vergeleken om voor elke bit te bepalen of deze 0 of 1 is: de waarde die het vaakst teruggevonden wordt in de verschillende duplicaten wint.

Een repetition code die drie duplicaten stuurt voor elk bericht wordt aangeduid met R_3. Voorbeeld 5 toont hoe een datasequentie via een repetition code kan worden verstuurd en hoe fouten kunnen worden gecorrigeerd. Enkel oneven aantallen duplicaten worden gebruikt, omdat, indien 50% van de duplicaten wordt beschadigd, bij even aantallen het aantal duplicaten dat voor een bepaalde bit waarde 0 heeft gelijk kan zijn aan het aantal duplicaten met een 1 voor die bit. De decoder kan dan niet beslissen welke waarde aangenomen moet worden. Dit kan niet met oneven aantallen.

Voorbeeld 5 *De bitsequentie 0010110 wordt met een R_3 repetition code beschermd tegen datacorruptie. m, t, n, r en $\hat{m}$ zijn gedefinieerd zoals in figuur 3.1. Elk bericht m is één bit lang. In het tweede codewoord wordt een bitfout geïntroduceerd, maar deze wordt door de decoder gecorrigeerd. Het vijfde bericht is onderhevig aan twee bitfouten. Dit kan niet meer gecorrigeerd worden door een R_3 code: er wordt foutief een waarde 0 toegekend aan de vijfde $\hat{m}$.*

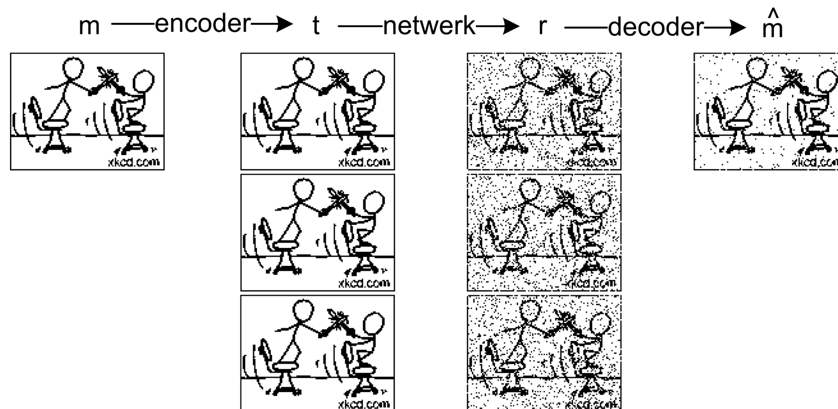
m	0	0	1	0	1	1	0
t	$\overbrace{000}$	$\overbrace{000}$	$\overbrace{111}$	$\overbrace{000}$	$\overbrace{111}$	$\overbrace{111}$	$\overbrace{000}$
n	000	001	000	000	101	000	000
r	$\underbrace{000}$	$\underbrace{001}$	$\underbrace{111}$	$\underbrace{000}$	$\underbrace{010}$	$\underbrace{111}$	$\underbrace{000}$
$\hat{\mathbf{m}}$	0	0	1	0	0	1	0

Repetition codes kunnen niet alle fouten detecteren: indien een bepaalde bit in meer dan 50% van de duplicaten werd beschadigd, wordt de verkeerde waarde aangenomen door de decoder. Het vijfde bericht in voorbeeld 5 toont dit. Des te meer duplicaten er worden gebruikt, zoveel te kleiner wordt de kans dat dit voorkomt, waardoor het BER daalt. Voor R_N kan worden aangetoond [33] dat:

$$\text{BER} = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}, \tag{3.3}$$

met N het aantal duplicaten in de repetition code en f de kans dat een bit beschadigd wordt door het netwerk, wat overeenkomt met het BER als geen FEC zou worden gebruikt. Het informatie ratio is echter klein: een R_N code heeft slechts een informatie ratio $R = \frac{1}{N}$.

Figuur 3.2 toont het versturen van een binaire afbeelding, beschermd door een R_3 code. Let op de grote hoeveelheid redundante data die moet worden verstuurd.



Figuur 3.2: *Het versturen van een afbeelding beschermd door een R_3 code. Het netwerk beschadigt 10% van het codewoord. Na het decoderen blijft 2.8% van de oorspronkelijke afbeelding beschadigd, zoals met vergelijking 3.3 kan worden berekend.*

Implementatie kan gebeuren op verschillende manieren: voor R_N kan elke bit N keer gedupliceerd worden. Dan spreken we van een (N,1) binaire

blok code. Het is ook mogelijk om groepen bits als blok te dupliceren. Elke grootte kan gebruikt worden voor de groepen. De keuze is afhankelijk van de toepassing en de eigenschappen van het netwerk. Een ($8N$,8) repetition code verstuurt bijvoorbeeld duplicaten per byte. Dit heeft voordelen wanneer bitfouten niet willekeurig voorkomen, maar opeenvolgende sequenties van bits worden beschadigd. Zo een groepering van fouten wordt een *error burst* genoemd. In echte netwerken is het realistischer om rekening te houden met error bursts dan enkel met willekeurige fouten [37]. Stel dat een R_3 code elke bit dupliceert (een (3,1) code), dan vernietigt elke error burst van drie of meer bits de informatie van minstens één bit in de oorspronkelijke datasequentie. Als elke byte wordt gedupliceerd (een (24,8) code), dan moet een error burst minstens negen bits lang zijn opdat informatie verloren zou gaan, of meerdere kortere bursts moeten in meer dan de helft van de duplicaten steeds dezelfde bits aantasten. De kans op een dergelijk scenario is veel kleiner dan de kans dat er zich één error burst van drie bits voordoet.

3.4 Hamming codes

De repetition codes uit de vorige sectie zijn weinig efficiënt: ze vereisen veel redundante data om een kleine hoeveelheid fouten te corrigeren. Richard W. Hamming was de eerste die een interessante code wist te ontwikkelen die verder ging dan het naïef kopiëren van bits. Met de Hamming codes werd het mogelijk om in een bericht één bitfout te corrigeren zonder dat dit bericht meermaals moest worden verzonden. [19, 33, 37, 38]

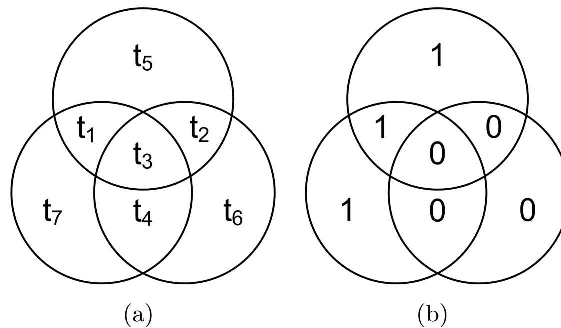
Om de concepten van de Hamming codes duidelijk te kunnen toelichten, zal eerst de (7,4) Hamming code worden besproken. Vervolgens wordt deze theorie gegeneraliseerd naar Hamming codes die berichten en codewoorden van andere lengtes gebruiken.

3.4.1 (7,4) Hamming code

De Hamming codes bouwen verder op het idee van de pariteitscontrole (sectie 2.3.1). Als een bericht kort genoeg is en er worden niet te veel fouten geïntroduceerd door het netwerk, zodat kan gesteld worden dat slechts één bit per bericht kan worden beschadigd, dan is pariteitscontrole voldoende om uit te zoeken of er een bit beschadigd is. Maar men weet niet welke bit werd beschadigd. Hamming codes maken het mogelijk om deze bit te identificeren. De (7,4) Hamming code gebruikt berichten van vier bits, maar in plaats van één pariteitsbit toe te voegen zoals bij de pariteitscontrole, voegt de code drie bits toe. Een codewoord is bijgevolg zeven bits lang.

De encoder moet voor elke pariteitsbit een waarde kiezen. Dit proces kan intuïtief worden voorgesteld met behulp van een Venn diagram, zoals weergegeven in figuur 3.3(a) [33]. In het diagram wordt de letter t gebruikt omdat hier het codewoord t wordt berekend. De vier bits van het bericht

worden op de plaatsen $t_1, ..., t_4$ ingevuld in het diagram. De drie pariteitsbits t_5, t_6, t_7 worden vervolgens gekozen door de regels van de even pariteit toe te passen in elke cirkel van het Venn diagram. De bitsequentie *1000* wordt geëncodeerd in voorbeeld 6.



Figuur 3.3: *De encoder van de (7,4) Hamming code weergegeven als een Venn diagram. (a) schematische weergave; (b) voorbeeld dat het bericht* 1000 *encodeert.*

Voorbeeld 6 *Het bericht* 1000 *moet worden geëncodeerd. De bitwaardes worden ingevuld in het Venn diagram. Om de waarde van t_5 te kennen, worden t_1, t_2 en t_3 beschouwd, omdat deze in dezelfde cirkel liggen als t_5. Om even pariteit te doen gelden binnen deze cirkel moet t_5 de waarde* 1 *krijgen. Analoog worden de waardes voor t_6 en t_7 bepaald. Deze zijn respectievelijk* 0 *en* 1. *Het codewoord voor dit bericht is bijgevolg* 1000101.

De Hamming code is een lineaire code. Volgens vergelijking 3.2 kan de code worden voorgesteld als een lineaire operatie, met

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.4}$$

als generator matrix [33]. De lezer kan eenvoudig controleren dat de vermenigvuldiging van [1 0 0 0] met G in hetzelfde codewoord resulteert als in voorbeeld 6. Berekeningen worden uiteraard *modulo 2* uitgevoerd. Merk op dat de Hamming code een systematische code is: de bits van het bericht komen voor in het codewoord. Dit is ook zichtbaar in de generator matrix. Die bestaat uit de identiteitsmatrix I_4 en een 3×4 matrix P. Er geldt:

$$G = \begin{bmatrix} I_4 & P \end{bmatrix} \tag{3.5}$$

I_4 zorgt ervoor dat de oorspronkelijke bits uit het bericht ook in het codewoord terecht komen. Met P worden de pariteitsbits berekend. Elke kolom

in P bevat precies één 0. Vermenigvuldiging van het bericht met een kolom resulteert bijgevolg in de pariteit voor drie van de vier bits uit het bericht. Elke kolom berekent de pariteit voor drie andere bits. Dit komt overeen met het berekenen van de pariteit in de cirkels van het Venn diagram.

Er moet worden opgemerkt dat de generator matrix G niet uniek is. Kolommen kunnen onderling worden omgewisseld. In deze situatie worden dezelfde bitwaardes berekend, maar staan de bits in een andere volgorde in het codewoord. Varianten die op deze manier bekomen werden, vormen nog nog steeds een (7,4) Hamming code. Zowel encoderen als decoderen verlopen identiek. Het enige verschil met de besproken code is de bitvolgorde in het codewoord.

Decoderen

Om een codewoord van de (7,4) Hamming code aan de ontvangstkant te decoderen zou een lijst kunnen worden gemaakt van alle mogelijke codewoorden. Het aantal codewoorden is beperkt: met vier bits als invoer kan de encoder slechts 16 correcte codewoorden genereren. Deze staan ter illustratie opgesomd in tabel 3.1. Het ontvangen codewoord, dat mogelijk beschadigd werd door het netwerk, moet vervolgens enkel worden vergeleken met alle mogelijkheden. Het codewoord waarvoor de Hamming afstand met de ontvangen bitsequentie het kleinst is, wordt geselecteerd als het correcte codewoord. Uiteraard bestaat de kans dat een verkeerd codewoord wordt geselecteerd omdat te veel bits werden beschadigd. De decoder faalt in dit geval in zijn taak en het originele bericht kan niet worden achterhaald.

Decoderen door het codewoord te kiezen dat de kleinste Hamming afstand heeft tot het ontvangen codewoord wordt *maximum likelihood (ML) decoding* genoemd. Voor een code die slechts 16 mogelijke codewoorden heeft, is deze aanpak mogelijk. Voor codes met een veel grotere verzameling codewoorden wordt ML decoding oninteressant omdat het vergelijken met alle codewoorden te veel rekenkracht vergt.

m	t	m	t	m	t	m	t
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

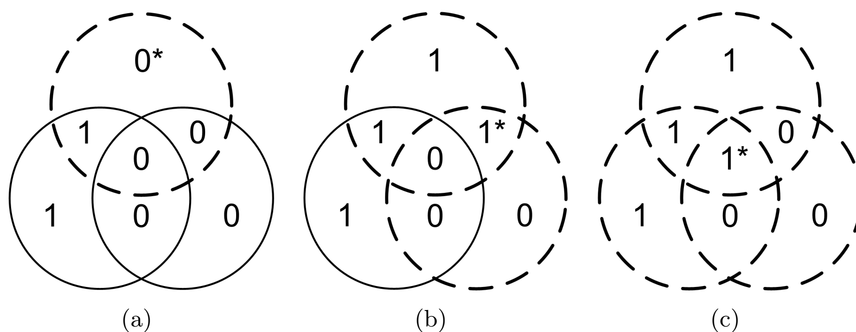
Tabel 3.1: *De 16 codewoorden t voor alle berichten m in de (7,4) Hamming code.*

Er bestaat een efficiëntere methode om de Hamming code te decoderen. Deze steunt op de observatie dat één bitfout de pariteit in een unieke collectie cirkels van het Venn diagram in figuur 3.3(a) doet falen. Er zijn drie mogelijkheden, die worden afgebeeld in figuur 3.4:

1. Indien t_5, t_6 of t_7 foutief is, klopt de pariteit in één cirkel niet. Voor elk van deze bits is dit een andere cirkel (figuur 3.4(a)).

2. Indien t_1, t_2 of t_4 foutief is, is de pariteit in twee cirkels incorrect. Voor elk van deze bits is dit een andere combinatie van twee cirkels (figuur 3.4(b)).

3. Enkel indien t_3 foutief is, geldt de pariteitsregel in geen enkele cirkel meer (figuur 3.4(c)).

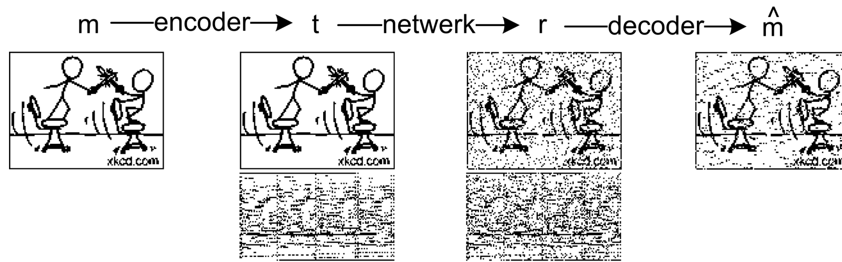


Figuur 3.4: *Voorstelling van enkele mogelijke combinaties van cirkels waarbinnen pariteit niet langer geldt door een bitfout in een ontvangen codewoord. De geëncodeerde bitsequentie is* 1000. *Bitfouten worden aangeduid met een asterisk. Cirkels waarbinnen de pariteit hierdoor niet langer geldt, worden aangeduid met een stippellijn.*

Uit deze observatie kan worden geconcludeerd dat het berekenen van in welke cirkels van een ontvangen codewoord de pariteit niet langer geldt voldoende informatie verschaft om te weten welke bit beschadigd werd tijdens transmissie.

In figuur 3.5 wordt een afbeelding verzonden die beschermd wordt met de (7,4) Hamming code. Als de code wordt vergeleken met de R_3 repetition code in figuur 3.2, dan kan worden geconcludeerd dat de Hamming code nog steeds 7% fouten niet corrigeert [33], wat aanzienlijk hoger ligt dan de 2.8% bitfouten die resteren in het beeld dat met de R_3 code werd beschermd. De intelligentie van de Hamming code zit in het veel hogere informatie ratio: de (7,4) Hamming code heeft een informatie ratio van 57%, wat aanzienlijk beter is dan het ratio van de R_3 code (33%). De Hamming code is dus in staat om met veel minder redundante data toch heel wat fouten te corrigeren.

Meer dan één bitfout in een codewoord kan door de Hamming code niet worden gecorrigeerd. Twee bitfouten resulteren in een ongeldige pariteit in cirkels van een andere bit. Daardoor zal de verkeerde bit worden gecorrigeerd, waardoor het decoderen zelfs een extra bitfout introduceert [33]. Een



Figuur 3.5: *Een afbeelding beschermd door de (7,4) Hamming code. De redundante data wordt in het codewoord gesplitst van het originele beeld voor de duidelijkheid. Het netwerk beschadigt 10% van het codewoord. Na het decoderen bevat de afbeelding nog 7% bitfouten.*

Syndroom z	000	001	010	011	100	101	110	111
Foute bit	—	r_7	r_6	r_4	r_5	r_1	r_2	r_3

Tabel 3.2: *De syndromen van de (7,4) Hamming code en welke bit in het ontvangen codewoord r ze als foutief aanduiden.*

gevolg hiervan is dat fouten in het gedecodeerde beeld gegroepeerd voorkomen. Dit is duidelijk te zien aan de bitfouten in figuur 3.5 die zichtbaar horizontale streepjes vormen.

3.4.2 Syndroom decoderen

Het decoderen door te bepalen in welke cirkels van figuur 3.3(a) de pariteit niet langer geldt, kan worden beschreven zonder Venn diagrammen. Hiervoor wordt elke cirkel van het diagram voorgesteld door een bit. De waarde van deze bit geeft aan of de pariteit binnen de cirkel geldig is. Drie bits vertegenwoordigen het hele diagram. Deze drie bits worden samen de *syndroomvector* genoemd, of korter, het *syndroom.* Een 1-bit in het syndroom betekent dat in de betreffende cirkel de pariteit niet geldt. Het syndroom bepaalt welke bit in het ontvangen codewoord moet worden gecorrigeerd om een transmissiefout weg te werken, net zoals dit bepaald wordt door de combinatie van cirkels waarbinnen pariteit niet klopt. Voor de (7,4) Hamming code worden de waardes voor het syndroom samengevat in tabel 3.2. Het syndroom wordt aangeduid met de letter z. Als het syndroom gelijk is aan *000*, is het codewoord correct ontvangen (pariteit is correct in elke cirkel). Er is geen transmissiefout opgetreden.

Voorbeeld 7 *De bits van het syndroom komen als volgt overeen met de cirkels van het Venn diagram:* z_1 *is de bovenste cirkel,* z_2 *de rechtse en* z_3 *de linkse. Veronderstel dat het codewoord* $t = 1000101$ *— dit is het codewoord voor bericht* 1000 *— wordt beschadigd, en* $r = 1\mathbf{1}100101$ *wordt ontvangen. Het Venn diagram voor deze situatie wordt weergegeven in figuur 3.4(b).*

In zowel de bovenste als de rechtse cirkel geldt de pariteit niet meer. Het syndroom is bijgevolg 110*. Uit tabel 3.2 weet men dat dit betekent dat bit* r_2 *fout is. Inderdaad, als deze bit wordt omgewisseld, wordt de bitfout hersteld. In het Venn diagram is de pariteit weer in orde in alle cirkels.*

Het concept van syndroom decoderen, waarbij uit een codewoord een syndroom wordt berekend dat vervolgens aangeeft wat er moet gebeuren, keert terug in het decoderen van andere codes. In feite kan ook de foutdetectie met behulp van de controlesom (sectie 2.3.2) en de CRC (sectie 2.3.3) aanschouwd worden als syndroom decoderen: de ontvanger berekent een syndroom en beslist dat de ontvangen bitsequentie correct is als en slechts als het syndroom de waarde 0 heeft.

Het syndroom decoderen van de Hamming code kan net als het encoderen worden voorgesteld als een lineaire operatie:

$$z = rH^T, \tag{3.6}$$

waarbij H de pariteitscontrole matrix wordt genoemd. H wordt zo gekozen zodat

$$GH^T = 0. \tag{3.7}$$

Voor de (7,4) Hamming code met als generator matrix G uit vergelijking 3.4 is deze matrix

$$H = [P^T I_3] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{3.8}$$

waarbij P gelijk is aan de matrix P in vergelijking 3.5 en I_3 de 3×3 identiteitsmatrix is. Merk op dat elke kolom van H een syndroom is.

Met deze definitie van H en rekening houdend met vergelijking 3.2 resulteert het berekenen van het syndroom voor een correct codewoord t in 0:

$$z = tH^T = mGH^T = m0 = 0 \tag{3.9}$$

Het berekenen van het syndroom voor een beschadigd codewoord r is enkel afhankelijk van de geïntroduceerde fouten n. Dit kan worden geconcludeerd uit vergelijkingen 3.1, 3.6 en 3.9:

$$z = rH^T = (t+n)H^T = tH^T + nH^T = nH^T \tag{3.10}$$

Aangezien de Hamming code slechts één bitfout kan corrigeren, kunnen we bij een correcte decodering stellen dat n slechts één bit heeft die 1 is. Bijgevolg selecteert de vermenigvuldiging van n met H^T één van de kolommen uit H^T, die zoals eerder vermeld de mogelijke syndromen zijn. Bovendien is het nummer van de geselecteerde kolom tevens het nummer van de bit die moet worden gecorrigeerd.

3.4.3 Generalisatie

De observaties die werden gemaakt bij het encoderen en decoderen van de (7,4) Hamming code kunnen worden gegeneraliseerd naar elke andere (n,k) lineaire blok code. Zoals vermeld, kan de encoder van elke lineaire code worden voorgesteld zoals in vergelijking 3.2. Bij elke $k \times n$ generator matrix G hoort een $(n - k) \times n$ pariteitscontrole matrix H die voldoet aan vergelijking 3.7, zodat ook vergelijking 3.9 geldt. Met vergelijking 3.6 kan vervolgens een syndroom worden berekend dat aangeeft hoe een mogelijk incorrect ontvangen codewoord moet worden gecorrigeerd.

Ook moet worden opgemerkt dat bij de (7,4) Hamming code de syndromen — en dus ook de kolommen van H — binaire representaties zijn van de getallen 1 tot 7. Hiermee kan een meer algemene definitie voor Hamming codes worden gegeven. Voor elk positief geheel getal $p >= 2$ bestaat er een (n,k) Hamming code, waarbij $n = 2^p - 1$ en $k = 2^p - p - 1$, en waarbij de kolommen van pariteitscontrole matrix H binaire representaties van de getallen 1 tot n zijn [38]. Deze codes hebben echter geen extra correctie capaciteit. Een (15,11) Hamming code (met $p = 4$) zal net als de (7,4) code slechts één bitfout in het codewoord kunnen herstellen. De code heeft wel een groter informatie ratio.

3.5 Gegarandeerde foutcorrectie

Naast de ontdekking van de Hamming codes uit de vorige sectie, wist Richard Hamming twee belangrijke uitspraken te doen over de mogelijkheden van eender welke foutcorrectie techniek. Eerst wordt het concept *minimale afstand* (aangeduid met het symbool d_{min}) van een code gedefinieerd. De minimale afstand van een code is de kleinste Hamming afstand tussen eender welk paar codewoorden die door de encoder van die code kunnen worden gegenereerd. Als men terugkijkt naar de codewoorden van de (7,4) Hamming code in tabel 3.2, dan blijkt dat $d_{min} = 3$: elk paar codewoorden verschilt minstens in drie bits. Hamming besloot het volgende [37, 38]:

Gegarandeerde foutcorrectie capaciteit Een code kan in een bericht alle bitfouten corrigeren, zolang het aantal fouten kleiner of gelijk is aan $(d_{min} - 1)/2$.

Gegarandeerde foutdetectie capaciteit Een code kan twee keer meer fouten detecteren dan corrigeren: tot maximaal $d_{min} - 1$ fouten kunnen worden gedetecteerd. Uiteraard weet de decoder in dit geval niet waar de fouten zich bevinden in het codewoord, anders zou correctie mogelijk zijn.

Dit besluit kan worden toegepast op de (7,4) Hamming code. Uit de uitleg over de Hamming code blijkt dat de decoder een beschadigd bericht

herstelt zolang er slechts één bitfout voorkomt. Dit is inderdaad de gegarandeerde correctie capaciteit van de Hamming code. Indien er twee bitfouten voorkomen, zal de Hamming decoder nog steeds een syndroom berekenen dat niet 0 is. Echter, zoals bleek uit de uiteenzetting, kan de code de bits niet herstellen, omdat het syndroom een verkeerde bit aanwijst als foutief. Toch kan de Hamming code het ontvangen codewoord als foutief markeren indien er twee bitfouten voorkomen. Dit is wat de gegarandeerde foutdetectie capaciteit aangeeft. Vanaf drie bitfouten kan het ontvangen codewoord met een ander correct codewoord overlappen. Hierdoor kan detectie van drie of meer bitfouten niet meer worden gegarandeerd.

Ook voor alle andere Hamming codes geldt dat $d_{min} = 3$. Door de eigenschap dat Hamming codes één bitfout kunnen corrigeren en er twee kunnen detecteren, behoren ze tot een groep codes die de SECDED (Single Error Correction Double Error Detection) codes worden genoemd.

3.6 Reed-Solomon codes

Reed-Solomon (RS) codes [37, 53] zijn een stuk geavanceerder dan Hamming codes. Ze werden in 1960 geïntroduceerd door Irving Reed en Gus Solomon in [46]. RS codes zijn in staat om meer dan één transmissiefout te corrigeren. Ze vallen onder de groep van *cyclische* codes.

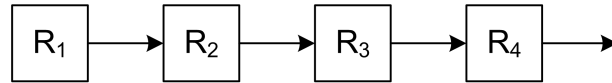
Cyclische codes zijn lineaire codes die, vanwege hun wiskundige opbouw, speciaal ontwikkeld zijn om efficiënte encodering en decodering mogelijk te maken. Ze voldoen verder aan de volgende eigenschap: indien een (n,k) code een geldig codewoord $t = (t_0, t_1, \ldots, t_{n-1})$ heeft, dan is ook een rechtse cyclische shift $t' = (t_{n-1}, t_0, \ldots, t_{n-2})$ een geldig codewoord [37].

Om RS codes te kunnen begrijpen moeten enkele begrippen besproken worden. De volgende twee subsecties geven een uiteenzetting over Linear Feedback Shift Registers en Galois velden. Deze vormen de fundamenten voor niet-binaire cyclische codes, een categorie waartoe ook RS codes behoren.

3.6.1 Linear Feedback Shift Registers

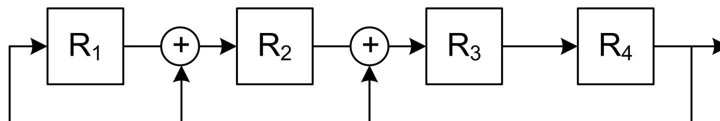
Shift registers [26] kennen we als digitale hardware circuits. Een shift register is een reeks geheugenelementen, registers genaamd, die in serie met elkaar verbonden zijn. Elk register heeft steeds een binaire status 0 of 1. Bij elke tik van een klok schuift de waarde van een register naar het aangrenzende register. Afhankelijk van de shift richting wordt een waarde doorgegeven aan de linker- of rechterbuur. De zeer eenvoudige shift registers worden onder andere gebruikt om parallelle data te serialiseren. In de eerste kloktik wordt parallelle data in de registers geladen. De volgende kloktiks wordt telkens het laatste register uitgelezen en wordt een shift uitgevoerd. Figuur 3.6

toont een shift register met vier geheugenelementen waarmee een dergelijke operatie kan worden uitgevoerd.



Figuur 3.6: *Een shift register met vier registers. De registers worden naar rechts geshift. De uitvoer wordt opgevangen aan het meest rechtse register.*

In shift registers is de invoer van een register steeds de uitvoer van een vorig register in de ketting. Feedback shift registers bevatten *taps*. Met taps wordt de uitvoer van een register naar een andere plaats in het schema geleid om zich te vermengen met de invoer van een ander register. Indien bij elke kloktik de nieuwe waardes voor de registers een lineaire functie zijn van de vorige waardes, spreekt men van een Lineair Feedback Shift Register, of kortweg LFSR [26, 37]. Figuur 3.7 toont het schema van een LFSR en in voorbeeld 8 worden de verschillende stages voor dit LFSR overlopen. LFSR's worden onder andere gebruikt voor het genereren van pseudo-random bitpatronen [26] en in de hardware implementatie van de veeltermdeling voor CRC's uit sectie 2.3.3 [62]. Zoals zal blijken, kennen LFSR's een gelijkaardige toepassing bij de encoder van RS codes.



Figuur 3.7: *Een LFSR met vier registers en een tap op de uitvoer van het laatste register R_4. De $\oplus$-operators werken als XOR poorten. Door de tap wordt de uitvoer van R_4 als invoer voor R_1 gebruikt. Daarnaast beïnvloedt de uitvoer van R_4 de invoer van registers R_2 en R_3.*

Voorbeeld 8 *Dit voorbeeld overloopt de stages die voorkomen in het LFSR van figuur 3.7. Als het LFSR initieel de sequentie* 1000 *bevat, dan toont tabel 3.3 de status van de registers na elke kloktik. Bij kloktik 1 verschuift de enige 1-bit van register R_1 naar R_2. Als register R_4 een 1-bit bevat, dan zal bij de volgende tik een 1-bit worden ingevoerd in R_1. Ook de invoer van R_2 en R_3 wordt in dit geval beïnvloed met behulp van de XOR poorten. Bij kloktik 6 krijgt R_3 de waarde 0 omdat na kloktik 5 zowel R_2 als R_4 de waarde 1 bevatten. Opmerkelijk is dat dit LFSR na zeven kloktiks opnieuw zijn initiële sequentie bevat en bijgevolg steeds door dezelfde stages blijft lopen.*

3.6.2 Galois velden

In sectie 2.3.3 werden Galois velden al eens aangehaald. Daar werd uitgelegd dat $GF(2)$ een veld aanduidt met twee symbolen, namelijk 0 en 1, en

Klok	Status R_1	R_2	R_3	R_4	Uitvoer
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	1	1	1
6	1	1	0	1	1
7	1	0	0	0	0

Tabel 3.3: *De status van het LFSR uit figuur 3.7 bij elke kloktik.*

dat deze symbolen kunnen worden opgeteld met een *modulo 2* optelling en vermenigvuldigd zoals met de decimale getallen 0 en 1. Optelling en vermenigvuldiging worden samengevat in tabel 3.4. Een Galois veld wordt ook een *eindig* veld genoemd, omdat de optelling en vermenigvuldiging van twee symbolen uit het veld steeds resulteren in een symbool dat ook tot het veld behoort.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Tabel 3.4: *Optelling (links) en vermenigvuldiging (rechts) in* $GF(2)$.

Galois velden worden toegelicht in [8, 37, 53]. Algemeen kan gesteld worden dat een Galois veld $GF(p)$ bestaat uit p symbolen. p moet een priemgetal zijn. Op elk Galois veld wordt een optelling en vermenigvuldiging gedefinieerd. Dit gebeurt zo dat het resultaat van deze operaties steeds een symbool van het veld is. Galois velden kunnen worden uitgebreid met extra symbolen. Deze uitbreidingen heten *extensievelden*. Een extensieveld, aangeduid met $GF(p^m)$, bevat p^m symbolen, waarbij m een positief geheel getal moet zijn. De symbolen van het oorspronkelijke Galois veld $GF(p)$ dat werd uitgebreid, zijn een subset van de symbolen van $GF(p^m)$. RS codes zijn niet-binaire codes. Zoals vermeld in sectie 3.2 betekent dit dat de berekeningen niet gebeuren met de symbolen 0 en 1, maar met een grotere verzameling symbolen die gevormd wordt door bits te groeperen. RS codes gebruiken de symbolen uit het Galois veld $GF(2^m)$, waarbij m het aantal bits aanduidt dat per symbool wordt gebruikt.

Definitie van symbolen

Als een Galois veld wordt uitgebreid, moeten de nieuwe symbolen een naam krijgen. In de theorie rond Galois velden kan elk symbool aangeduid worden

door een macht van α. Beginnend bij $GF(2)$ wordt een *oneindige* verzameling van symbolen gedefinieerd door

$$\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^j, \ldots\}$$

of ook

$$\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \ldots, \alpha^j, \ldots\}$$

omdat de 0-de macht van een getal steeds gelijk is aan 1.

Om een *eindig* veld $GF(2^m)$ met 2^m symbolen te genereren uit deze oneindige verzameling moet er een conditie worden toegevoegd die het veld sluit. Dit kan door:

$$\alpha^{(2^m-1)} = 1.$$

Deze definitie zorgt ervoor dat elk symbool met een macht groter of gelijk aan $2^m - 1$ gereduceerd kan worden tot een symbool met een kleinere macht. Rekenen met machten leert ons dat

$$\alpha^{(2^m+n)} = \alpha^{(2^m-1)}\alpha^{(n+1)} = \alpha^{(n+1)}.$$

Het veld is nu eindig gedefinieerd. De symbolen voor $GF(2^m)$ zijn de verzameling

$$GF(2^m) = \left\{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{(2^m-2)}\right\}. \tag{3.11}$$

Operaties

Elk symbool in $GF(2^m)$ bestaat uit m bits. Elke α^i heeft bijgevolg een binaire representatie. Deze bitsequentie kan worden voorgesteld als een veelterm zoals ook eerder werd gedaan bij de CRC's. In het geval van $GF(2^3)$ is α^i drie bits lang en wordt als veelterm $a_i(x)$ voorgesteld als

$$\alpha^i = a_i(x) = a_{i,0} + a_{i,1}x + a_{i,2}x^2, \tag{3.12}$$

waarbij de coëfficiënt $a_{i,j}$ de waarde van bit j van de bitsequentie voor symbool α^i aanneemt. De bitwaardes voor de symbolen zijn zo gekozen dat in $GF(2^m)$ de som kan worden berekend als

$$\alpha^i + \alpha^j = (a_{i,0} + a_{j,0}) + (a_{i,1} + a_{j,1})x + \ldots + (a_{i,m-1} + a_{j,m-1})x^{m-1}$$

en dat de vermenigvuldiging kan worden berekend als

$$\alpha^i * \alpha^j = \alpha^{((i+j)\%(2^m-1))},$$

waarbij de *modulo* $(2^m - 1)$ ervoor zorgt dat de vermenigvuldiging steeds resulteert in een symbool binnen het eindige veld (zie ook vergelijking 3.11).

Bijlage A vat $GF(2^3)$ samen. De bijlage bevat de optellings- en vermenigvuldigingstabel voor dit veld en een lijst van de symbolen met hun bijhorende bitsequenties.

3.6.3 RS encoder

RS codes zijn niet-binair. Eerder werd vermeld dat de code groepen bits als één symbool behandelt. Het aantal bits dat gebruikt wordt per symbool wordt aangeduid met de letter m. Er is een ruime keuze in (n,k) RS codes. Als beperking geldt dat [53]

$$m > 2 \qquad \text{en} \qquad 0 < k < n < 2^m + 2. \tag{3.13}$$

Verder moet gelden dat

$$(n, k) = (2^m - 1, 2^m - 1 - 2t). \tag{3.14}$$

t is het aantal symbolen dat de RS code kan corrigeren als het codewoord beschadigd geraakt.

Het aantal bits m per symbool bepaalt tevens welke symbolen worden gebruikt, namelijk die van $GF(2^m)$. Een voorbeeld: een (7,3) RS code heeft symbolen van drie bits lang. De code encodeert berichten van drie symbolen (negen bits in totaal) tot codewoorden van zeven symbolen (21 bits). t is gelijk aan 2, dus tot twee symbolen kunnen worden gecorrigeerd na ontvangst met transmissiefouten.

Het encoderen van een RS code heeft veel gelijkenissen met het berekenen van een CRC. Opnieuw wordt gebruik gemaakt van een generator veelterm $g(x)$. De bericht veelterm $m(x)$ die moet worden beschermd, bestaat uit k symbolen. $m(x)$ wordt eerst uitgebreid met $n - k$ 0-symbolen en vervolgens gedeeld door de generator veelterm. Het uitbreiden wordt wiskundig voorgesteld door $m(x)$ te vermenigvuldigen met $x^{(n-k)}$. Het codewoord $t(x)$ wordt gevormd door de rest van de deling $p(x)$ bij $m(x)x^{(n-k)}$ op te tellen. Samengevat:

$$p(x) = m(x)x^{(n-k)} \,\%\, g(x) \tag{3.15}$$

$$t(x) = m(x)x^{(n-k)} + p(x) \tag{3.16}$$

Het grote verschil met de berekeningen van de CRC is dat in dit geval niet met binaire symbolen wordt gerekend maar met de symbolen uit het Galois veld.

De generator veelterm heeft de vorm

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{2t-1} x^{2t-1} + x^{2t}, \tag{3.17}$$

waarbij t zoals eerder gedefinieerd het aantal symboolfouten is dat kan worden gecorrigeerd. De generator veelterm moet zo worden opgebouwd dat $2t$ opeenvolgende machten van α wortels zijn van de veelterm. Voor de (7,3) RS code met $t = 2$ gebeurt dit (rekening houdend met de operaties in

$GF(2^3)$, terug te vinden in tabellen A.2 en A.3) als volgt:

$$\begin{aligned} g(x) &= (x-\alpha)\left(x-\alpha^2\right)\left(x-\alpha^3\right)\left(x-\alpha^4\right) \\ &= \left(x^2-(\alpha+\alpha^2)x+\alpha^3\right)\left(x^2-(\alpha^3+\alpha^4)x+\alpha^0\right) \\ &= \left(x^2-\alpha^4x+\alpha^3\right)\left(x^2-\alpha^6x+\alpha^0\right) \\ &= x^4-\left(\alpha^4+\alpha^6\right)x^3+\left(\alpha^3+\alpha^3+\alpha^0\right)x^2-\left(\alpha^4+\alpha^2\right)x+\alpha^3 \\ &= x^4-\alpha^3x^3+\alpha^0x^2-\alpha^1x+\alpha^3 \end{aligned}$$

Omdat $GF(2^3)$ een uitbreiding is van $GF(2)$ en omdat in dit basisveld geldt dat $+1 = -1$ ([53]), kan de generator veelterm voor de (7,3) RS code nu geschreven worden als

$$g(x) = \alpha^3 + \alpha^1x + \alpha^0x^2 + \alpha^3x^3 + x^4. \tag{3.18}$$

De α-machten zijn de waardes voor $g_0, ..., g_3$ uit vergelijking 3.17.

3.6.4 Encoderen met een LFSR

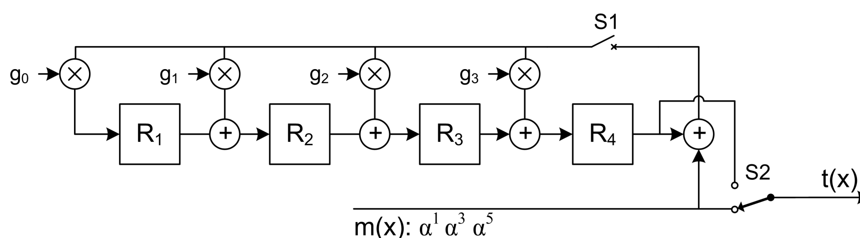
Met bovenstaande theorie omtrent veeltermdeling uit vergelijking 3.15 en de opbouw van de generator veelterm kan met vergelijking 3.16 een codewoord berekend worden en is de RS encoder gedefinieerd. Met behulp van een LFSR (zie ook sectie 3.6.1) kan de berekening zeer efficiënt worden afgehandeld. Voor een (n,k) RS code kan in n kloktiks een codewoord worden gegenereerd. Dit vereist een LFSR circuit met $n-k$ geheugenelementen.

Figuur 3.8 toont het LFSR dat gebruikt wordt voor de (7,3) RS code. Dit schema bevat twee soorten operators. De $\oplus$-operator telt de twee symbolen aan zijn invoerpoorten op. De $\otimes$-operator vermenigvuldigt deze symbolen. De coëfficiënten van de generator veelterm $g(x)$ worden in stijgende volgorde aan de $\otimes$-operators gekoppeld zoals te zien is in de figuur. Het encoderen van een bericht verloopt in twee delen:

1. Het eerste deel bestaat uit de eerste k kloktiks. Schakelaar $S1$ is gesloten en schakelaar $S2$ staat omlaag. Tijdens deze kloktiks worden de symbolen uit het bericht $m(x)$ één voor één in het LFSR geladen. Tegelijkertijd worden ze gekopiëerd naar de uitvoer, om zo de eerste k symbolen van het codewoord $t(x)$ te vormen.

2. Tijdens de laatste $n-k$ kloktiks wordt $S1$ geopend zodat er geen feedback meer is in het LFSR. De waardes die de registers nu hebben zijn de symbolen die de rest van de veeltermdeling $p(x)$ uit vergelijking 3.15 vormen. Door $S2$ omhoog te plaatsen worden deze symbolen in de laatste $n-k$ kloktiks uit het LFSR gehaald en toegevoegd aan de uitvoer. Het codewoord $t(x)$ is nu compleet.

Voorbeeld 9 overloopt stap voor stap hoe een bericht wordt geëncodeerd met behulp van het LFSR.



Figuur 3.8: *Het LFSR voor een (7,3) RS encoder.*

Voorbeeld 9 *De bitsequentie* 010 110 111 *wordt geëncodeerd met een (7,3) RS code. Deze code maakt gebruik van symbolen met een lengte van drie bits. Berekeningen gebeuren bijgevolg in $GF(2^3)$. De bitsequentie wordt voorgesteld als een symboolsequentie door de symbolen $\alpha^1\,\alpha^3\,\alpha^5$ (zie ook tabel A.1). De berichtvector $m(x)$ is bijgevolg*

$$m(x) = \alpha^1 + \alpha^3 x + \alpha^5 x^2$$

en moet worden gedeeld door $g(x)$ die terug te vinden is in vergelijking 3.18.

De coëfficiënten van $m(x)$ worden tijdens de eerste drie kloktiks in het LFSR van figuur 3.8 geladen. De waardes van de registers voor elke tik staan vermeld in tabel 3.5. Na drie tiks bevatten de registers de coëfficiënten van de restterm $p(x)$, dewelke in de volgende vier tiks uit de registers worden geshift en toegevoegd worden aan de uitvoer. De uitvoer bevat nu zeven symbolen. Deze symbolen vormen de coëfficiënten van de codewoord veelterm $t(x)$:

$$t(x) = \alpha^0 + \alpha^2 x + \alpha^4 x^2 + \alpha^6 x^3 + \alpha^1 x^4 + \alpha^3 x^5 + \alpha^5 x^6$$

Op bitniveau is dit het codewoord 100 001 011 101 010 110 111.

Invoer			Kloktik	R_1	R_2	R_3	R_4	Feedback
α^1	α^3	α^5	0	0	0	0	0	α^5
	α^1	α^3	1	α^1	α^6	α^5	α^1	α^0
		α^1	2	α^3	0	α^2	α^2	α^4
		—	3	α^0	α^2	α^4	α^6	—

Tabel 3.5: *De status van het LFSR tijdens de eerste $k = 3$ kloktiks bij het encoderen van het bericht in voorbeeld 9.*

3.6.5 RS decoder

De RS decoder werkt met syndroom decoding. Uit het ontvangen codewoord $r(x)$ wordt een syndroom berekend. Uit dit syndroom wordt vervolgens afgeleid welke symbolen in $r(x)$ incorrect werden ontvangen en wat de correcte waardes moeten zijn.

Stel dat het codewoord correct wordt ontvangen en dus volledig gelijk is aan $t(x)$. Uit vergelijkingen 3.15 en 3.16 is geweten dat $t(x)$ deelbaar is door $g(x)$. Bijgevolg zijn de wortels van $g(x)$ ook wortels van $t(x)$. Voor de (7,3) RS code met $g(x)$ zoals in vergelijking 3.18 geldt dus dat

$$t(\alpha) = t(\alpha^2) = t(\alpha^3) = t(\alpha^4) = 0$$

want het evalueren van een functie naar zijn wortels resulteert in 0. Indien $r(x)$ echter verschilt van $t(x)$ en de wortels worden ingevuld in $r(x)$, dan zal het resultaat niet meer voor elke wortel 0 zijn. Elk van deze resultaten vormt een coëfficiënt van de syndroom vector $s(x)$:

$$s_i = r(\alpha^i) \qquad \text{met}\, i = 1, \ldots, n-k$$

Een codewoord $t(x)$ wordt beschadigd door er een foutveelterm $n(x)$ bij op te tellen. Deze veelterm heeft de waarde 0 voor alle coëfficiënten, behalve voor de coëfficiënten die horen bij de graad waarin $t(x)$ beschadigd is. Een voorbeeld van een $n(x)$ die een codewoord van een (7,3) RS code beschadigt:

$$n(x) = 0 + 0x + 0x^2 + \alpha^2 x^3 + \alpha^5 x^4 + 0x^5 + 0x^6.$$

Deze foutveelterm beschadigt twee symbolen van $t(x)$. Eén wordt beschadigd door er α^2 bij op te tellen. Binair wordt α^2 voorgesteld met de bits 001. Er wordt dus één bit van het symbool van $t(x)$ beschadigd. Het andere wordt beschadigd door $\alpha^5 = 111$. Elke bit wordt dus beschadigd.

Het invullen van de wortels van $g(x)$ in $r(x)$ om de syndromen te berekenen kan worden gezien als het invullen van de wortels in de som $t(x)+n(x)$. Het invullen van de wortels in $t(x)$ resulteert in 0 dus komt het berekenen van de syndromen overeen met het invullen van de wortels in $n(x)$, of ook

$$s_i = n(\alpha^i) \qquad \text{met}\, i = 1, \ldots, n-k$$

De foutveelterm $n(x)$ is uiteraard de grote onbekende in het verhaal. Stel dat er ν fouten worden geïntroduceerd door $n(x)$, dan heeft deze veelterm ν niet-0 coëfficiënten. De foutveelterm kan dan korter worden genoteerd als

$$n(x) = n_{j_1} x^{j_1} + n_{j_2} x^{j_2} + \ldots + n_{j_\nu} x^{j_\nu}, \tag{3.19}$$

waarbij $l = 1, 2, ..., \nu$ de fouten indexeert, j_l de plaats aanduidt waar de fout voorkomt, en n_{j_l} de het symbool is dat de fout introduceert op die plaats.

Omdat het syndroom kan worden berekend uit $n(x)$, kan het ook worden berekend uit deze nieuwe definitie voor $n(x)$. Het invullen van α in vergelijking 3.19 resulteert in onbekende machten van α, namelijk α^{j_l}. Als alle α^{j_l} bekend zouden zijn, dan zouden ook alle j_l bekend zijn, en bijgevolg de positie van de fouten. We herschrijven de berekening van de coëfficiënten voor het syndroom $s(x)$ met de nieuwe definitie voor $n(x)$. De onbekende α^{j_l}

machten worden als variable β_l aangeduid. Zoals geweten zijn er $n-k=2t$ coëfficiënten voor $s(x)$. Dit resulteert in de volgende $2t$ vergelijkingen:

$$\begin{aligned} s_1 = n(\alpha) &= n_{j_1}\beta_1 + n_{j_2}\beta_2 + \ldots + n_{j_\nu}\beta_\nu \\ s_2 = n(\alpha^2) &= n_{j_1}{\beta_1}^2 + n_{j_2}{\beta_2}^2 + \ldots + n_{j_\nu}{\beta_\nu}^2 \\ &\ldots \\ s_{2t} = n(\alpha^{2t}) &= n_{j_1}{\beta_1}^{2t} + n_{j_2}{\beta_2}^{2t} + \ldots + n_{j_\nu}{\beta_\nu}^{2t} \end{aligned} \quad (3.20)$$

Gegeven dat er maximaal t fouten kunnen worden gecorrigeerd, zijn er maximaal $2t$ onbekenden (t foutwaardes n_{j_l} en t foutposities β_l) in deze $2t$ vergelijkingen. De vergelijkingen vormen bijgevolg een oplosbaar stelsel. Elke techniek die dit stelsel oplost, wordt een Reed-Solomon decoderingsalgoritme genoemd. [37] bespreekt enkele van deze algoritmes, waaronder de algoritmes van Peterson en van Berlekamp-Massey.

De algoritmes gaan ervan uit dat er maximaal t fouten zijn. Indien er meer dan t fouten in een codewoord voorkomen, resulteert het oplossen van het stelsel in foutieve waardes voor n_{j_l} en β_l. De verkeerde symbolen worden als foutief aanzien en "gecorrigeerd" met een foutieve waarde. Dit is analoog met wat er gebeurt bij Hamming codes als een codewoord meer dan één bitfout bevat: ook hier zal het berekende syndroom de verkeerde bit als foutief aanduiden waardoor de decoder faalt in zijn poging om het originele bericht te berekenen.

3.6.6 Eigenschappen van RS codes

RS codes zijn lineair. Volgens vergelijking 3.2 kan de encoder ook met een generator matrix opgebouwd worden. Inderdaad, een RS code met een generator veelterm zoals gedefinieerd in vergelijking 3.17 heeft een equivalente matrix van de vorm [24]:

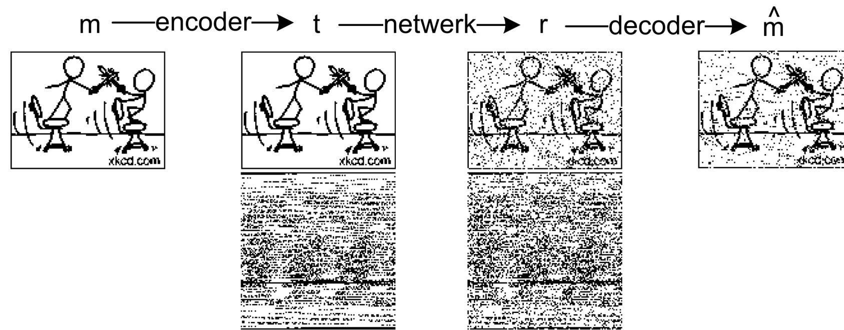
$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{2t-1} & 1 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{2t-2} & g_{2t-1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & g_0 & \cdots & \cdots & g_{2t-2} & g_{2t-1} & 1 \end{bmatrix}$$

De implementatie van een encoder met deze matrix vraagt echter meer rekenkracht dan de implementatie met een LFSR beschreven in sectie 3.6.4.

Eén van de belangrijkste eigenschappen van RS codes is dat ze de grootst mogelijke minimale afstand van alle lineaire codes hebben. Voor elke (n,k) RS code geldt dat

$$d_{min} = n - k + 1.$$

Dit wordt bewezen in [37]. Met de gegarandeerde foutcorrectie capaciteit van Hamming (sectie 3.5) in het achterhoofd is de RS code in staat om



Figuur 3.9: *Een afbeelding geëncodeerd met de (7,3) RS code wordt verstuurd over een netwerk dat 10% willekeurige fouten introduceert. De decoder weet het aantal bitfouten tot ongeveer 6% te reduceren.*

met een bepaalde hoeveelheid redundantie het meest aantal beschadigde symbolen te herstellen.

In figuur 3.9 wordt de ondertussen vertrouwde afbeelding geëncodeerd met een (7,3) RS code met symbolen die drie bits lang zijn. Vervolgens wordt het codewoord beschadigd met 10% willekeurige fouten. Wat opvalt is dat de (7,3) RS code, die in vergelijking met de (7,4) Hamming code behoorlijk wat complexer is en daarbij nog eens veel meer redundante bits gebruikt, slechts weinig extra fouten kan corrigeren ten opzichte van de Hamming code. Dit vereist een verklaring.

RS codes werken op niet-binaire symbolen. In figuur 3.9 werd een code gebruikt waarbij een symbool drie bits lang is. Voor de decoder maakt het niet uit of er één of drie bits van een symbool beschadigd zijn. De decoder zal bij het oplossen van het stelsel in vergelijking 3.20 altijd een nieuwe waarde berekenen voor alle bits van het beschadigde symbool. De fouten die door het netwerk worden geïntroduceerd in figuur 3.9 zijn willekeurige bitfouten: elke bit heeft 10% kans om beschadigd te worden. De kans dat een symbool met drie bits beschadigd wordt, is bijgevolg 30%! Gemiddeld worden in een codewoord meer dan twee van de zeven symbolen beschadigd. Dit is gemiddeld meer dan de code kan corrigeren.

De kans dat een RS code een symbool niet corrigeert, kan worden berekend met [53]

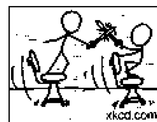
$$P_E \approx \frac{1}{2^m - 1} \sum_{j=t+1}^{2^m-1} j \binom{2^m - 1}{j} p^j (1-p)^{2^m-1-j}$$

waarbij p de kans is dat een symbool beschadigd wordt, t het aantal corrigeerbare symbolen is en symbolen bestaan uit m bits. Met de waardes uit figuur 3.9 ($p = 30\%$; $t = 2$; $m = 3$) betekent dit dat na het decoderen nog steeds 17% van de symbolen beschadigd is. Uiteraard moeten niet alle bits in elk foutief symbool ook foutief zijn. Zoals eerder vermeld is de kans op

een symboolfout 3× groter dan een bitfout. Als het aantal symboolfouten van 30% tot 17% gereduceerd wordt, daalt het aantal bitfouten van 10% naar ongeveer 6%, wat overeenkomt met wat wordt geconcludeerd in het experiment in figuur 3.9.

Drie bitfouten in een codewoord van zeven symbolen, 21 bits in totaal, volstaan om de (7,3) decoder te doen falen, want ze kunnen meer dan twee symbolen beschadigen. Wat maakt RS codes dan zo interessant? De codes zijn wel efficiënt van zodra bitfouten in groep voorkomen. Error bursts werden eerder al eens aangehaald bij de bespreking van repetition codes (sectie 3.3). De RS decoder maakt geen onderscheid tussen verschillende symbolen van het codewoord. Zolang er minder dan t foutieve symbolen zijn, met $2t = n - k$, zal de code de symbolen weten te herstellen. In het ideale geval waarbij alle bits van exact twee symbolen worden beschadigd, kunnen tot zes bits worden hersteld. Dit is 28% van het codewoord.

Om de kracht van de RS code te bewijzen wordt het experiment in figuur 3.9 aangepast. Na het beschadigen van 10% van de bits wordt het experiment zo gemanipuleerd dat drie van de zeven symbolen in elk codewoord toch correct zijn. Omdat bitfouten in $\frac{3}{7}$de van het codewoord verwijderd worden, blijft ongeveer 6% bitfouten over. De decoder zal nu veel beter functioneren omdat er zeker drie symbolen correct zijn, zodat de kans op meer dan twee beschadigde symbolen veel kleiner is. Figuur 3.10 toont het resultaat van de decoder in deze situatie. Van de 6% bitfouten blijft bijna niets meer over (0.4%).



Figuur 3.10: *Het resultaat van de RS decoder op een afbeelding waaraan 6% niet-willekeurige ruis werd toegevoegd. Nog slechts* 0.4% *is onherstelbaar.*

Een ander voorbeeld met wat meer realistische parameters betreft de (255,247) RS code, waarbij de symbolen acht bits lang zijn. Deze code telt slechts 64 bits redundantie in een codewoord van 2040 bits. De code kan tot vier symbolen corrigeren. Elke error burst die tot 25 bits beschadigt, zal nooit meer dan vier symbolen aantasten, en kan dus worden gecorrigeerd. Uiteraard mag er dan geen andere bitfout meer voorkomen binnen een van de andere symbolen van het codewoord.

3.7 Andere codes en technieken

In de vorige secties werden enkele FEC codes besproken om de lezer een idee te geven van wat FEC codes zijn. Deze codes werden geselecteerd op basis van hun educatief nut of hun veelvuldig praktisch gebruik. Repetition

codes worden in praktijk zelden gebruikt omdat andere codes een veel betere verhouding bieden tussen informatie ratio en BER. Toch vormen ze samen met Hamming codes een goede inleiding tot de FEC materie. RS codes benadrukken de complexiteit waarmee sommige codes proberen het informatie ratio zo hoog mogelijk te houden, terwijl ze toch zoveel mogelijk transmissiefouten proberen te corrigeren. Hamming codes en RS codes zijn verder interessant vanwege hun wijdverspreide toepassingen [24]. Hamming codes worden gebruikt in duur computergeheugen (ECC RAM) voor servers om mogelijke bitfouten te elimineren en daardoor een hogere betrouwbaarheid van het systeem te garanderen. RS codes vormen onder andere de basis voor de bescherming van CD's en DVD's tegen de beschadiging van bits door vuil en krassen.

Uiteraard zijn er nog heel wat andere codes ontwikkeld die niet mogen ontbreken in dit werk. Deze sectie vormt een overzicht. Daarnaast worden ook enkele technieken besproken die de efficiëntie van de codes verhogen. [37] is een zeer volledig naslagwerk in verband met de theoretische benadering van codes.

3.7.1 BCH codes

BCH codes [37] werden rond dezelfde periode ontdekt door zowel Hocquenghem [21] als door Bose en Ray-Chaudhuri [6]. Deze codes hebben veel gelijkenissen met RS codes. Samen met RS codes zijn ze de meest gebruikte cyclische codes.

Het grote verschil tussen BCH en RS codes zit in de encoder. BCH codes rekenen in een Galois veld $GF(p)$ en bijhorende extensievelden $GF(p^m)$, waarbij p een andere waarde kan zijn dan 2 (herinner dat RS codes steeds de waarde 2 gebruiken). De generator veelterm wordt op een andere manier gedefinieerd dan die van RS codes. Het decoderingsproces is echter zeer gelijkaardig en de RS decoderingsalgoritmes kunnen ook gebruikt worden voor het decoderen van BCH codes.

3.7.2 LDPC codes

Het mag ondertussen duidelijk zijn dat decoderen de meest intensieve stap is in een code. Robert Gallager bedacht in zijn doctoraatsthesis [17] dat het ontwerp van een code moest gebaseerd zijn op de decoder, zodat deze zo efficiënt mogelijk kan uitgevoerd worden. Het resultaat van deze bevindingen zijn de Low-Density Parity-Check (LDPC) codes [33, 37, 51]. Deze codes zijn lineaire blok codes. In de literatuur worden ze soms ook als Gallager codes aangewezen. In deze sectie worden enkel binaire LDPC codes bekeken, maar een uitbreiding naar niet-binaire varianten bestaat [13].

De decoder operatie maakt net als bij de Hamming codes gebruik van een pariteitscontrole matrix H. Net als in vergelijking 3.9 geldt dat de

vermenigvuldiging van een correct codewoord t met deze matrix 0 heeft als resultaat. Gallager constateerde dat hoe minder 1 waardes er voorkomen in H, hoe sneller deze operatie kan worden uitgevoerd.

Een (n,k) LDPC code heeft een $(n-k) \times n$ pariteitscontrole matrix. $(n-k)$ wordt in de literatuur de *dimensie* van de code genoemd en wordt aangeduid met de letter m. De dimensie is het aantal pariteitsbits in het codewoord.

In de standaard LDPC code heeft elke rij in de matrix hetzelfde gewicht. Hiermee wordt bedoeld dat elke rij even veel 1 waardes telt. Het rijgewicht ω_r wordt klein gekozen. Typische waardes zijn 3 of 4. Deze kleine waardes worden in praktijk gebruikt voor codes waarbij $n > 10000$! Ook het kolomgewicht ω_c is voor elke kolom identiek. Opdat elke rij hetzelfde gewicht ω_r kan hebben, moet voldaan worden aan de vergelijking

$$\omega_c n = \omega_r m,$$

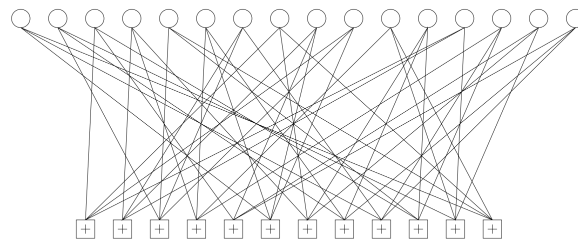
waaruit ω_c berekend kan worden van zodra n, m en ω_r gekozen zijn.

Voor het decoderingsalgoritme wordt de pariteitscontrole matrix omgevormd tot een *bigraaf*. Dit is een graaf met twee soorten knopen waarbij de ene soort met de andere wordt verbonden. In deze context zijn er bitknopen en controleknopen. Voor een $m \times n$ matrix zijn er n bitknopen en m controleknopen: voor elke kolom respectievelijk rij is er een knoop. Een 1 waarde in de pariteitscontrole matrix is een boog tussen de bitknoop die bij de kolom van die waarde hoort en de controleknoop aangeduid door de rij. Voorbeeld 10 demonstreert deze concepten.

Voorbeeld 10 *Een (16,4) LDPC code heeft een* 12×16 *pariteitscontrole matrix* H*. Dit voorbeeld toont een matrix met kolomgewicht* $w_c = 3$ *en rijgewicht* $w_r = 6$*. Uiteraard is dit slechts één van de vele matrices die kunnen worden gebruikt in een (16,4) code. Om duidelijk aan te tonen dat het een dunne matrix is, worden enkel de 1 waardes getoond. Figuur 3.11 toont de bijhorende bigraaf. Voor elke 1 in de matrix is er een boog in de graaf die een bitknoop verbindt met een controleknoop.*

$$H = \begin{bmatrix}
 & & 1 & & & & & 1 & & 1 & & & 1 & & & \\
 & & & 1 & & & 1 & & 1 & & & & 1 & & & \\
 & 1 & & & 1 & & 1 & & & 1 & & & & & & \\
 & & & 1 & & 1 & & & & & 1 & & & 1 & & \\
 & & 1 & & & & & & 1 & & & & & & 1 & 1 \\
1 & & & & & 1 & & & 1 & 1 & & & & & & \\
 & & & 1 & & & & 1 & & & & 1 & & & 1 & \\
 & 1 & & & & 1 & & & & & & 1 & & & & 1 \\
1 & & & & & & 1 & & & & & & & 1 & & 1 \\
 & & 1 & & 1 & & & & & & & 1 & & 1 & & \\
 & 1 & & & & & & & & & 1 & & 1 & & 1 & \\
1 & & & & 1 & & & 1 & & & 1 & & & & &
\end{bmatrix}$$

Het decoderingsalgoritme werkt als volgt: bij de ontvangst van een codewoord worden de bits geplaatst op de bitknopen van de bigraaf. Op de graaf wordt vervolgens een *belief propagation* [66] algoritme uitgevoerd. Dit algoritme bepaalt aan de hand van *message passing* over de graaf voor elke bit een geloofwaardigheid. Dit soort algoritmes heeft de eigenschap iteratief te werken. Na elke iteratie is er meer zekerheid over de correctheid van een bit. Het algoritme stopt als voor elke bit met voldoende zekerheid kan worden gezegd of deze correct of foutief is, of na een bepaald aantal iteraties. Bits die foutief worden bevonden, worden gecorrigeerd. Zo wordt het originele codewoord benaderd. Het algoritme kan niet evenveel fouten corrigeren als een ML decoder, maar kent een veel efficiëntere implementatie, in het bijzonder wanneer er weinig bogen zijn in de graaf. Dit laatste is net een eigenschap van LDPC codes omdat ze zo worden ontwikkeld dat de pariteitscontrole matrix weinig 1 waardes bevat.



Figuur 3.11: *De bigraaf horende bij de pariteitscontrole matrix uit voorbeeld 10. De 16 bitknopen bovenaan worden door bogen verbonden met de 12 controleknopen onderaan volgens de posities van de 1 waardes in de matrix. Bron: [33]*

LDPC codes zijn lange tijd vergeten geweest, maar vandaag er is opnieuw veel interesse uit wetenschappelijke hoek. Gallager bewees dat bij (n,k) LDPC codes de minimale afstand lineair stijgt met n als de rij- en kolomgewichten niet wijzigen [37]. Hoe dunner de graaf wordt, des te groter de minimale afstand. Grote pariteitscontrole matrices kunnen bijgevolg veel bescherming bieden en vanwege de dunne bigraaf zijn ze zeer snel te decoderen.

3.7.3 Convolutionele codes

Convolutionele codes [33, 37] verschillen grondig van de blok codes die tot hiertoe werden besproken. Blok encoders verwachten dat de te verwerken informatie is opgesplitst in berichten van een vaste grootte. Ze berekenen een codewoord door enkel informatie te gebruiken uit dit bericht. Convolutionele codes daarentegen combineren deze informatie met de data uit andere berichten.

De encoder van een convolutionele code voert een filter operatie uit, vergelijkbaar met het proces van digitale audio filters [47]. Convolutionele codes zijn steeds lineair. Net als bij audio filters zijn er *FIR encoders*, die een stuk informatie slechts een beperkt aantal stappen gebruiken (Finite Impulse Response), en *IIR encoders*, waarbij een stuk informatie de uitvoer van de encoder voor onbepaalde tijd beïnvloedt (Infinite Impulse Response).

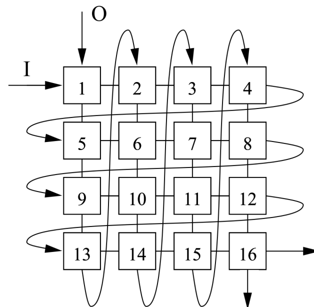
Alle convolutionele codes kunnen worden gedecodeerd met het Viterbi algoritme. Dit algoritme is even precies als een ML decoder en is zeer gemakkelijk te parallelliseren.

Ongetwijfeld de bekendste toepassing van convolutionele codes is het gebruik ervan in de communicatie met de Voyager ruimtevaartuigen van het Amerikaanse NASA.

3.7.4 Interleaving

Om een goede bescherming te bieden tegen transmissiefouten, moet een systeem error bursts kunnen corrigeren. Het verlies van opeenvolgende bits in een codewoord is voor veel codes echter fataal. Met interleaving kunnen error bursts beter opgevangen worden. [33, 43, 44]

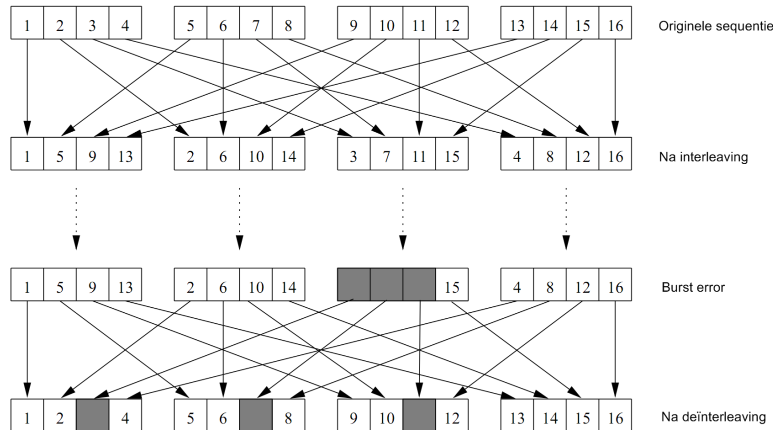
Een interleaver is een proces dat de symbolen van een oorspronkelijke datasequentie door elkaar haalt. Een deïnterleaver keert dit proces om en herstelt de oorspronkelijke volgorde van de symbolen. Een blok interleaver kan worden voorgesteld met een $n \times m$ matrix. Telkens worden nm symbolen rijgewijs ingelezen in de matrix en vervolgens kolomgewijs weer uitgelezen. Hierdoor krijgen de symbolen een andere volgorde. Deze volgorde is niet willekeurig maar vormt een duidelijk patroon. Figuur 3.12 demonstreert dit proces met behulp van een 4×4 interleaver.



Figuur 3.12: *De matrix voor een 4×4 interleaver. De invoer- en uitvoervolgorde worden met pijlen aangeduid. I: invoer; O: uitvoer. Bron: [44]*

Als een geïnterleavede datasequentie wordt beschadigd door een error burst, dan zal de deïnterleaver de bitfouten verspreiden over de originele sequentie. Dit wordt getoond in figuur 3.13. Codes die beter fungeren met verspreide bitfouten zullen bijgevolg beter presteren als codewoorden worden

geïnterleaved alvorens ze worden verzonden.



Figuur 3.13: *Het interleaving proces. De geïntroduceerde error burst wordt verspreid over de oorspronkelijke sequentie. Bron: [44]*

3.7.5 Concatenated codes

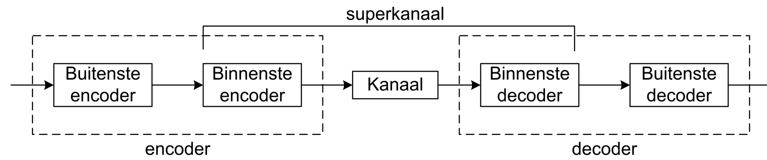
Intuïtief lijkt het logisch dat wanneer twee codes na elkaar worden gebruikt de foutcorrectie capaciteit van de combinatie hoger zal zijn dan wat met de codes afzonderlijk kan worden bereikt. Dit is het idee achter concatenated codes [37].

Typisch worden twee codes achter elkaar geplaatst. De buitenste code is vaak een complexe code zoals een RS code. Voor de binnenste code wordt een binaire code gebruikt. Een bitsequentie wordt eerst door de buitenste code geëncodeerd. De resulterende codewoorden dienen als invoer voor de binnenste code.

De binnenste code genereert in feite een "superkanaal" voor de buitenste code: het kanaal waarover gecommuniceerd moet worden, wordt door de binnenste code zo beschermd dat de buitenste code ideaal kan fungeren om de resterende fouten te corrigeren. Stel als voorbeeld dat de binnenste code een convolutionele code is. Deze zal door zijn foutcorrectie het communicatiekanaal verbeteren. De Viterbi decoder — de decoder voor convolutionele codes — heeft de eigenschap om resterende fouten te groeperen [37]. Deze kunnen worden gecorrigeerd door de buitenste code, die in dit geval bij voorkeur een RS code is omdat RS codes de eigenschap hebben zeer goed gegroepeerde fouten te kunnen corrigeren.

Figuur 3.14 toont schematisch een kanaal dat door twee codes wordt beschermd. Ook meer dan twee codes kunnen worden samengevoegd. Er kan verder ook een interleaver worden toegevoegd voor een betere bescherming tegen error bursts. Elke extra stap vraagt echter extra rekenkracht, waardoor in praktijk nooit meer dan enkele codes achter elkaar geplaatst

worden.



Figuur 3.14: *Een concatenated code.*

3.7.6 Product codes

Een product code [37] is net als een concatenated code opgebouwd door de samenvoeging van andere codes. Deze methode om codes samen te voegen vereist echter dat de codes systematische lineaire blok codes zijn. De te coderen symbolen worden in een matrix genoteerd. Met de eerste code C_1 wordt elke rij geëncodeerd. Dit resulteert in pariteitssymbolen die achter elke rij worden toegevoegd. De matrix wordt dus uitgebreid met enkele nieuwe kolommen. Vervolgens wordt elke kolom in de uitgebreide matrix geëncodeerd met de tweede code C_2. De tweede code beschermd dus niet alleen de symbolen van het originele bericht, maar ook de pariteitssymbolen van de eerste code. Schematisch wordt de product code weergegeven in figuur 3.15.

Bericht	A
B	C

Figuur 3.15: *Schematische voorstelling van een product code $C_1 \times C_2$. (A) Pariteitssymbolen van het encoderen van de rijen van de bericht matrix met C_1; (B) Pariteitssymbolen van het encoderen van de kolommen van de bericht matrix met C_2; (C) Pariteit door het encoderen van de pariteit van C_1 met C_2.*

Door de dubbele bescherming van het bericht en door de bescherming van de pariteitssymbolen van C_1 hebben product codes een veel grotere correctie capaciteit. Er kan worden bewezen [37] dat de combinatie van een code C_1 met een minimale afstand d_1 en een code C_2 met minimale afstand d_2 resulteert in een product code $C_1 \times C_2$ met een minimale afstand gelijk aan $d_1 d_2$.

3.7.7 Turbo codes

Turbo codes [37] zijn vrij recente codes: ze werden pas ontwikkeld in 1993. Deze codes combineren korte convolutionele codes. Hierdoor weten ze de

performance te bereiken van lange convolutionele codes met een veel minder complexe decoder. De decoder werkt net als de LDPC decoder iteratief. Bij elke iteratie neemt de kwaliteit van het gedecodeerde beeld toe.

3.8 Shannon's Theorema

Claude Shannon formuleerde in 1948 in zijn paper [50] een theorema waarin hij bewijst dat er voor een kanaal met gekende ruisfactor een getal bestaat, de *capaciteit* C van het kanaal, waarvoor geldt dat, wanneer informatie verstuurd wordt over dit kanaal met een informatie ratio R kleiner dan C, de kans op fouten oneindig klein kan zijn. Communicatie met een ratio $R < C$ kan bijgevolg betrouwbaar genoemd worden. Dit theorema wordt ook wel het *Noisy-channel coding theorem* genoemd. [33, 51]

Het is interessant om data te versturen met een ratio dat zo dicht mogelijk bij de capaciteit in de buurt komt, omdat zo het kanaal het best benut wordt. De capaciteit voor een kanaal is een theoretische waarde. Shannon's theorema bepaalt niet hoe een methode werkt die data betrouwbaar kan versturen en tegelijk in staat is de capaciteit te benaderen. Het theorema beschrijft enkel welke capaciteit de beste methode voor het kanaal zou kunnen halen.

De omgekeerde bewering is ook belangrijk: als informatie verstuurd wordt met een ratio $R > C$, dan kan geen enkele code data versturen met een oneindig kleine kans op fouten. Daarom wordt de capaciteit soms ook de *Shannon limiet* genoemd.

Uiteraard rijst de vraag in hoeverre de capaciteit van een kanaal kan worden bereikt. In theorie kunnen alle codes zorgen voor een oneindig kleine kans op fouten, zolang de lengte van het codewoord, en bijgevolg de hoeveelheid redundantie, maar groot genoeg gekozen wordt. Zeer grote codewoorden maken het encoderen en het decoderen echter praktisch niet haalbaar, omdat dit te veel rekenkracht vraagt. Dit geldt zeker voor codes die enkel over een ML decoder beschikken. Het kan namelijk worden aangetoond [51] dat ML decoderen een NP-compleet probleem is.

We onderscheiden enkele groepen codes:

Zeer goede codes Deze codes laten communicatie toe met een oneindig kleine kans op fouten en met een ratio dat de capaciteit van het kanaal benadert.

Goede codes Met goede codes kan ook gecommuniceerd worden met een oneindig kleine kans op fouten, maar met een maximaal ratio dat lager ligt dan de capaciteit van het kanaal.

Slechte codes Slechte codes zijn niet in staat om de kans op fouten oneindig klein te maken tenzij de lengte van hun codewoorden oneindig groot wordt.

Praktische codes Codes waarbij de encoder en de decoder zowel in tijd als in ruimte (geheugengebruik) maximaal met polynomiale complexiteit uitvoerbaar zijn.

Het uitwerken van een code die zo goed is als Shannon belooft en toch praktisch uitvoerbaar is, is nog steeds een onopgelost probleem. De grootste groep van ontwikkelde codes zijn lineair. Ze worden algemeen als 'goed' omschreven. Lineaire codes hebben eenvoudige encoders, zoals vergelijking 3.2 bewijst. Ze zijn echter enkel praktisch als ze een intelligentere decoder hebben dan een ML decoder.

Repetition codes zijn slechte codes: ze zijn niet in staat om de kans op fouten oneindig klein te maken. RS codes, Turbo codes en LDPC codes zijn op dit moment de beste codes en zijn in staat de Shannon limiet zeer dicht te benaderen, al is hiervoor het gebruik van lange codewoorden en bijgevolg zeer veel rekenkracht vereist.

Hoofdstuk 4

Beschermen van multimediastromen

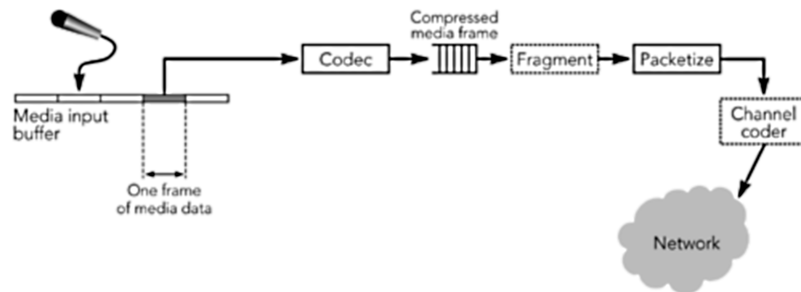
In het vorige hoofdstuk werden FEC codes beschreven als technieken die toelaten om informatie na ontvangst te corrigeren indien er transmissiefouten optraden. Dit hoofdstuk zet deze theorie om naar praktische toepassingen. In het bijzonder wordt hier gefocust op de bescherming van multimediastromen die worden verstuurd met behulp van het Real-time Transport Protocol (RTP) [48]. Dit protocol werd al eens aangehaald in sectie 2.5, waarin werd beschreven waarom real-time mediastromen moeten worden beschermd via FEC. De eerste sectie van dit hoofdstuk licht het RTP protocol toe als achtergrondinformatie. In het vervolg van het hoofdstuk worden technieken aangehaald die FEC codes in een praktische omgeving plaatsen om de ontvangst van multimediastromen te verbeteren. Deze tekst beperkt zich vaak tot audio- en videostromen, maar de theorie kan evenzeer worden toegepast op andere vormen van multimediale data.

4.1 Mediatransport via RTP

Deze sectie geeft een overzicht van de functies die de zender en ontvanger moeten vervullen bij het uitwisselen van een RTP stroom. Deze samenvatting beperkt zich tot de netwerkaspecten die hierbij komen kijken. Voor een uitgebreide discussie betreffende het protocol en de niet besproken elementen wordt [43] aangeraden.

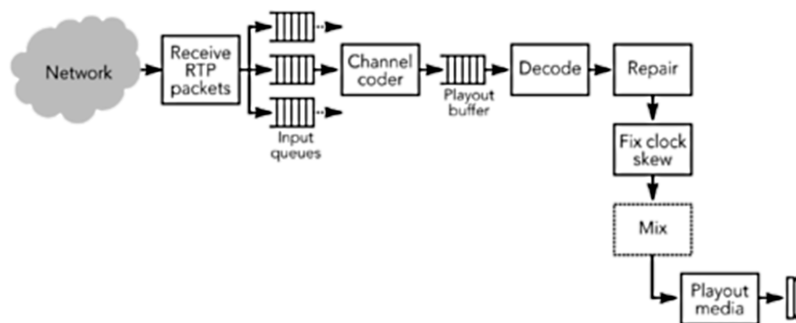
Figuur 4.1 bevat alle stappen die een zender uitvoert om media te versturen. Een mediabron wordt opgenomen. Na digitalisatie wordt deze ongecomprimeerde data doorgegeven aan de media encoder. Deze comprimeert de data tot een voldoende kleine hoeveelheid informatie die geschikt is voor transmissie. De media encoder genereert media frames. Voor een audio codec zijn dit typisch zeer korte stukjes geluid. Bij een video codec bevat elk frame meestal informatie over één beeld uit de videosequentie. De frames

worden onderverdeeld in RTP pakketten. Afhankelijk van de grootte van de frames worden ze gebundeld om minder netwerkpakketten te moeten versturen, of juist gefragmenteerd omdat de hoeveelheid informatie niet in één keer kan worden verstuurd. Elk pakket krijgt een tijdstip en een sequentienummer mee waarmee de ontvanger kan bepalen wanneer de inhoud van het pakket moet worden afgespeeld. De pakketten worden vervolgens over het netwerk verzonden via UDP.



Figuur 4.1: *Taken van een RTP zender. Bron: [43]*

Aan de ontvangstkant (figuur 4.2) zorgt de extra informatie uit het RTP pakket ervoor dat uit de ontvangen pakketten de originele mediastroom kan worden gereconstrueerd (in de veronderstelling dat geen informatie verloren gaat). Ontvangen pakketten worden per mediastroom in *playout buffers* geplaatst. Omdat het transport over UDP verloopt, zijn er weinig zekerheden over hoe de pakketten arriveren. Buffers laten toe dat de *delay* en *jitter* die het netwerk introduceert, worden gecompenseerd en dat pakketten in de juiste volgorde worden geplaatst. Vervolgens wordt data op het juiste tijdstip doorgegeven aan de media decoder. De gedecodeerde mediastroom wordt tenslotte via een geschikt uitvoerapparaat weergegeven.



Figuur 4.2: *Taken van een RTP ontvanger. Bron: [43]*

Met RTP kunnen complexe structuren van mediastromen worden opgebouwd. Zo verzorgt RTP onder andere synchronisatie tussen verschillende stromen zodat bijvoorbeeld audio en video samen één geheel vormen. RTP

is verder zeer geschikt om een sessie met duizenden ontvangers van dezelfde mediastroom te beheren. Dergelijke mogelijkheden maken van RTP een sterk protocol voor mediatransport in zeer uiteenlopende real-time multimedia toepassingen.

Zowel zender als ontvanger hebben in hun figuren een optionele *channel coder*. Deze laat toe om extra operaties uit te voeren op de RTP pakketten vooraleer ze worden verzonden door de zender of nadat ze arriveerden bij de ontvanger. De channel coder kan FEC technieken implementeren om datacorruptie tijdens transmissie te beperken.

4.2 Foutcorrectie voor RTP

Net als alle andere Internetcommunicatie is RTP onderhevig aan twee soorten datacorruptie:

- Verlies van pakketten
- Corruptie van de inhoud van pakketten

Verlies van pakketten door netwerkcongestie en andere kwesties vormt het grootste probleem op het Internet. Dat werd ontwikkeld als een *best-effort* netwerk zonder garanties. Bijgevolg kan een deel van de informatie van een mediastroom niet of te laat aankomen bij de ontvanger. Bij draadloze verbindingen komt daarnaast ook nog corruptie van de informatie bij door externe invloeden op het signaal. Ontbrekende of foutieve informatie kan niet aan de media decoder worden gegeven. Dit resulteert in een verlies van kwaliteit en bijgevolg in een lagere QoE.

Intelligente media decoders proberen om met behulp van zogenaamde *error concealment* technieken transmissiefouten zoveel mogelijk te verbergen in hun uitvoer. Continue media verschillen vaak weinig tussen twee opeenvolgende media frames en deze coherentie wordt uitgebuit. Onder andere met interpolatie kan een decoder ontbrekende data proberen aan te vullen.

Foutcorrectie is echter nog steeds zeer nuttig in deze context. Error concealment kan zichtbaar zijn voor de gebruiker omdat deze technieken proberen te raden wat de inhoud van verloren informatie was. Dit proces is uiteraard niet altijd even accuraat. Foutcorrectie daarentegen kan het aantal transmissiefouten aanzienlijk verminderen, waardoor minder fouten moeten worden verborgen met behulp van error conceilment en de kwaliteit van de ontvangen mediastroom stijgt.

Omdat met real-time stromen wordt gewerkt, is er een grote beperking op de mogelijkheden van FEC: pakketten moeten tijdig bij de ontvanger aankomen of de ontvanger moet tijdig de redundante informatie verzamelen om verloren pakketten te herstellen. In het bijzonder bij interactieve applicaties mag de totale end-to-end delay niet groter worden dan ongeveer 100 ms om

onder de grens van de menselijke perceptie te blijven [3]. In een multimedia context wordt niet verwacht dat alle transmissiefouten worden geëlimineerd door FEC technieken. De resterende fouten kunnen met behulp van error concealment verborgen worden. Dit zal resulteren in een verhoogde QoE terwijl toch het real-time karakter van de toepassingen behouden blijft.

4.3 Reconstructie van volledige pakketten

Een verloren pakket zorgt ervoor dat de media decoder gedurende een bepaalde periode geen uitvoer kan genereren of error concealment technieken moet toepassen. Om dit te vermijden moet getracht worden om met behulp van redundante informatie de ontbrekende pakketten te reconstrueren. Verschillende originele pakketten worden gecombineerd om een redundant pakket te vormen dat deze reconstructie toelaat.

4.3.1 Parity code

[30] beschrijft een methode waarbij elke bit van een pakket via een XOR operatie wordt gecombineerd met een corresponderende bit uit een tweede pakket. Dit resulteert in een derde pakket. Stel dat A en B originele pakketten zijn, en $C = A$ XOR B is het resulterende redundante pakket, dan geldt:

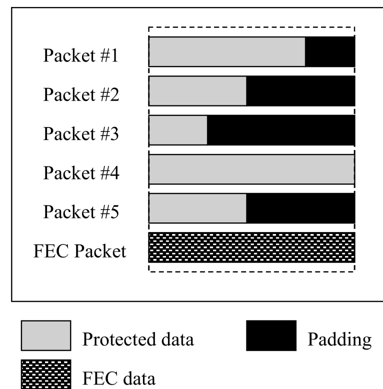
$$C \text{ XOR } A = (A \text{ XOR } B) \text{ XOR } A = B$$
$$C \text{ XOR } B = (A \text{ XOR } B) \text{ XOR } B = A$$

Als één van de drie pakketten verloren gaat, kan dit bijgevolg worden gereconstrueerd uit de andere twee.

Om de XOR operatie te kunnen uitvoeren, moeten alle pakketten even lang zijn. Indien pakketten een variabele lengte hebben, kunnen extra bits met padding worden toegevoegd om dit te bereiken.

Ook meer dan twee originele pakketten kunnen gecombineerd worden door de XOR operatie te herhalen voor elk origineel pakket. Met behulp van het resulterende redundante pakket kan steeds één pakket per groep gecombineerde pakketten worden gereconstrueerd. De grootte van de groep pakketten die tezamen worden gecombineerd, moet bijgevolg worden gekozen naargelang het percentage pakketten dat verloren gaat. Figuur 4.3 toont het genereren van een FEC pakket uit vijf originele pakketten.

De XOR operatie uit [30] is een eenvoudige FEC code zoals de codes beschreven in hoofdstuk 3. Ze wordt ook de *parity code* genoemd, omdat ze voor elke bit in het redundante pakket eigenlijk slechts de pariteit berekent van de corresponderende bits in de originele pakketten. Dit is exact wat de pariteitscontrole uit sectie 2.3.1 ook doet. De parity code verschilt van de codes uit hoofdstuk 3 in die zin dat de code niet in staat is om te bepalen



Figuur 4.3: *Een FEC pakket wordt gegenereerd uit vijf originele pakketten. Met padding worden alle pakketten even lang gemaakt. Bron: [31]*

welke gegevens er verloren gingen. De code kan pas een pakket reconstrueren als via een andere methode wordt bepaald welk pakket niet aankwam. Bij een RTP stroom vormt dit echter geen probleem. RTP voegt sequentienummers toe bij elk pakket. Bijgevolg is geweten welk pakket niet werd ontvangen!

4.3.2 Gebruik van andere codes

De parity code kan slechts één pakket herstellen per groep gecombineerde originele pakketten. Als het netwerk veel pakketten verliest, kunnen bijgevolg slechts enkele pakketten worden gecombineerd en wordt er procentueel veel redundante informatie gegenereerd. Met de meer geavanceerde codes uit het vorige hoofdstuk is het mogelijk deze hoeveelheid te doen dalen. [58] gebruikt RS codes als bescherming. Een (n,k) RS code wordt gebruikt om k pakketten te beschermen, en genereert $n-k$ redundante pakketten. Omdat de sequentienummers in de RTP pakketten de positie van de verloren informatie aanduiden, kunnen met deze code $n-k$ pakketten gereconstrueerd worden. Dit is een logisch gevolg van vergelijking 3.20. Daar de posities β_l in deze vergelijking gekend zijn, halveert het aantal onbekenden in het stelsel. Er ontstaat dus plaats voor meer n_{j_l} variabelen, zodat twee keer zoveel pakketten kunnen gereconstrueerd worden.

4.3.3 Redundante mediastroom

In [30] wordt voorgesteld om de redundante pakketten te transporteren via een aparte mediastroom die gebruik maakt van andere netwerkpoorten dan de mediastroom die wordt beschermd. Dezelfde tekst bespreekt een RTP payload formaat dat ontwikkeld werd om FEC informatie te transporteren. Dit heeft als grote voordeel dat een stroom compatibel blijft met ontvangers

die niet overweg kunnen met de redundante data. Achterwaartse compatibiliteit is belangrijk in situaties waarbij niet alle ontvangers kunnen worden aangepast als er een nieuwe zender wordt gebruikt die FEC informatie verstuurt. Oude ontvangers zullen de redundante mediastroom negeren en bijgevolg geen foutcorrectie toepassen, terwijl aangepaste ontvangers een betere QoE kunnen aanbieden aan hun gebruikers.

4.3.4 Keuze van de pakketten

Om een optimale bescherming te kunnen bieden, is de keuze van welke van de originele pakketten worden gecombineerd cruciaal. Opeenvolgende pakketten combineren is niet efficiënt als het netwerk vaak pakketten in burst verliest. Door interleaving (sectie 3.7.4) op pakketniveau kunnen pakketten in een andere volgorde geplaatst worden zodat een error burst beter kan worden opgevangen.

Om foutcorrecties uit te voeren, moet de ontvanger wachten op alle pakketten die tezamen werden beschermd. Real-time multimedia applicaties kunnen echter niet altijd lange wachttijden accepteren. FEC informatie die te laat wordt ontvangen om nog gebruikt te worden is slechts verspilling van bandbreedte. Dit beïnvloedt de keuze van de hoeveelheid pakketten die samen een groep vormen. Grote groepen zijn niet geschikt voor interactieve applicaties. Ook bij interleaving moet er worden opgelet dat samenhorende pakketten toch niet te ver verspreid geraken, zodat ze tijdig allemaal worden ontvangen en foutcorrectie indien nodig dus tijdig kan worden toegepast.

4.3.5 Gevolgen van extra pakketten

In sectie 2.2 werd beschreven dat pakketverlies vooral veroorzaakt wordt door netwerkcongestie. Het lijkt dan op het eerste zicht niet erg interessant om extra pakketten te gaan introduceren om foutcorrectie mogelijk te maken, omdat die extra pakketten het netwerk nog meer zullen belasten. In [58] werden echter enkele experimenten uitgevoerd die bewijzen dat de extra pakketten voor FEC wel degelijk nuttig kunnen zijn.

In deze experimenten werden RS codes gebruikt. Er werd uitgezocht wat het gevolg was van deze codes op het BER onder variërende netwerkbelasting. Uit deze experimenten kan worden geconcludeerd dat toevoeging van FEC pakketten een betere ontvangst garandeert zolang de totale netwerkbelasting minder dan 95% van de capaciteit van het netwerk bedraagt.

4.4 Vervangen van volledige pakketten

4.4.1 Redundant Audio Coding

Niet voor alle mediastromen moet reconstructie van een pakket exact zijn. In [45] wordt een techniek beschreven die toegepast wordt op audiostromen. Voor elk RTP pakket berekent de zender een tweede redundante versie die ook wordt uitgestuurd. Een verloren pakket wordt vervangen met de redundante kopie. Deze techniek wordt *Redundant Audio Coding* genoemd. De kopie wordt verondersteld om door een codec met lagere kwaliteit gegenereerd te worden, zodat ze aanzienlijk minder bandbreedte verbruikt dan het originele pakket. De gebruiker zal een daling in kwaliteit ervaren, maar de audiostroom kan toch correct waargenomen worden. Een daling in kwaliteit is steeds beter dan het volledig ontbreken van informatie. De subjectieve waarneming van het resultaat deze techniek wordt als behoorlijk goed omschreven [20].

4.4.2 Transport via piggybacking

Om de redundante versie van een pakket te transporteren, wordt in [45] een RTP payload formaat voorgesteld om deze via piggybacking toe te voegen aan een volgend pakket van de originele mediastroom. Op die manier moet de ontvanger wachten op dit volgende pakket om een verloren pakket te vervangen.

Een pakket kan kopieën van meerdere andere pakketten bevatten en een kopie kan worden meegestuurd met meerdere originele pakketten. Hierdoor stijgt de kans dat een kopie voor een verloren pakket aankomt. Uiteraard stijgt ook de benodigde bandbreedte.

Men is niet verplicht om de kopie van een pakket met het volgende pakket mee te sturen. Om beter te beschermen tegen error bursts, kunnen kopieën aan een veel later pakket worden toegevoegd. Dit betekent wel dat de ontvanger langer zal moeten wachten op de redundante versie van een verloren pakket, hetgeen voor sommige applicaties mogelijk minder geschikt is.

Omdat piggybacking een speciaal payload formaat gebruikt, is deze techniek niet achterwaarts compatibel. Ontvangers die het payload formaat niet begrijpen, zullen niet in staat zijn de mediastroom te decoderen.

4.5 Bescherming van de inhoud van pakketten

De tot hiertoe besproken technieken focussen op het herstellen van pakketverlies. In vele situaties volstaan deze technieken omdat corruptie van pakketten slechts zelden voorkomt op netwerken met bedrade infrastructuur. Herinner u dat RTP communicatie via het UDP protocol verloopt. Indien

er toch een bit zou beschadigd worden, dan zal de controlesom van de UDP header niet meer correct zijn en zal het pakket door de netwerkkaart weggeworpen worden. Bescherming tegen pakketverlies zal mogelijk het foutief ontvangen pakket herstellen.

Bij draadloze verbindingen is de kans op bitfouten binnen een pakket veel groter. In [28] spreekt men van een BER variërend van $10^{-}5$ tot $10^{-}3$ voor een cellulair telefoonnetwerk. Omdat de UDP controlesom incorrect wordt bij het voorkomen van een bitfout, moeten op dit soort verbindingen veel meer berichten worden weggeworpen dan op het hedendaagse Internet. Om dit pakketverlies te beschermen met eerder besproken technieken zou te veel redundantie nodig zijn en zou het informatie ratio zeer klein worden. Door de controlesom van de UDP pakketten uit te schakelen, kunnen bitfouten met de FEC technieken uit hoofdstuk 3 opgespoord en gecorrigeerd worden.

Een zeer voor de hand liggende mogelijkheid zou zich tussen de UDP en RTP laag nestelen. Elk RTP pakket wordt volledig met een FEC code beschermd vooraleer het wordt verzonden en de FEC decoder bij de ontvanger voert de correctie uit vooraleer het pakket aan de RTP applicatie wordt doorgegeven. Het is echter een slecht idee om een volledig RTP pakket om te zetten naar een codewoord omdat daardoor de RTP header informatie niet meer leesbaar is voor apparaten die niet op de hoogte zijn van de bescherming. Een RTP applicatie die een pakket ontvangt als codewoord zal niet weten wat te doen en dit resulteert in ongedefinieerd gedrag.

De inhoud van het payload veld in elk RTP pakket kan wel omgezet worden in een codewoord. Het payload type dat wordt toegekend aan de stroom wordt zo gekozen dat ontvangers die de FEC mogelijkheden niet ondersteunen de stroom negeren, zoals dit hoort wanneer een payload type niet herkend wordt [30].

4.6 Unequal Error Protection

Niet elk stuk van een datastroom is steeds even belangrijk. Belangrijkheid kan worden onderscheiden op verschillende niveau's:

Pakketstroom niveau In een pakketstroom kan het ene pakket belangrijker zijn dan het andere. Bij een video codec die gebruik maakt van *motion compensation* [60] zijn I-frames belangrijker dan P- of B-frames, omdat een fout in een I-frame zich propageert in alle P- en B-frames die dat I-frame als referentieframe gebruiken [36]. Pakketten met een I-frame zijn bijgevolg belangrijker dan andere.

Pakket niveau In een pakketformaat is niet elk veld even belangrijk. Header informatie is in vergelijking met payload data zeer belangrijk. Een fout in de payload van een videostroom leidt tot een tijdelijke degradatie van het beeld of een deel van het beeld. Een header fout

daarentegen kan tot fout gedrag van een systeem kan leiden [43]. Een bitfout in een UDP header kan ervoor zorgen dat het adres van de ontvanger wijzigt, waardoor een pakket naar het verkeerde systeem wordt gerouteerd!

Payload niveau Sommige media codecs kunnen een onderscheid maken tussen informatie die vitaal is opdat de decoder kan werken, en andere informatie die slechts bijdraagt tot een betere kwaliteit van het eindresultaat. Vitale informatie is uiteraard belangrijker.

Methodes die belangrijke data beter beschermen dan andere informatie worden *Unequal Error Protection* (UEP) technieken genoemd. Ook de term *Unequal Loss Protection* wordt soms gebruikt.

4.6.1 UDP Lite

In sectie 4.5 werd beschreven dat de UDP controlesom kan worden uitgeschakeld zodat de payload van een RTP pakket kan worden beschermd tegen bitfouten. Daar werd echter niet vermeld wat er gebeurt als er een fout voorkomt in een header. In de inleiding van deze sectie werd reeds benadrukt dat een header fout ernstige gevolgen kan hebben. Meestal beschermt de controlesom een pakket tegen headerfouten, maar die werd uitgeschakeld.

UDP Lite [29] is een alternatief voor UDP dat toelaat om een gedeeltelijke controlesom te gebruiken. Deze controlesom wordt niet berekend over het hele pakket, maar er kan worden bepaald over hoeveel octets de controlesom reikt.

Met UDP Lite wordt het mogelijk om pakketten zo te beschermen dat een bitfout in een header resulteert in het wegwerpen van het pakket, terwijl een pakket met een bitfout in de payload toch wordt afgeleverd aan de applicatie. Omdat deze methode een verschillende bescherming biedt voor headers en payload, valt dit onder de groep van UEP technieken.

4.6.2 UEP van payload data

Sommige media codecs zijn in staat om een opsplitsing te maken tussen de informatie die cruciaal is om aan de ontvangstkant de payload te decoderen, terwijl de overige informatie bijkomstig is. Als deze laatste incorrect wordt ontvangen, kan de decoder enkel gebruik maken van de cruciale informatie. Dit zal resulteren in een uitvoer met lage kwaliteit, maar zoals bij de Redundant Audio Coding (sectie 4.4.1) is het steeds beter om een kwalitatief lagere versie weer te geven dan wanneer niets kan worden weergegeven. Niet-cruciale informatie die correct ontvangen wordt, wordt gedecodeerd en draagt bij tot een betere kwaliteit.

Vaak kan worden aangenomen dat een media encoder de meest belangrijke informatie vooraan in de payload plaatst [31]. Daardoor volgt de cruciale

informatie als eerste na de headers in een pakket. Door in de controlesom van UDP Lite ook deze informatie te betrekken, maakt dit protocol het mogelijk de belangrijkste informatie te beschermen. Voorbeeld 11 beschrijft een UEP bescherming die wordt gebruikt in de hedendaagse mobiele telefonie.

Voorbeeld 11 *De AMR (Adaptive Multi-Rate) media codec is ontwikkeld voor gebruik in derde-generatie mobiele telefonie. Voor deze codec wordt in [52] een RTP payload formaat gedefinieerd. Een frame van de AMR encoder bevat drie klassen bits. Klasse A bits zijn vitaal. Als deze beschadigd worden, kan het frame niet worden gedecodeerd. Klasse B en C bits verhogen de audio kwaliteit als ze correct ontvangen worden.*

Een RTP pakket met AMR data als payload wordt verstuurd met UDP Lite. De controlesom wordt enkel berekend over de IP/UDP Lite/RTP headers en *de A bits van de payload. Zolang de controlesom klopt bij de ontvanger, zal het pakket worden gedecodeerd. Als ook B of C bits correct arriveren, zorgen ze voor een betere geluidskwaliteit.*

Het AMR payload formaat ondersteunt ook interleaving en Redundant Audio Coding (sectie 4.4.1) voor een nog betere bescherming tegen transmissiefouten.

4.7 UEP met foutcorrectie

In de vorige sectie werd een onderscheid gemaakt tussen cruciale en niet-cruciale informatie. Indien enkel de niet-cruciale data beschadigd was, werd toch gedecodeerd met enkel de cruciale informatie. Er werd echter nergens een poging ondernomen om informatie te herstellen.

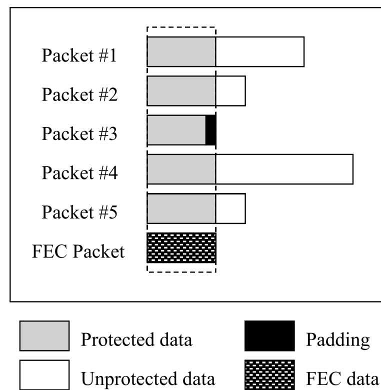
UEP kan worden uitgebreid met foutcorrectie. Een media codec geeft aan welke onderdelen van zijn frames belangrijk zijn. Afhankelijk van deze informatie wordt beslist hoeveel redundantie zal worden gebruikt om elk onderdeel de juiste bescherming te bieden. Dit heeft als voordeel ten opzichte van het volledig beschermen van de payload met FEC dat bandbreedte wordt bespaard, daar de minder belangrijke data met minder redundantie wordt beschermd. Toch heeft belangrijke data een grotere kans om hersteld te worden, waardoor de QoE voor de eindgebruiker toch aanzienlijk kan worden verbeterd.

4.7.1 Uneven Level Protection in RTP

In sectie 4.3.3 werd vermeld dat [30] een RTP payload formaat definieert dat FEC informatie kan transporteren. Dit payload formaat voorziet ook de mogelijkheid om verschillende onderdelen van een mediastroom apart te beschermen. In de specificatie van het payload formaat worden deze onderdelen *levels* genoemd. Elk level krijgt een aangepaste bescherming naar-

gelang zijn belangrijkheid. Daarom spreekt men in dit geval over *Uneven Level Protection* (ULP).

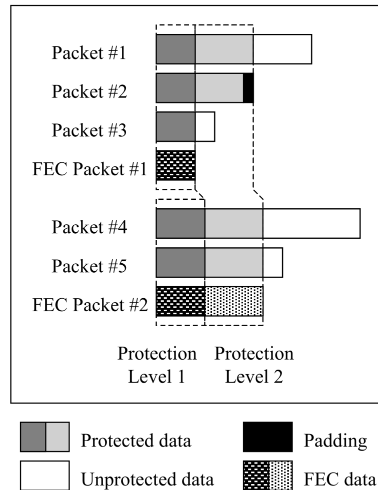
Zoals vermeld in sectie 4.6.2, wordt belangrijke informatie vaak vooraan geplaatst in een pakket. Door het schema uit figuur 4.3 aan te passen zodat enkel de eerste l bits van elk pakket worden betrokken in de berekening van de FEC pakketten, zal de uiteindelijke FEC stroom minder bandbreedte in beslag nemen. Toch kan hiermee de belangrijkste data van de mediastroom hersteld worden. Deze nieuwe vorm van bescherming wordt weergegeven in figuur 4.4. Er moeten in vergelijking met figuur 4.3 minder pakketten worden verlengd met padding. Bescherming van padding bits is verspilling van bandbreedte. In het nieuwe schema beschermen de FEC pakketten veel minder padding bits. De efficiëntie neemt dus toe. De techniek wordt *Single Level ULP* genoemd omdat er slechts één niveau wordt beschermd met een FEC techniek.



Figuur 4.4: *Single level ULP. Bron: [31]*

FEC technieken kunnen op meerdere levels worden toegepast. Figuur 4.5 toont multi-level bescherming. In de figuur zijn er twee levels waarmee vijf pakketten worden beschermd. Er worden twee FEC pakketten berekend. Het eerste FEC pakket zorgt voor level-1 bescherming voor de belangrijkste data uit de eerste drie pakketten. Het tweede pakket zorgt voor level-1 bescherming voor de belangrijkste data van pakketten 4 en 5. Daarbovenop bevat het ook level-2 bescherming voor de iets minder belangrijke data in pakketten 1,2, 4 en 5. Zoals te zien is in de figuur, is het met ULP mogelijk om op een zeer flexibele manier FEC pakketten samen te stellen. Niet elk FEC pakket moet informatie voor alle levels bevatten en elk blok FEC informatie moet niet steeds uit dezelfde hoeveelheid originele pakketten worden berekend. Het payload formaat voor de FEC pakketten laat toe om de ontvanger voldoende te informeren over welke informatie exact aanwezig is in elk FEC pakket. Hoe deze informatie wordt verzameld, is afhankelijk van de toepassing. Het is vooral belangrijk om op te merken dat als FEC pakket 1 en 2 tezamen worden bekeken, er duidelijk meer bandbreedte wordt gebruikt

voor de meer belangrijke level-1 bescherming dan voor level-2.



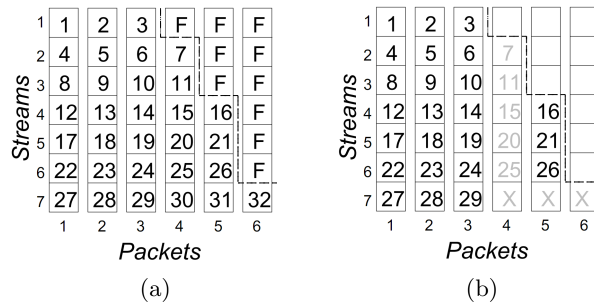
Figuur 4.5: *Multi-level ULP. Bron: [31]*

Net als bij de techniek uit sectie 4.3.3 kunnen FEC pakketten als aparte mediastromen worden verstuurd. Hierdoor geniet ULP dezelfde achterwaartse compatibiliteit. In [31] wordt de performance van ULP getest. De resultaten bevestigen dat door ULP de bandbreedte efficiënter benut wordt, terwijl de informatie nog steeds voldoende sterk beveiligd wordt tegen transmissiefouten.

4.7.2 Het Unequal Loss Protection framework

De volgende vraag werd tot nog toe niet beantwoord: gegeven een media encoder die in zijn uitvoer de data rangschikt van belangrijk naar minder belangrijk, hoe moet FEC worden toegepast om de data zo efficiënt mogelijk te beschermen? In [36] wordt het Unequal Loss Protection framework voorgesteld dat op basis van een model van het pakketverlies op het netwerk een hoeveelheid FEC toekent voor elk stuk uit de datasequentie.

Het framework werd ontwikkeld om een geëncodeerde afbeelding te verdelen in pakketten waarbij ook FEC data wordt toegevoegd. In het bijhorende voorbeeld worden 32 bytes verdeeld over zes pakketten van zeven bytes. Bytes op dezelfde plaats uit de verschillende pakketten worden samen beschermd tegen transmissiefouten. Samen beschermde bytes worden in het framework een *stream* genoemd. Er is plaats voor tien bytes FEC data. Het bijhorende algoritme bepaalt hoeveel originele bytes en hoeveel FEC bytes terecht komen in elke stream. De uiteindelijke verdeling wordt getoond in figuur 4.6(a). Elke rij vormt een stream. Zoals te zien is, worden de eerste bytes van de originele datasequentie met meer FEC bytes beschermd dan de laatste. Elke kolom vormt een van de pakketten die zullen worden verstuurd.



Figuur 4.6: *Het ULP framework. Elke byte uit de oorspronkelijke data wordt met zijn bytenummer aangeduid. Een byte die aangeduid wordt met de letter F is een FEC byte. (a) Verdeling van 32 bytes over zes pakketten van zeven bytes. De eerste bytes, verdeeld over de eerste streams, worden beschermd met het meeste FEC data. (b) Herstel na het verlies van pakket 4. Enkel stream 7 kan niet worden hersteld. Bron: [34]*

Elk verloren pakket zorgt voor een ontbrekende byte in elke stream. In het ULP framework wordt gebruik gemaakt van RS codes om de bytes te beschermen. Eerder in sectie 4.3.2 werd bewezen dat een (n,k) RS code tot $n - k$ symbolen kan herstellen als de posities van de fouten gekend zijn. De posities van verloren bytes zijn gekend aangezien pakketten een sequentienummer dragen. Anders gezegd kan elke stroom met k originele bytes en $n - k$ FEC bytes hersteld worden als k van de n bytes aankomen. Figuur 4.6(b) toont een situatie waarin er een pakket verloren gaat. De eerste zes streams kunnen worden hersteld. De laatste stream bevat te weinig (geen) redundantie. Toch is de datasequentie tot byte 29 bruikbaar.

Het ULP framework dat tot hiertoe werd beschreven, is niet geschikt voor real-time mediastromen. Zoals het werd voorgesteld in [36] moet een volledige afbeelding beschikbaar zijn vooraleer de data kan worden verdeeld in pakketten, en moeten alle pakketten bij de ontvanger zijn aangekomen vooraleer de afbeelding kan worden hersteld. In [18] wordt aangetoond dat het framework kan worden gebruikt voor real-time toepassingen. In het bijzonder wordt video van de H.263 codec met het framework beschermd. Deze codec is speciaal ontwikkeld voor lage-bitrate videoconferencing toepassingen [23].

Het belangrijkste probleem dat moet worden opgelost is het opsplitsen van de videosequentie in korte subsequenties die individueel in pakketten worden verdeeld door het framework. Lange subsequenties zullen betere bescherming bieden tegen fouten (in het bijzonder error bursts), maar impliceren een grotere delay omdat de ontvanger de subsequentie pas kan decoderen wanneer alle data ervan ontvangen werd.

4.7.3 Object-gebaseerde UEP

Een beeld wordt meestal bekeken als een rooster van pixels. Als mens onderscheiden we echter verschillende objecten in een beeld. Sommige objecten staan op de voorgrond terwijl anderen de achtergrond van het beeld vormen. Met object-gebaseerde compressie wordt het mogelijk om visueel belangrijke objecten kwalitatief beter te encoderen dan anderen.

De perceptie van een videosequentie kan gevoelig verbeterd worden door de belangrijkste objecten beter te beschermen tegen transmissiefouten. In de MPEG-4 standaard [25] worden videoscènes expliciet opgebouwd uit een statische achtergrond en verschillende objecten op de voorgrond. In [1] wordt onderzocht hoe verschillende lagen van bescherming kunnen worden gebruikt voor beelden met MPEG-4 compressie.

Met object-gebaseerde UEP kunnen interessante toepassingen worden ontwikkeld. In combinatie met face recognition [67] kan men bijvoorbeeld het gezicht of de features van het gezicht in een videochat applicatie voorzien van extra FEC data. Dit is wenselijk want voor deze applicaties is het gezicht de meest waardevolle informatie.

4.8 Media-specifieke technieken

In de eerder besproken technieken moet een onderscheid gemaakt worden tussen media-onafhankelijke en media-specifieke technieken. Voor een media-onafhankelijke methode is de keuze van de media codec niet belangrijk. Elke vorm van payload data kan worden beschermd met behulp van de techniek. In sommige gevallen moet een media encoder kunnen aangeven welke stukken van zijn uitvoer belangrijker zijn dan anderen, maar elke codec die hier tot in staat is, kan gebruikt worden. Onder deze noemer vallen onder andere de technieken uit sectie 4.3, die volledige pakketten kunnen reconstrueren uit FEC data zonder enige informatie over de gebruikte media codec.

Media-specifieke technieken zijn vaak veel efficiënter in het herstellen van beschadigingen omdat zij gebruik kunnen maken van kennis over de mediastroom en kenmerken van de gebruikte media codec. Deze mogelijkheden zijn tevens het grootste nadeel van deze technieken: ze zijn enkel bruikbaar voor een bepaald type mediastroom. Redundant Audio Coding (sectie 4.4.1) hoort bij deze groep, omdat deze methode enkel bruikbaar is voor audiostromen.

Als voorbeeld van het uitbuiten van de kennis over de media codec wordt de JPEG 2000 standaard [11] bekeken. Deze standaard is een nieuwe compressiestandaard voor afbeeldingen. JPEG 2000 ondersteunt het progressief doorsturen van zeer grote afbeeldingen naar de ontvanger. In [40] wordt een bescherming van de informatie voorgesteld die gebruik maakt van RS codes. De redundante data wordt als Error Protection Block (EPB) rechtstreeks aan de JPEG 2000 bitstroom toegevoegd door middel van de in de

standaard voorziene *markers*. Hierdoor moeten geen extra transportmechanismen gedefinieerd worden. Omdat de exacte opbouw van de JPEG 2000 bitstroom gekend is, kan voor elke byte exact bepaald worden hoe belangrijk die is en met hoeveel redundantie deze beschermd dient te worden. Hier tegenover staat dat de techniek specifiek ontwikkeld is voor JPEG 2000 afbeeldingen en niet kan worden gebruikt in een andere context. De EPB markers worden herkend door de decoder en signaleren zo de aanwezigheid van FEC data. Indien de decoder geen FEC ondersteuning voorziet, zal de marker niet herkend worden en wordt een EPB segment gewoon genegeerd. Dit maakt de techniek achterwaarts compatibel met JPEG 2000 decoders zonder foutcorrectie mogelijkheden.

4.9 Adaptieve FEC

Om te bepalen hoeveel FEC informatie nodig is om een mediastroom met voldoende kwaliteit af te leveren bij de ontvanger, volstaat het om kennis te hebben van enkele netwerkparameters. Als geweten is hoeveel informatie wordt verstuurd en hoeveel informatie verloren gaat tijdens de netwerktransmissie, kan hiermee worden rekening gehouden bij de selectie van de gebruikte FEC code.

Het grote probleem in dit verhaal is dat netwerken niet steeds evenveel pakketten beschadigen. Netwerkparameters kunnen van seconde tot seconde verschillen afhankelijk van de wijzigende belasting van het netwerk of van wijzigende omgevingsfactoren in het geval van draadloze netwerken. Uiteraard is het ongewenst om FEC informatie te versturen als er geen pakketten verloren gaan. Langs de andere kant moet meer bescherming worden toegevoegd als het netwerk plots veel meer transmissiefouten introduceert als voorheen. Systemen die op elk moment zorgen voor een aangepaste hoeveelheid FEC worden adaptieve mechanismen genoemd.

Om de hoeveelheid FEC data te kunnen aanpassen aan de huidige omstandigheden, moet de zender op de hoogte worden gebracht van de hoeveelheid pakketten dat verloren gaat tijdens transmissie. Het RTP protocol beschrijft een geschikte methode die de ontvanger toelaat informatie over de ontvangstkwaliteit aan de zender te rapporteren, namelijk RTCP (RTP Control Protocol). Elke vijf seconden [48] verstuurt een ontvanger een RTCP rapport om de zender te melden hoeveel pakketten er niet werden ontvangen. Deze informatie kan worden gebruikt in adaptieve algoritmes.

4.9.1 Adaptieve algoritmes

In wetenschappelijke kringen werden al verschillende methodes voorgesteld. [5] introduceert het *Bolot algoritme*. Dit algoritme bepaalt het aantal kopieën die per pakket moeten worden verzonden in Redundant Audio Coding (zie sectie 4.4.1). Bolot probeert het pakketverlies na reconstructie met

FEC tussen twee voorgedefinieerde limieten te houden. Na ontvangst van een RTCP pakket wordt het pakketverlies voor en na reconstructie berekend en wordt aan de hand van deze waardes een gepast schema geselecteerd voor de bescherming van de data. Deze schema's zijn vooraf gedefinieerd. Een schema bepaalt hoeveel kopieën er per pakket moeten worden verzonden en aan welke pakketten deze worden toegevoegd met behulp van piggybacking.

Het Bolot algoritme heeft één groot nadeel: in het algoritme wordt gebruik gemaakt van een lijst waardes die proefondervindelijk werden vastgesteld. In [41] wordt gesteld dat deze lijst niet voldoende is opdat het algoritme onder alle netwerkcondities optimaal zou reageren. In dezelfde paper wordt een aangepaste versie voorgesteld, met name het *USF algoritme* (genoemd naar de University of South Florida, waar het werd ontwikkeld), dat geen gebruik meer maakt van deze lijst met waardes en een betere performance weet te halen.

4.9.2 Gezamenlijke media/FEC ratio selectie

Omdat netwerken een beperkte maximale capaciteit hebben, is het niet mogelijk om eender welke hoeveelheid data te versturen. De netwerkcapaciteit moet verdeeld worden over media en FEC data.

Om de capaciteit van een netwerk zoveel mogelijk te benutten, moet er voldoende FEC data worden verstuurd om de mediastroom te beschermen, terwijl de resterende capaciteit volledig wordt gebruikt door die mediastroom. Algoritmes die voor een bepaalde netwerkcapaciteit de meest ideale verhouding tussen media en FEC data berekenen zodat de ontvangst van de stroom zo goed mogelijk is, doen aan gezamenlijke media/FEC ratio controle. [4] beschrijft een dergelijk algoritme voor Internet telefonie toepassingen. Het algoritme baseert zich op de informatie uit RTCP pakketten om een wiskundig model in te vullen dat het pakketverlies benadert. Met dit model kiest het algoritme op elk moment de geschikte verhouding tussen media en FEC informatie.

In [16] wordt een gelijkaardige methode voorgesteld, maar dan voor het beschermen van MPEG-2 [56] video. Tot slot wordt in [15] een techniek voorgesteld die gebaseerd is op het algoritme uit [4], maar media-onafhankelijk is, en niet enkel een model opstelt voor het netwerk, maar dit model tussen het ontvangen van RTCP pakketten met predictie probeert aan te passen. Hiervoor maakt het gebruik van de evolutie van de netwerkparameters tijdens de laatste drie minuten.

Deel II

Implementatie

Hoofdstuk 5

Applicatie

5.1 Doelstellingen

In dit en de hierop volgende hoofdstukken wordt een software implementatie besproken die verschillende technieken uit de literatuurstudie in deel I in de praktijk toepast en vergelijkt.

Om dit doel te kunnen realiseren, werd een applicatie ontwikkeld. De applicatie werd geschreven in C++ en voor een Windows besturingssysteem. Voor deze ontwikkeling werden volgende doelstellingen vooropgesteld:

- De applicatie moet focussen op multimedia. Ander dataverkeer zou ook kunnen worden beschermd, maar dit valt buiten het bereik van dit werk.
- Het transport van de mediastromen moet via het Real-time Transport Protocol (RTP) verlopen.
- De toepassing moet zowel audio- als videostromen ondersteunen.
- Beschermende technieken tegen transmissiefouten moeten dynamisch kunnen worden in- en uitgeschakeld en moeten geparametriseerd zijn.

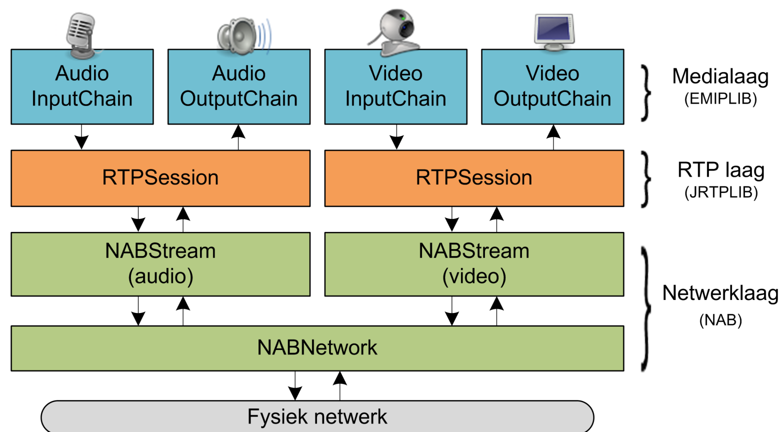
In het vervolg van dit hoofdstuk wordt de opbouw van deze applicatie in detail besproken. De implementatie van de beschermende technieken wordt in hoofdstuk 6 uiteengezet.

5.2 Libraries

In de ontwikkelde applicatie wordt gebruik gemaakt van verschillende bestaande libraries. Dit is een bewuste keuze: libraries versnellen software ontwikkeling en zijn vaak al grondig getest. Het heeft weinig zin om functionaliteit zelf te ontwikkelen als ze al beschikbaar is. Door met libraries te werken, wordt tevens onderzocht hoe moeilijk het is om deze te combineren

met een bescherming tegen transmissiefouten. Met de libraries worden ook andere applicaties ontwikkeld. Daardoor wordt het duidelijk welke stappen moeten worden genomen om enerzijds beschermende code toe te voegen aan dergelijke applicaties, terwijl anderzijds zichtbaar wordt waar de bestaande libraries moeten worden aangepast om de voordelen van het beschermde dataverkeer te benutten.

De applicatie bestaat uit drie verschillende technische lagen. Deze worden voorgesteld in figuur 5.1. De bovenste laag is de medialaag. Deze laag verzorgt de opname en het afspelen van de audio- en videostromen. De middelste laag is verantwoordelijk voor de RTP encapsulatie van de mediadata in RTP stromen. Ze zorgt voor een interface tussen de medialaag en de onderste laag: de netwerklaag. Deze laatste moet ervoor zorgen dat gecreëerde RTP pakketten op een fysiek netwerk terecht komen en tot bij de juiste ontvanger geraken. In elke laag worden verschillende libraries gebruikt, die wat toelichting vereisen. Tenzij anders vermeld, werden deze libraries ontwikkeld aan het Expertisecentrum voor Digitale Media[1] (EDM), een onderzoeksinstituut van de Universiteit Hasselt.



Figuur 5.1: *De verschillende technische lagen van de applicatie.*

EMIPLIB

EMIPLIB[2] (EDM Media over IP library) is een library die de taken in de medialaag voor zijn rekening neemt. Met behulp van een rijk gamma aan media codecs is de library in staat om datastromen te genereren van zowel live opgenomen bronnen afkomstig van een microfoon of een webcamera als van bronnen afkomstig van een opslagmedium. De library bevat een component die een nauwe samenwerking met JRTPLIB mogelijk maakt om transport via RTP te voorzien.

[1] `http://www.edm.uhasselt.be/`
[2] `http://research.edm.uhasselt.be/emiplib/`

JRTPLIB

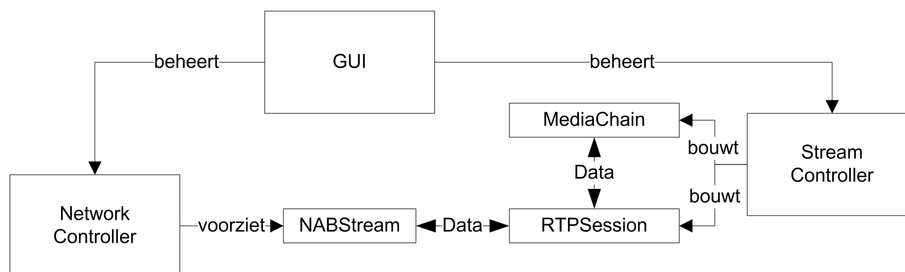
JRTPLIB[3] is een object-georiënteerde implementatie van het RTP protocol volgens de richtlijnen in RFC 3500 [48]. Door het gebruik van deze library is een toepassing in staat aangeleverde multimediastromen te versturen via RTP en worden aspecten als timing, delay- en jittercompensatie en lipsync [43] uit handen van de ontwikkelaar genomen.

NAB

De NAB (Network Abstraction) library abstraheert — zoals de naam doet vermoeden — het netwerk. NAB maakt het mogelijk om via een gemeenschappelijke interface verschillende onderliggende netwerkarchitecturen te benutten voor het fysieke transport van data. NAB ondersteunt onder andere multicast en het client-server model, en is tevens uitbreidbaar om ook andere modellen toe te voegen. De NAB implementatie maakt op zijn beurt gebruik van enkele libraries die multi-platform abstracties vormen voor netwerksockets, waaronder ENUt[4] (EDM Network Utilities) en ENet[5], dat werd ontwikkeld door Lee Salzman.

5.3 Opbouw

De structuur van de applicatie, die werd opgebouwd met de libraries uit de vorige sectie, wordt schematisch weergegeven in figuur 5.2. De applicatie bevat drie belangrijke controlecomponenten. Eén van deze is de `NetworkController`, die de selectie van de gewenste transportmechanismes in de NAB library verzorgt. Daarnaast is er de `StreamController`, die er aan de hand van EMIPLIB en JRTPLIB voor zorgt dat er naargelang de instellingen de gewenste multimediale informatie wordt afgespeeld. Overkoepelend is er de *Graphical User Interface* (GUI) die beide componenten de juiste instructies geeft.



Figuur 5.2: *Structuur van de applicatie.*

[3] http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib
[4] http://research.edm.uhasselt.be/enut/
[5] http://enet.bespin.org/

5.3.1 Netwerk

De NAB library voorziet de applicatie van `NABStream` objecten. Van en naar deze objecten kan informatie gelezen en geschreven worden, net zoals dit het geval is met een netwerksocket. Het verschil met gewone sockets is dat de gebruiker van een `NABStream` object niet moet weten hoe deze objecten het transport verder afhandelen. De `NetworkController` zorgt ervoor dat de juiste `NABStream` objecten worden aangemaakt voor de applicatie. Deze zullen worden gebruikt door de `RTPSession` objecten uit JRTPLIB om de mediastromen te versturen.

De applicatie maakt een duidelijk onderscheid tussen video- en audiostromen, zoals ook te zien is in figuur 5.1. Er worden twee verschillende `NABStream` objecten aangemaakt; de ene transporteert video, terwijl de andere gebruikt wordt voor audio.

Als basis netwerktechnologie werd ervoor gekozen om het transport van de mediastromen in deze applicatie via UDP over multicast te laten verlopen. Met deze techniek moeten alle computers slechts naar hetzelfde multicast IP adres luisteren om deel te nemen aan een mediasessie, waardoor de setup zeer eenvoudig blijft. De multicast adressen en UDP poorten worden geabstraheerd door NAB, zodat noch de applicatie, noch de gebruiker van de applicatie, hiervan op de hoogte moet zijn.

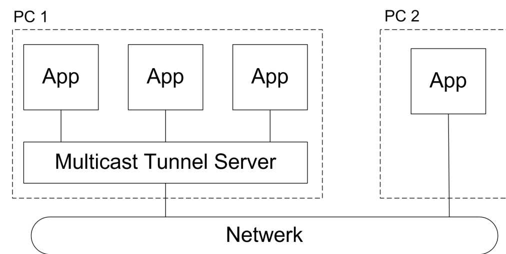
Multicast Tunnel Server

Praktisch gezien heeft het gebruik van multicast één groot nadeel bij de ontwikkeling van software: er kunnen namelijk geen twee instanties van dezelfde applicatie op dezelfde fysieke machine gestart worden, omdat slechts één instantie tegelijkertijd kan luisteren naar een bepaalde UDP poort. Maar net omdat er multicast gebruikt wordt, moeten alle instanties op dezelfde poort luisteren.

Dit probleem wordt opgelost met de Multicast Tunnel Server, die deel uitmaakt van de ENUt library. Deze server wordt tussen de applicatie instanties en het netwerk geplaatst. Applicaties verbinden een willekeurige poort met de server, die op zijn beurt voor hen naar het multicast netwerk luistert. Op die manier moet enkel de server met de multicast poort verbinden en kunnen er meerdere instanties van de applicatie op dezelfde fysieke machine starten. Deze opstelling wordt weergegeven in figuur 5.3.

De Multicast Tunnel Server kan ook in andere opstellingen gebruikt worden. In sectie 8.3 zal bijvoorbeeld blijken dat een applicatie kan verbinden met een server op een andere machine. Deze mogelijkheid kan worden benut wanneer een applicatie via unicast aan een multicast sessie wil deelnemen: de server vangt het multicast verkeer op en dit wordt via unicast met de applicatie uitgewisseld.

Ook de applicatie die werd ontwikkeld in het kader van dit werk kan

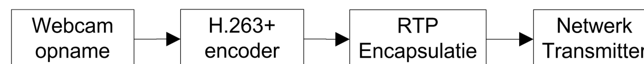


Figuur 5.3: *Gebruik van de Multicast Tunnel Server. Op PC 1 worden drie instanties van de applicatie gestart. De server maakt het mogelijk dat ze alle drie met het netwerk kunnen communiceren. Op PC 2 wordt slechts één instantie gestart die rechtstreeks met het netwerk kan verbinden.*

verbonden worden via een Multicast Tunnel Server. De NAB library handelt de details van deze verbinding af.

5.3.2 Media

De EMIPLIB library heeft een flexibele architectuur om media processing te voorzien in een applicatie. Er wordt gewerkt met `MIPComponentChain` objecten. Deze objecten representeren een ketting van componenten, elk met een zeer specifieke taak, die samen zorgen voor de volledige afhandeling van een bepaald type medium. Figuur 5.4 bevat een vereenvoudigd voorbeeld van een `MIPComponentChain`. In dit voorbeeld wordt gebruik gemaakt van een component die de beelden van een webcamera inleest, een component die deze invoer vervolgens encodeert met behulp van een video codec, een component die zorgt voor RTP encapsulatie en tot slot een component die de informatie vervolgens op het fysieke netwerk kan plaatsen.



Figuur 5.4: *Een vereenvoudigde EMIPLIB componentketting.*

In de doelstellingen werd gesteld dat zowel audio als video moet worden gebruikt in de applicatie. Om dit te kunnen verwezelijken, bevat de applicatie vier EMIPLIB componentkettingen, met name `VideoInputChain`, `VideoOutputChain`, `AudioInputChain` en `AudioOutputChain`. De exacte implementatie van de kettingen is in de context van dit werk niet erg van belang en wordt samengevat in bijlage C.

Beide invoerkettingen kunnen geconfigureerd worden om live data afkomstig van een microfoon of een webcamera te verwerken, of kunnen gebruik maken van invoerbestanden. In het laatste geval ondersteunt de `AudioInputChain` Waveform audiobestanden. De `VideoInputChain` maakt

gebruik van de FFmpeg video tools[6] en kan bijgevolg bestanden uitlezen die werden gecomprimeerd met een van de vele door FFmpeg ondersteunde video codecs. Door ondersteuning te bieden voor invoerbestanden is het eenvoudiger om de applicatie te testen en is het bezit van webcamera of microfoon niet langer een vereiste.

Invoerdata wordt vervolgens gecomprimeerd om efficiënt over het netwerk verstuurd te worden. Voor audio wordt de Speex [57] spraakcodec gebruikt. Om de beste kwaliteit te behouden, werd gekozen voor de *Ultra-Wideband* compressie met een 32kHz sampling ratio. Videobeelden worden met de FFmpeg implementatie van de H.263+ codec [23] gecomprimeerd. Zowel de grootte van de beelden als het bitrate van de codec kunnen in de applicatie worden aangepast.

Aanpassingen

Om alle vermelde functionaliteit te voorzien in de applicatie, zijn er enkele aanpassingen gebeurd in EMIPLIB. De belangrijkste hebben betrekking tot de invoer en verwerking van video.

Omdat de Windows-versie van EMIPLIB geen videobestanden kon inlezen, werd een `MIPAVFileDecoder` component ontwikkeld. Deze is in staat om videobestanden te decoderen op basis van de functionaliteit van FFmpeg en de beelden toe te voegen aan de componentketting voor verdere verwerking. De component is zeer flexibel: de grootte van de beelden kan worden aangepast naar het gewenste formaat en de gewenste framerate moet niet overeenkomen met de framerate van het invoerbestand. De huidige implementatie kan enkel videobestanden inlezen die gebruik maken van het AVI containerformaat [35].

EMIPLIB maakte al gebruik van de FFmpeg library. De versie was echter te verouderd om bovenstaande wijzigingen mogelijk te maken. Bijgevolg werd de EMIPLIB code aangepast om gebruik te maken van een nieuwere FFmpeg versie. Dit impliceert tevens dat EMIPLIB nu kan rekenen op de kracht van de nieuwe `SwScale` software scaling module voor het converteren van pixelformaten en resoluties. Deze module vervangt de verouderde `img_convert` met efficiëntere en meer uitgebreide functionaliteit.

5.3.3 RTP encapsulatie

De connectie tussen de medialaag en het netwerk wordt, zoals eerder werd vermeld, verzorgd via JRTPLIB. Uit deze library wordt een `RTPSession` object geïnstantieerd voor elk `NABStream` object. Omdat voor audio en video verschillende `NABStream` objecten worden gebruikt, zullen er ook twee `RTPSession` objecten in de applicatie zijn (indien beide mediatypes actief zijn).

[6] `http://ffmpeg.org/`

Elk `RTPSession` object is verantwoordelijk voor zowel het verzenden als het ontvangen van zijn type medium. In de medialaag zijn er echter twee componentkettingen voorzien per mediatype. Dit vormt geen probleem: zowel de invoerketting als de uitvoerketting kunnen tegelijkertijd interageren met hetzelfde `RTPSession` object.

5.3.4 Graphical User Interface

De gebruiker kan de applicatie aansturen met behulp van een *Graphical User Interface* (GUI). Een belangrijke doelstelling bij het ontwikkelen van deze component was zijn generaliteit. Met de GUI moet de gebruiker in staat zijn om de verschillende instellingen op het gebied van netwerk, media en FEC bescherming te wijzigen.

Alle instellingen worden bij het afsluiten persistent bewaard, zodat de applicatie bij het heropstarten de vorige instellingen opnieuw kan laden.

Bijlage B bevat enkele screenshots die een overzicht geven van de GUI en zijn mogelijkheden.

Hoofdstuk 6

Bescherming in de praktijk

6.1 Architectuur

In het vorige hoofdstuk werd uiteengezet hoe een typische multimedia applicatie kan worden ontwikkeld. De voorgestelde applicatie heeft een gelijkaardige opbouw als elke andere applicatie met gelijkaardige doelstellingen die gebruik zou maken van de besproken libraries, en is bijgevolg representatief voor al deze applicaties.

Om in een dergelijke applicatie bescherming tegen transmissiefouten toe te voegen, moet er extra verwerking van de dataflow worden voorzien in het lagenmodel dat werd besproken in sectie 5.2. De beste plaats om deze functionaliteit toe te voegen zit tussen de RTP laag en de netwerklaag: tussen deze lagen worden volledige RTP pakketten uitgewisseld. Hoe deze beschermende tussenlaag werkt, wordt toegelicht in de volgende secties.

In dit deel wordt verondersteld dat de lezer over een grondige kennis van het Real-time Transport Protocol beschikt. [43] wordt aangeraden als referentiewerk voor dit protocol.

6.1.1 Onderscheppen van RTP pakketten

Uit de beschrijving van de opbouw van de applicatie (sectie 5.3) blijkt dat een `RTPSession` object pakketten uitwisselt met het netwerk via een `NABStream` object. Een beschermende laag moet deze uitwisselingen kunnen onderscheppen. Tijdens de ontwikkeling van deze laag werd getracht de bestaande publieke interface van de `RTPSession` klasse niet te wijzigen, zodoende dat bestaande applicaties niet moeten worden aangepast.

Om RTP pakketten te onderscheppen, werd een nieuwe klasse geïntroduceerd: de `ProtectingRTPSession`. Deze klasse is afgeleid van `RTPSession`, zodat ze dezelfde functionaliteit aanbiedt aan een applicatie. De interne structuur van `RTPSession` werd echter zo aangepast dat afgeleide klassen het versturen en ontvangen van RTP pakketten kunnen manipuleren. De klasse werd hiervoor uitgebreid met twee functies:

- `PreprocessPolledData()` wordt aangeroepen net nadat pakketten van de netwerklaag werden geplukt en net vooraleer deze pakketten toegevoegd worden in de interne structuren van het `RTPSession` object.

- `ProcessSendPacket(...)` wordt aangeroepen door de `SendPacket()` functies van een `RTPSession` object net nadat het pakket volledig is samengesteld en net voor het wordt doorgegeven aan de netwerklaag.

Door deze twee bypasses te implementeren in een afgeleide klasse zoals de `ProtectingRTPSession`, kan de RTP pakketstroom worden beschermd tegen transmissiefouten.

6.1.2 Een generieke interface

In de literatuurstudie van dit werk werden zeer uiteenlopende technieken betreffende bescherming tegen transmissiefouten beschreven. Om vlot van techniek te kunnen wisselen, is het voordelig om met een algemene interface te werken waarmee alle implementaties van technieken op dezelfde manier kunnen worden aangesproken. Er is een aparte interface voor encoders (die redundante data toevoegen ter bescherming van de pakketstroom) en decoders (die na het ontvangen redundante data verwijderen en eventueel gebruiken om herstellingen op de beschadigde pakketstroom uit te voeren).

Voor encoders gelden de volgende regels:

- Een encoder moet in staat zijn een pakket aan te passen en uit te breiden met redundante informatie.

- Een encoder moet de mogelijkheid hebben extra pakketten te introduceren in de pakketstroom.

Decoders hebben gelijkaardige vereisten:

- Een decoder moet redundante informatie uit pakketten kunnen verwijderen.

- Een decoder moet pakketten kunnen herstellen of vervangen.

- Bepaalde pakketten moeten worden gesplitst. Te denken valt aan technieken waarbij piggybacking (sectie 4.4.2) wordt gebruikt.

- Extra pakketten moeten mogelijk worden aangemaakt uit redundante data. Ook deze pakketten moeten hun weg vinden naar de applicatie.

Figuur 6.1 toont de interfaces waarmee deze doelen bereikt worden. Deze zijn gedefinieerd in `Protection.h`. Een encoder krijgt pakketten aangeboden door de `process(...)` functie. De parameters van deze functie zijn

```
class ProtectionEncoder
{
virtual void process(void** p, size_t* len) = 0;
virtual bool pull(void** p, size_t* len) = 0;
};

class ProtectionDecoder
{
virtual void push(RTPRawPacket* p) = 0;
virtual RTPRawPacket* pull(void) = 0;
};
```

Figuur 6.1: *Interface voor implementatie van bescherming.*

in-out parameters. Een pakket kan dus volledig vervangen worden. Extra pakketten die worden gegenereerd, moeten gebufferd worden en kunnen uit de encoder uitgelezen worden met de `pull(...)` functie.

De decoder interface heeft een iets andere werking: een decoder krijgt pakketten aangeboden via zijn `push(...)` functie, maar kan ze niet onmiddellijk wijzigen. Aangeboden pakketten worden echter uit de rest van de applicatie verwijderd. Het is de verantwoordelijkheid van de decoder om in zijn uitvoerbuffer de juiste pakketten te plaatsen. Ook als een decoder niets met het pakket kan doen, moet het gewoon in de buffer gekopieerd worden, zodat het niet verloren gaat. Met deze werkwijze heeft de decoder de volledige controle over zijn uitvoer. De uitvoerbuffer kan pakket per pakket worden uitgelezen door opeenvolgende aanroepen van de `pull(...)` functie van de decoder.

6.1.3 Verwerking van pakketstromen

Het gebruik van één enkele beschermingstechniek is eenvoudig: in de implementatie van de bypasses moet enkel het juiste encoder/decoder paar worden geïnstalleerd.

Er werd echter gekozen voor een meer algemene aanpak, waarin verschillende beschermingen achter elkaar kunnen worden geplaatst. Hierdoor kunnen concatenated codes (sectie 3.7.5) worden gevormd. Dit is zeer interessante functionaliteit. Zo wordt het bijvoorbeeld mogelijk om enerzijds een bescherming tegen pakketverlies te voorzien met de parity code (sectie 6.2) en deze te combineren met de Reed-Solomon bescherming (sectie 6.4) tegen bitfouten.

Het idee achter de verwerking van pakketten is analoog aan het idee van de componentkettingen in EMIPLIB, dat in sectie 5.3.2 werd beschreven. Aan een `ProtectingRTPSession` kunnen verschillende encoders en deco-

ders worden toegevoegd. De encoders zullen in volgorde van toevoeging een pakket één voor één mogen verwerken. Hiervoor wordt de `process(...)` functie van elke encoder aangeroepen. Elke encoder krijgt het pakket aangeboden zoals de vorige encoder in de ketting het heeft afgeleverd.

Ook extra pakketten, die worden gegenereerd door encoders, verdienen een goede bescherming tegen transmissiefouten. Nadat een origineel pakket volledig werd verwerkt en verstuurd, worden de extra pakketten van alle encoders via aanroepen van hun `pull(...)` functie verzameld. Al deze pakketten worden vervolgens door de ketting van encoders gehaald om ook deze extra bescherming te geven.

Een encoder is zelf verantwoordelijk voor de selectie van de pakketten die hij kan beschermen. Wanneer bijvoorbeeld een redundant pakket dat afkomstig is van de parity code encoder, wordt verwerkt door de ketting van encoders, zal dit op een bepaald moment aan de parity code encoder worden aangeboden. Een encoder mag echter nooit zelf gegenereerde pakketten beschermen om oneindige lussen te vermijden. In dit geval moet de encoder zelf beslissen dat het aangeboden pakket niet beschermd kan worden en gewoon aan de volgende encoder moet worden aangeboden.

Een alternatieve implementatie kan pakketten afkomstig van een bepaalde encoder enkel nog aanbieden aan encoders die later voorkomen in de encoder ketting. Op die manier worden implementatiefouten, die tot dergelijke oneindige lussen leiden, vermeden.

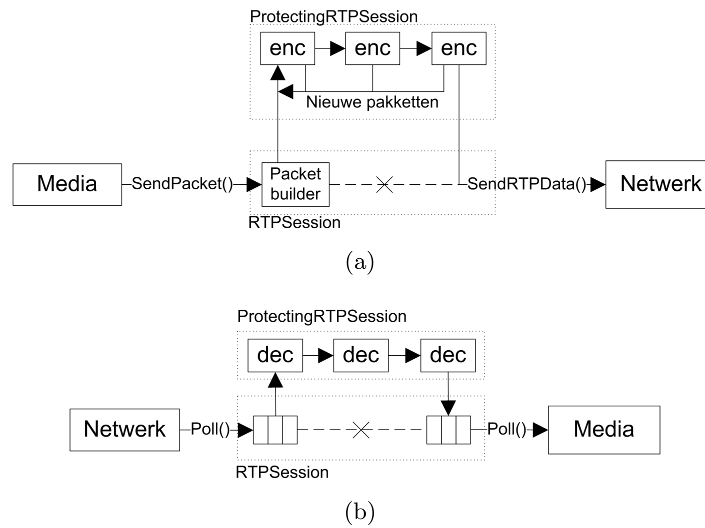
Voor het decoderen geldt een gelijkaardig verhaal: een pakket doorloopt een ketting van decoders. Elke decoder zal als invoer de uitvoer van de vorige decoder in de ketting ontvangen. Met de `pull(...)` functie worden pakketten uit de vorige decoder gehaald, om vervolgens te worden doorgegeven aan de volgende decoder in de ketting met behulp van zijn `push(...)` functie.

Het volledige verwerkingsproces dat werd toegevoegd aan JRTPLIB, wordt nog eens samengevat in figuur 6.2. De volgende secties bespreken de geïmplementeerde beschermingstechnieken.

6.2 Pakketbescherming met parity code

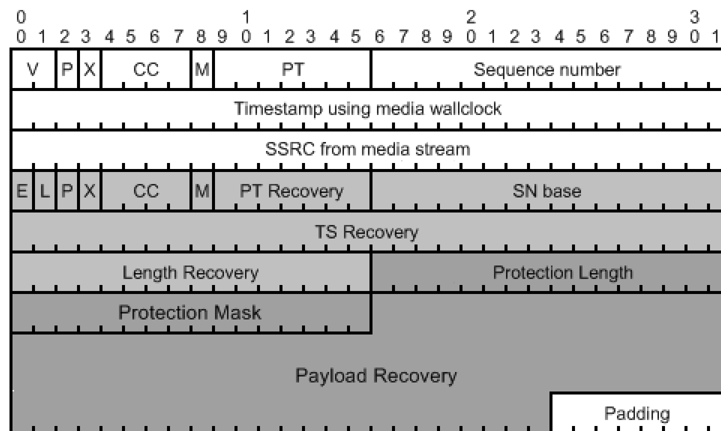
De eerste techniek die werd geïmplementeerd ter bescherming van RTP stromen is de *parity code* uit sectie 4.3.1. Deze code combineert pakketten met behulp van een eenvoudige per-bit XOR operatie en compenseert het verlies van pakketten. De implementatie volgt de richtlijnen in RFC 5109 [30]. Per groep van x pakketten wordt een redundant pakket aan de informatiestroom toegevoegd. Zoals besproken in de literatuurstudie, kan met deze techniek per groep één verloren pakket volledig gereconstrueerd worden.

In figuur 6.3 wordt het gebruikte formaat voor de redundante pakketten voorgesteld. Dit formaat bestaat uit drie blokken: een header voor het



Figuur 6.2: *Het onderscheppen en beschermen van RTP pakketten. (a) het encoder proces; (b) het decoder proces.*

pakket, een blok waarmee de header van een verloren pakket kan worden gereconstrueerd en een blok om de payload te reconstrueren. De verschillende blokken worden in de figuur met een andere grijswaarde ingekleurd.



Figuur 6.3: *Het formaat van een redundant pakket met XOR informatie.*

De header voor een redundant pakket leent veel informatie uit de oorspronkelijke mediastroom. De *Synchronization Source* (SSRC) is dezelfde, zodat een redundant pakket gelinkt wordt met de juiste mediainformatie, en voor de berekening van de *timestamp* wordt de klok van de mediastroom gebruikt. Redundante pakketten hebben hun eigen *Payload Type (PT)* en sequentienummer om zich te onderscheiden van de mediapakketten.

Het tweede blok bestaat in de eerste plaats uit velden die het recon-

strueren van verschillende header velden mogelijk maken. Voor deze velden wordt het resultaat van de XOR operatie opgenomen in dit blok. Bijvoorbeeld: om het *PT Recovery* veld te vormen, worden de *PT* velden van de originele pakketten met behulp van de XOR operatie gecombineerd. Verder is er nog een veld waarmee de lengte van een verloren pakket kan worden gereconstrueerd (*Length recovery*). Dit is het resultaat van de XOR operatie op de lengtes van alle originele pakketten. Dit is nodig wanneer pakketten van verschillende lengte worden gecombineerd. De lengte van een verloren pakket zou niet gekend zijn zonder dit veld.

Om te weten welke pakketten werden gebruikt om een redundant pakket te vormen, bevat dit laatste twee velden: de sequentienummer basis (*SN base*), waarin het laagste sequentienummer van de groep gecombineerde mediapakketten wordt bewaard, en een pakketmasker *(Protection mask)*. In dit masker staat de i-de bit op 1 als en slechts als het pakket met sequentienummer (SN base)$+i$ werd gebruikt om het redundante pakket te vormen. Omdat er een 16-bit masker wordt gebruikt, kunnen tot maximaal zestien pakketten bijdragen tot een redundant pakket. De huidige implementatie combineert enkel opeenvolgende pakketten. In [30] wordt een extensie van het pakketmasker tot 48-bit besproken. Deze werd niet opgenomen in de implementatie omdat uit de experimenten in sectie 9.3.1 blijkt dat het combineren van meer dan zestien pakketten enkel nuttig is bij zeer kleine pakketverliezen. In dit geval zou een pakket veel te laat worden gereconstrueerd omdat het te lang duurt vooraleer alle benodigde pakketten voor reconstructie ontvangen zijn. Daardoor kan de inhoud van het pakket niet meer tijdig worden afgespeeld in een real-time applicatie.

Het derde blok in het pakketformaat bevat het resultaat van de per-bit XOR operatie van de payloads van de originele pakketten. Het *Payload Recovery* veld wordt echter voorafgegaan door twee andere velden die volgens de kleurcode tot hetzelfde blok behoren. Het *Protection Mask* veld werd eerder al besproken. In *Protection Length* wordt de lengte van het *Payload Recovery* veld bewaard. Deze twee velden horen volgens RFC 5109 bij het derde blok omdat het formaat in deze RFC ondersteuning biedt voor Unequal Error Protection. Voor elk beschermingsniveau wordt een apart *Payload Recovery* veld gebruikt dat kan samengesteld zijn uit andere pakketten dan het *Payload Recovery* veld van een ander niveau. Bijgevolg moet elk blok een eigen *Protection Mask* hebben. Met de *Protection Length* velden kunnen de verschillende niveaus uit elkaar gehouden worden. Het aangeven van deze lengte is in de code horende bij dit werk overbodig, omdat UEP niet werd geïmplementeerd.

Voor- en nadelen

Het grote voordeel van de parity code is zijn algemene inzetbaarheid. De techniek werkt onafhankelijk van het type informatie dat moet worden be-

schermd. Dit mag video, audio of eender welk mediatype zijn, zolang het transport maar via RTP verloopt. De methode verzekert ook een achterwaartse compatibiliteit. Omdat aparte pakketten met een eigen PT worden gebruikt voor de redundante informatie, kunnen ze zeer eenvoudig worden genegeerd door decoders die niet begrijpen wat ze met de pakketten moeten doen. De mediapakketten arriveren ongewijzigd en zullen gewoon kunnen worden gedecodeerd.

Het nadeel van deze techniek is dat ze vrij veel extra bandbreedte verbruikt. Elk redundant pakket introduceert zijn eigen RTP-, IP- en transportheaders. Verder is veel padding nodig als pakketten met een zeer variërende lengte moeten worden gecombineerd. Dit wordt verduidelijkt in figuur 4.3, waarin bij onder andere het derde pakket zeer veel padding moet worden toegevoegd. Vooral bij het beschermen van video kan dit gevoelige gevolgen hebben, omdat het verschil in lengte tussen I-frames en P- of B-frames aanzienlijk kan zijn.

6.3 Redundant Audio Coding

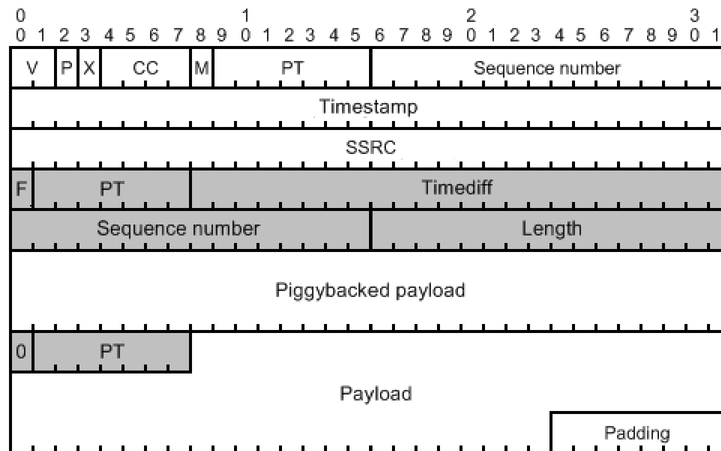
De theorie betreffende de Redundant Audio Coding techniek werd in detail uiteengezet in sectie 4.4.1. Samengevat komt het erop neer dat een pakket met audio meerdere keren wordt verstuurd. Indien een origineel pakket verloren gaat, kan het worden vervangen door een redundante versie.

In de implementatie werd getracht deze techniek toe te passen op audio gecomprimeerd met de Speex codec [57]. Er wordt voor elk origineel pakket één redundante kopie gemaakt. Deze wordt met een bepaalde delay verstuurd. Piggybacking (sectie 4.4.2) wordt gebruikt voor het transport van de redundante informatie. De delay wordt bepaald door een parameter die aangeeft hoeveel pakketten na het originele pakket er moet worden gewacht vooraleer de kopie wordt verstuurd met behulp van piggybacking.

6.3.1 Pakketformaat

Het pakketformaat is afgeleid van het formaat dat wordt voorgesteld in RFC 2198 [45]. Dit formaat heeft echter zijn beperkingen: elk blok data dat wordt meegestuurd met een ander pakket kan volgens dit formaat slechts een maximale lengte van 1024 bytes hebben en het is niet mogelijk om dit blok te voorzien van een sequentienummer. Daarom werd het formaat wat afwijkend geïmplementeerd. Hierdoor gebruikt het formaat net iets meer bandbreedte voor header informatie dan origineel was voorzien in de RFC.

Het formaat wordt afgebeeld in figuur 6.4. Met het “basispakket voor piggybacking” wordt in deze uitleg het pakket bedoeld dat de redundante data zal meenemen tijdens transport. Als RTP header wordt exact de header van een basispakket gebruikt, maar het PT wordt vervangen, zodat een ontvanger dit pakket kan onderscheiden als een pakket met piggybacking.



Figuur 6.4: *Het formaat van een pakket met piggybacked informatie*

Het pakket bestaat verder uit verschillende blokken data met elk een eigen blok header. Deze header begint met een F-bit, die steeds 1 is, behalve voor het laatste blok. In dezelfde byte wordt met een PT aangegeven welk type data in dit blok aanwezig is.

Het laatste blok (voorafgegaan door een F-bit met waarde 0) bevat de payload van het basispakket. De andere blokken hebben enkele extra velden in hun blok header. Deze velden bevatten informatie om te bepalen voor welk origineel pakket dit blok een redundante kopie vormt. Opvallend is het *timediff* veld. Dit veld bevat het verschil tussen de timestamp van het basispakket (terug te vinden in de RTP header) en de timestamp van het gekopieerde pakket. Er kan gewerkt worden met een verschil omdat beide pakketten in tijd niet zeer ver uit elkaar zullen worden verstuurd bij de Redundant Audio Coding techniek. In het oorspronkelijke pakketformaat uit [45] werd zo getracht met een minimale overhead toch een timestamp aan de redundante data te koppelen. Deze doelstelling valt wat weg door de uitbreiding van het formaat met een volledig lengte- en seqentienummerveld. Het *timediff* veld werd tien bit langer om *byte alignment* te verzekeren. Toch is dit met drie bytes nog steeds korter dan een kopie van een timestamp, wat vier bytes in beslag zou nemen.

De aanwezigheid van een sequentienummer voor de redundante data is belangrijk voor applicaties die steunen op JRTPLIB en EMIPLIB. Uit het lagenmodel uit sectie 5.2 is geweten dat JRTPLIB bij het ontvangen van een RTP pakket informatie zal doorgeven aan EMIPLIB. De implementatie vereist dat een volledig RTP pakket wordt doorgegeven, met een juist sequentienummer en timestamp. Zoniet kan EMIPLIB geen juiste afspeeltijd berekenen voor de inhoud van het pakket. Als redundante informatie uit piggybacked pakket moet worden doorgegeven aan EMIPLIB, moet de payload bijgevolg voorzien worden van de juiste RTP header, inclusief een

correct sequentienummer.

Het formaat, zoals het tot hiertoe beschreven werd, zou kunnen worden geoptimaliseerd zodat het minder bandbreedte vereist. Uit de praktijk blijkt dat een payload van een audiopakket nooit de grens van 1024 bytes overschrijdt. De manier waarop in [45] het *length* veld wordt gebruikt (door het voor te stellen met slechts 10 bits, die op de plaats staan van de laatste 10 bits van het *timediff* veld, dat op zijn beurt 10 bits korter is) volstaat dus voor Redundant Audio Coding. Ook zou het sequentienummer, net als de timestamp, kunnen worden voorgesteld als een verschil ten opzichte van het sequentienummer van het basispakket. Zo kunnen in totaal drie bytes per piggybacked datablok worden uitgespaard. Door deze optimalisaties verliest het piggybacking formaat echter zijn generaliteit. De optimalisaties kunnen gebruikt worden in de context van Redundant Audio Coding, maar voor eventuele andere technieken die van piggybacking gebruik willen maken, kan het te beperkend zijn.

6.3.2 Beperkingen van Speex

Zoals beschreven staat in sectie 4.4.1, is de Redundant Audio Coding techniek zo efficiënt omdat redundante kopieën met een lagere kwaliteit kunnen worden gecomprimeerd. Met Speex zijn er twee opties om het bandbreedte verbruik te verlagen:

- Door het gebruik van een andere sampling rate moet er minder data worden gecomprimeerd. Speex ondersteunt drie verschillende sampling rates.
- Speex heeft een kwaliteitsparameter die de sterkte van de compressie bepaalt.

De eerste optie werd getest, maar is geen goed idee. Verschillende sampling rates vereisen verschillende Speex decoders. Dit wordt niet voorzien in EMIPLIB. Daarnaast wordt ook de timing in de war gebracht. Het verschil tussen timestamps van twee opeenvolgende pakketten is normaal gelijk aan het aantal samples in een pakket. Het mengen van verschillende sampling rates maakt het correct berekenen van de juiste afspeeltijd onmogelijk. Daarom werd geopteerd voor het gebruik van de kwaliteitsparameter van de Speex encoder om de bandbreedte nodig voor de redundante kopieën te verkleinen.

Om de kwaliteit van de compressie te laten verschillen in kopieën, is een tweede Speex encoder nodig. Hierdoor ontstaan er problemen bij de ontvanger. Een Speex decoder houdt een status bij tussen het verwerken van verschillende pakketten. Indien een pakket wordt vervangen door een pakket van een encoder met andere parameters, zal de inter-pakket status van de

decoder niet langer correct zijn. Bijgevolg ontstaan er in de resulterende klank artefacten die meer storen dan het ontbreken van een pakket.

Enkele oplossingen werden onderzocht om toch gebruik te kunnen maken van Redundant Audio Encoding in combinatie met een Speex encoder. Een eerste maakt gebruik van twee decoders. Bij het verwerken van een ontvangen pakket, zou dit pakket worden opgesplitst in twee pakketten: het eerste vormt het basispakket waarvan sprake in sectie 6.3.1, het tweede vormt een pakket gegenereerd uit redundante data. Beide worden aangeboden aan de EMIPLIB componentketting in figuur C.2. Opdat EMIPLIB de pakketten zou kunnen onderscheiden, wordt voor een redundant pakket de *marker bit* in de RTP header op 1 gezet. Deze wordt toch niet gebruikt door Speex zelf. Beide pakketten worden vervolgens gedecodeerd. Om dit te verwezenlijken wordt de `MIPSpeexDecoder` in de componentketting vervangen door een `MIPMultiSpeexDecoder`. Deze component zal zowel de originele als de redundante pakketten decoderen, zodat er twee decoders ontstaan met een correcte inter-pakket status. Als de decoder van originele pakketten geen uitvoer genereert, zal de uitvoer van de redundante decoder worden gebruikt. Dit systeem werkt goed op voorwaarde dat er niet té veel pakketten verloren gaan. Wordt het pakketverlies te groot, dan ontstaan er echter twee decoders met een foutieve status, die elkaar proberen aan te vullen. Dit is niet erg bevorderlijk voor de uiteindelijk waargenomen audio.

Een andere oplossing laat de vereiste van een lagere kwaliteit voor redundante kopieën vallen. Kopieën kunnen dan wel gebruikt worden door de decoder van de oorspronkelijke audiostroom. In dit geval vervalt Redundant Audio Coding in een techniek die niet meer doet dan een audiostroom twee keer versturen. Door de piggybacking blijft de techniek toch nog wat bandbreedte besparen omdat minder pakketten worden verstuurd en minder header informatie nodig is dan wanneer elk pakket zou worden gedupliceerd.

6.4 Reed-Solomon bitbescherming

De technieken in de vorige secties proberen de effecten van pakketverlies te verkleinen. Deze sectie bespreekt het implementatiegedeelte dat bescherming tegen bitfouten moet toevoegen. De bescherming is gebaseerd op de Reed-Solomon code, die grondig werd besproken in sectie 3.6. Deze code werd gekozen omdat ze wijdverspreid is en algemeen aanvaard wordt als een degelijke code met efficiënte en geteste bestaande implementaties.

De RS bescherming wordt op volledige RTP pakketten toegepast, zodat niet alleen payload, maar ook alle header informatie wordt beschermd tegen bitfouten. Veronderstel een (n,k) RS code. Om een pakket te encoderen, moet het worden gesplitst in blokken van elk k symbolen. Indien het aantal symbolen in het pakket geen veelvoud is van k, moet eerst padding worden toegevoegd. Dit gebeurt volledig volgens de richtlijnen van RTP: de pad-

ding bit in het pakket wordt op 1 gezet, het pakket wordt verlengd met de benodigde hoeveelheid bytes met waarde 0 en de laatste byte van het pakket — dewelke zelf tot de padding behoort — wordt gebruikt om aan te duiden hoeveel bytes padding werden toegevoegd. De RS decoder zal deze padding niet verwijderen. Dit is geen probleem omdat de RTP decoder deze padding begrijpt en zelf kan verwijderen.

6.4.1 RS implementatie

Er werd gekozen om gebruik te maken van een open-source implementatie van de RS code. Twee libraries werden onder de loep genomen. De eerste, RSCODE[1] genaamd, werd al snel te beperkt gevonden voor dit werk: deze library ondersteunt enkel symbolen met een lengte van acht bits, zodat de n parameter steeds 255 is. De k parameter kan wel gekozen worden, maar deze moet vastgelegd worden tijdens het compileren en kan dus niet *at-runtime* worden gewijzigd. Vooral deze laatste restrictie werd als te beperkend beschouwd.

Een tweede library, de Schifra[2] library, laat wel toe om tijdens de uitvoering van de applicatie verschillende parameters te gebruiken. Hierdoor ontstaat een veel flexibelere applicatie. Toch heeft ook deze library zijn beperkingen. Net als bij de RSCODE library moeten parameters gekend zijn tijdens het compileren. Hierdoor kan de code geoptimaliseerd worden voor een snelle uitvoering. Het grote verschil met RSCODE zit in het feit dat Schifra toelaat meerdere RS encoders/decoders met verschillende parameters te compileren. Door hierrond een interface klasse te voorzien, werd in de applicatie een dynamische structuur opgebouwd, die naargelang de parameters van de gebruiker de juiste encoder/decoder selecteert. Hierdoor wordt de efficiëntie van compiler optimalisatie gecombineerd met de flexibiliteit van at-runtime parameters.

6.4.2 Selectie RS codes

Er werd onderzocht welke (n,k) parameters het meest bruikbaar zijn voor de bescherming van een RTP pakket. Er moet rekening gehouden worden met de grootte van de pakketten. Er is geweten uit vergelijking 3.14 dat de waarde voor n samenhangt met de grootte m van de symbolen. Tabel 6.1 toont voor verschillende waardes van m wat de n parameter voor de RS encoder wordt en hoeveel bandbreedte een resulterend codewoord in beslag zal nemen.

Een grotere m-waarde impliceert een grotere n-waarde. Het heeft geen zin om kleine pakketten te beschermen met een encoding met een grote m-waarde. Ter illustratie: een audiopakket uit de applicatie kan variëren van 27

[1] `http://rscode.sourceforge.net/`

[2] `http://www.schifra.com/`

m	n	Bits per codewoord	Bytes per codewoord
3	7	21	2,625
4	15	60	7,5
5	31	155	19,375
6	63	378	47,25
7	127	889	111,125
8	*255*	*2040*	*255*
9	511	4599	574,875
10	1023	10230	1278,75

Tabel 6.1: *Bandbreedte voor verschillende RS codewoorden.*

tot 136 bytes. Als symbolen van tien bits zouden worden gebruikt, neemt een resulterend codewoord afgerond 1279 bytes in beslag (zie tabel 6.1). Het is weinig efficiënt om elk pakket van 27 bytes te beschermen met een dergelijk groot codewoord. Videostromen kennen een veel groter bereik van pakketgrootte, maar de pakketten zijn over het algemeen groter dan die van audiostromen. Afgaand op de grootte van de audiopakketten, werd eerst geëxperimenteerd met een symboolgrootte van zes bits. Een resulterend codewoord heeft in dit geval 48 bytes nodig om verstuurd te worden. Dit is klein genoeg om padding overhead bij kleine pakketten te beperken.

Metingen wezen echter uit dat de Schifra library met een symboolgrootte van zes bits zeer traag rekent. Aangezien de applicatie real-time mediastromen moet beschermen, is deze symboolgrootte niet bruikbaar. De library heeft een veel hogere efficiëntie wanneer een symboolgrootte van acht bit wordt gebruikt. In dit geval wordt er gerekend met bytes, het geen veel "natuurlijker" is voor processors, zodat real-time verwerking van datablokken ter grootte van RTP pakketten mogelijk wordt.

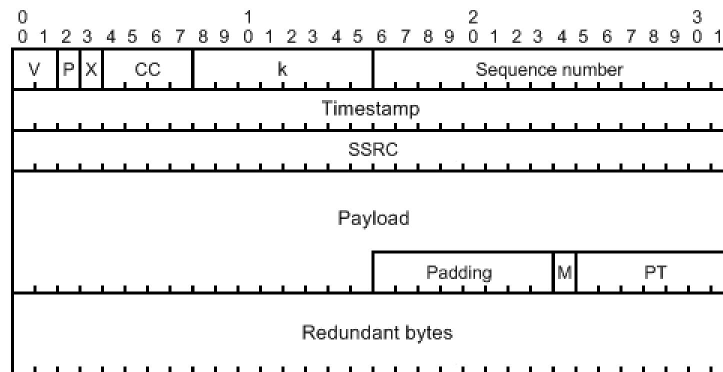
Het gebruik van 8-bit symbolen zorgt voor een grote overhead bij kleine pakketten. Om te vermijden dat bij deze pakketten vooral padding bits worden beschermd, werd met het idee gespeeld om de RS encoder automatisch een encoding te laten selecteren met een k-waarde die zo dicht mogelijk aansluit bij de grootte van het pakket. Het is echter niet mogelijk om een encoding te voorzien met een k-waarde die lager is dan de helft van de n-waarde. Het gebruik van 8-bit symbolen impliceert dat k niet kleiner kan zijn dan 128, zodat er nog steeds veel padding moet toegevoegd worden aan kleine pakketten. Daarom is dit idee niet uitgewerkt.

De k-waarde kan, zoals werd vermeld, dynamisch gekozen worden, maar de mogelijkheden zijn beperkt omdat de (n,k) parameters tijdens het compileren gekend moeten zijn. In de huidige implementatie zijn de parameters (255,147), (255,247) en alle tussenliggende parameters met een interval van 10 voor de k-waarde beschikbaar.

6.4.3 Pakketformaat

De RS implementatie encodeert volledige pakketten. Bijgevolg zijn deze pakketten niet meer leesbaar zonder de pakketten eerst te decoderen. In tegenstelling tot de vorige technieken, kan van een RS geëncodeerd pakket niet meer gesteld worden dat het een RTP pakket is. Toch lijkt het pakket daar nog sterk op, omdat een RS code een systematische code is. De originele bytes komen integraal voor in het codewoord, waardoor een geëncodeerd pakket eigenlijk het originele pakket is waaraan extra payload wordt toegevoegd.

Er werd getracht deze structuur niet te fel te wijzigen. Toch moet een RS decoder weten welke parameters er gebruikt zijn door de encoder om het pakket te kunnen decoderen. In de implementatie wordt enkel de waarde 255 gebruikt voor n, dus deze parameter moet niet bekend gemaakt worden. De PT byte (inclusief de marker bit) in het pakket wordt vervangen door de waarde van k. Dit kan omdat volgens vergelijking 3.13 geldt dat $k < n$. Daar $n = 255$, zal k steeds kunnen worden voorgesteld met één byte. De PT wordt achteraan de payload toegevoegd, zodat de RS decoder deze kan terugplaatsen na het decoderen. Het verplaatsen van de PT gebeurt voor het encoderen, zodat ook deze informatie geniet van de RS bescherming. Het pakketformaat wordt weergegeven in figuur 6.5.



Figuur 6.5: *Het formaat van een RS geëncodeerd pakket*

6.4.4 Decoderen

Na ontvangst van een met de RS code beschermd pakket, kan de decoder proberen eventuele bitfouten op te sporen. De Schifra library laat toe om niet alleen deze poging te ondernemen, maar laat ook weten hoeveel symbolen foutief werden bevonden, en hoeveel er daarvan zijn gecorrigeerd.

De gebruikte media codecs zijn Internet codecs. Omdat het Internet

traditioneel geen pakketten met bitfouten aflevert, zijn deze codecs ontwikkeld om te functioneren in een omgeving met pakketverlies, maar kunnen ze helemaal niet overweg met foutieve pakketten [57]. Daarom werd beslist om geen foutieve pakketten door te geven aan de media codecs. De RS decoder krijgt de kans om eventuele bitfouten te herstellen. Indien er toch nog fouten overblijven, zal het pakket worden genegeerd. Een RS decoder kan dit besluiten omdat die steeds meer fouten kan detecteren dan corrigeren. Dit is geweten uit sectie 3.5.

Hoofdstuk 7

Integratie in NIProxy

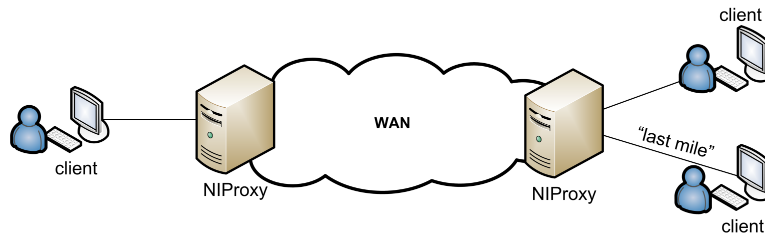
Tot hiertoe werd enkel beschreven hoe bescherming tegen transmissiefouten werd geïmplementeerd in een eigen ontwikkelde applicatie. In dit hoofdstuk wordt verder gekeken en wordt besproken hoe de technieken die voor dergelijke bescherming zorgen, kunnen worden geïntegreerd in een ander systeem dat hier baat bij heeft. Het systeem, de NIProxy genaamd, wordt toegelicht in de volgende secties. Ook wordt geargumenteerd waarom een bescherming tegen transmissiefouten dit systeem ten goede komt.

7.1 Overzicht

De NIProxy (Network Intelligence Proxy) [65] is een netwerkcomponent met hetzelfde doel dat ook centraal staat in dit werk: het verbeteren van de QoE bij gebruikers van multimediale toepassingen. De proxy tracht dit te verwezenlijken door zich in het netwerk te integreren en diensten te verlenen aan een eindapplicatie.

Een Wide Area Network (WAN) zoals het Internet bestaat typisch uit een *backbone* dat is opgebouwd met zeer snelle en betrouwbare links. Voor de verbinding tussen een WAN en een eindgebruiker worden links gebruikt die veel meer beperkingen hebben met betrekking tot bandbreedte en betrouwbaarheid om de kostprijs aanvaardbaar te houden. Naar deze links wordt wel eens verwezen als de *last mile* van een netwerk. Om de QoE voor eindgebruikers te verbeteren, is het belangrijk om zo intelligent mogelijk om te springen met de mogelijkheden van deze links. Daarom werd de NIProxy ontwikkeld. De proxy wordt aan de rand van het WAN geplaatst en tracht de last mile tot de gebruiker zo intelligent mogelijk te beheren. In figuur 7.1 wordt getoond hoe de proxy zich in een netwerk nestelt.

In de context van de NIProxy wordt gesproken van clients. Dit zijn de applicaties voor wie dataverkeer van het WAN bedoeld is. De clients ontvangen dit echter niet rechtstreeks. Ze verbinden met een NIProxy instantie, die voor hen het dataverkeer ontvangt. Vervolgens stuurt de proxy



Figuur 7.1: *Integratie van de NIProxy in een netwerk. De links tussen het WAN en de clients worden de* last mile *genoemd.*

die informatie door waar de client echt behoefte aan heeft.

7.1.1 Bandbreedte beheer

De taak van de NIProxy bestaat erin de bandbreedte naar de verbonden clients te beheren. Dit is een zeer belangrijke taak: zonder dit beheer kan een link verzadigd worden en zullen mediastromen zeer slecht worden ontvangen wat resulteert in een zeer lage QoE. Door de bandbreedte limieten te respecteren, worden mediastromen veel aangenamer ervaren.

De NIProxy voegt intelligentie toe aan het netwerk. Om zijn doel te verwezenlijken, verzamelt de proxy informatie. Dit doet hij op twee gebieden:

Network awareness Ten eerste verzamelt de proxy gegevens over de last mile verbinding. Op elk moment is het belangrijk te weten wat de capaciteit is van de link zodat overbelasting vermeden kan worden.

Application awareness Daarnaast houdt de proxy rekening met de applicatie. Een mediastroom die door de applicatie niet of bijna onopmerkbaar wordt weergegeven aan de eindgebruiker, verspilt enkel bandbreedte. Door gegevens te verzamelen over het belang van mediastromen binnen de applicatie, kan de proxy op elk moment enkel die mediastromen versturen die belangrijk zijn voor de gebruiker, indien er niet voldoende bandbreedte beschikbaar is voor alle stromen.

Aan de hand van de verzamelde informatie tracht de NIProxy een selectie te maken van mediastromen die worden doorgestuurd naar de ontvanger om enerzijds de bandbreedte limiet niet te overschrijden en anderzijds het ontbreken van bepaalde stromen zo weinig mogelijk te doen opvallen voor de gebruiker om een hoge QoE te blijven garanderen.

7.1.2 Services

Naast het selectieproces voor multimediastromen kan de NIProxy ook services uitvoeren voor zijn clients. Zo kan de proxy bijvoorbeeld stromen transcoderen naar een lagere kwaliteit als de client geen behoefte heeft aan

de originele kwaliteit, of kunnen stromen samengevoegd worden waar mogelijk. Services kunnen bijdragen tot de taak van de proxy om de bandbreedte intelligent te beheren. Zo kan de NIProxy, gebruik makend van een mixer, alle audiostromen combineren tot slechts één stroom, zodat minder informatie moet worden verstuurd naar de client. Toch zal de gebruiker hetzelfde resultaat waarnemen als wanneer de verschillende audiostromen pas bij de client zouden worden gemixt.

7.1.3 Toepassingen

De mogelijkheden van de NIProxy werden uitgeprobeerd in enkele toepassingen. [64] beschrijft een NVE (Networked Virtual Environment) met audio- en videocommunicatie. De mediastromen worden gerangschikt volgens belangrijkheid naargelang de positie van de gebruiker in de virtuele wereld. Als een gebruiker naast een videobron staat, is deze veel belangrijker voor de QoE dan een bron die zich op een grote afstand bevindt. Bijgevolg kan de NIProxy, rekening houdend met de beschikbare bandbreedte, beslissen om de veraf gelegen bron met een lagere kwaliteit te versturen, of zelfs helemaal tegen te houden, ten voordele van de bron die dichterbij staat.

In een andere toepassing wordt het weergeven van de omgeving van een NVE beheerd door de NIProxy [63]. In een virtuele wereld moeten zeer veel modellen worden ingeladen om de wereld voor te stellen op het beeldscherm. Deze informatie kan niet allemaal tegelijk tot bij de gebruiker worden gebracht vanwege de bandbreedte beperkingen. In een NVE is het zeer storend wanneer er plots een object verschijnt op het scherm dat er eigenlijk al lang staat, maar nog niet kon worden weergegeven omdat de informatie niet beschikbaar was voor de client. Door rekening te houden met de plaats van de gebruiker in de virtuele wereld, kan de NIProxy beslissen welke modellen met de hoogste prioriteit moeten worden verzonden, zodat de gebruiker snel een correct beeld van de wereld te zien krijgt. Andere objecten die de gebruiker niet ziet, omdat ze zich bijvoorbeeld achter hem bevinden, kunnen later worden verstuurd.

7.2 FEC service

Forward Error Correction is een nieuwe vorm van netwerkintelligentie die door de NIProxy kan worden uitgebuit. Naast de bescherming tegen overbelasting van de netwerklink, is de NIProxy door zijn positie tussen het betrouwbare WAN en de onbetrouwbare last mile uiterst geschikt om multimediastromen te beschermen tegen transmissiefouten. Zo kan de proxy een nog betere QoE garanderen. Het integreren van de technieken die werden ontwikkeld in hoofdstuk 6 is dus een logische keuze.

Eerder werd gesproken over NIProxy services. Het toevoegen van FEC aan mediastromen kan gezien worden als een service die de NIProxy kan

uitvoeren om de QoE bij de ontvangers te verbeteren. De beschermingstechnieken werden dan ook als NIProxy service geïmplementeerd.

7.2.1 Testapplicatie

Eerder werd beschreven dat mediatransport in de voor dit werk ontwikkelde applicatie uit sectie 5 via multicast verloopt. Twee instanties van de applicatie kunnen met elkaar communiceren door naar dezelfde multicast adressen te luisteren. Bij het gebruik van de NIProxy verandert er weinig aan dit idee; elke client verbindt in dit geval met de NIProxy, die op zijn beurt voor hen naar de multicast adressen luistert.

Omdat de eerder beschreven applicatie beschermde mediastromen kan decoderen, is ze geschikt om gebruikt te worden als client voor de NIProxy. Hiervoor moest de applicatie worden aangepast, zodat ze kan verbinden met de proxy. Verbinden via de NIProxy is echter één van de abstracties die worden aangeboden door de NAB library. Daar de applicatie deze library gebruikt als netwerklaag, was deze aanpassing zeer eenvoudig.

7.2.2 Implementatie

FEC werd in de NIProxy geïntegreerd als service. De architectuur van een NIProxy service is zeer eenvoudig. Een service registreert zich voor het ontvangen van bepaalde datastromen en vervolgens wordt elk pakket uit deze stroom ter verwerking aangeboden aan de service. Deze kan ervoor kiezen dit pakket aan te passen, te verwijderen of ongemoeid te laten. Verder kan een service extra pakketten genereren en toevoegen aan de datastroom.

Er werden twee services voor de NIProxy geïmplementeerd, met name de `ProxyAudioFecService` en de `ProxyVideoFecService`. De expliciete opsplitsing tussen audio en video is noodzakelijk, omdat beide mediatypes op verschillende manieren worden beschermd. Redundant Audio Coding kan bijvoorbeeld niet worden toegepast op videobeelden.

Het verwerken van pakketten in één van de services verloopt analoog aan het proces in de bypass van de eerder beschreven applicatie bij het verzenden en beschermen van de mediastromen (zie ook figuur 6.2(a)). De implementatie vertoont dan ook sterke gelijkenissen: na het ontvangen wordt een pakket verwerkt door een ketting van encoders. Het resultaat wordt vervolgens doorgestuurd naar de client.

Een belangrijk verschil met de applicatie zit in het aantal te verwerken stromen. Op een multicast netwerk is het best mogelijk dat er meer dan één mediabron is. De NIProxy moet dus mogelijk meerdere mediastromen tegelijk voor de client beschermen. Hiermee werd rekening gehouden: voor elke mediabron waarvan pakketten worden ontvangen, zal een nieuwe ketting van encoders worden geïnstantieerd. Sommige encoders verzamelen gegevens uit meerdere pakketten om bescherming te voorzien. Te denken valt aan

de parity code, die meerdere pakketten groepeert tot een extra redundant pakket. Pakketten van verschillende bronnen mogen niet vermengd worden om een goede werking te garanderen. Bijgevolg zijn meerdere instanties van de encoders noodzakelijk!

De NIProxy FEC services ondersteunen net als de applicatie verschillende instellingen. In de huidige implementatie worden deze ingelezen uit een configuratiebestand.

7.2.3 mmfec library

De code van de FEC encoders wordt in dit werk in twee verschillende toepassingen benut, met name in de applicatie uit hoofdstuk 5 en in de NIProxy FEC services. Om onderhoudsvriendelijk te blijven, werd deze code gebundeld in een statische library, die de mmfec (Multimedia FEC) library werd gedoopt. Zo kunnen beide toepassingen zonder veel moeite genieten van eventuele verbeteringen of uitbreidingen. De library bevat ook de code van de FEC decoders, zodat alle FEC functionaliteit eenvoudig kan worden hergebruikt in andere toepassingen. In tegenstelling tot de applicaties zelf, is deze library niet gebonden aan één platform. De library werd getest in een Windows en een Linux omgeving.

Deel III

Evaluatie

Hoofdstuk 8

Simulatie

8.1 Emulators en simulators

De testomgeving waarin geëxperimenteerd werd met de applicatie en de NI-Proxy werd opgezet op een LAN. De netwerkomstandigheden op een LAN kunnen echter niet worden vergeleken met de omstandigheden in een echte omgeving: op een LAN met slechts enkele apparaten kunnen onmogelijk dezelfde complexe situaties voorkomen als op een wereldwijd netwerk. Om toch realistische resultaten te kunnen bekomen, moeten deze situaties worden gesimuleerd.

De ontwikkelde applicatie werd voorzien van een functie die pakketverlies kan simuleren. Deze simulatie is echter zeer eenvoudig: er wordt volledig willekeurig beslist of een pakket al dan niet effectief wordt uitgezonden. In een realistische situatie zijn pakketverliezen meer gecorreleerd [32]. Daarom werden enkele tools bekeken die een meer realistisch resultaat genereren.

Er zijn twee groepen software die netwerken simuleren. Aan de ene kant spreekt men van emulators. Dit zijn netwerkapparaten die zich in een fysiek netwerk nestelen en het dataverkeer op een link effectief beïnvloeden. NIST Net [9] is een voorbeeld van een veelgebruikte emulator. Anderzijds zijn er simulators. Dit zijn tools die netwerken in software simuleren. Data wordt fictief gegenereerd of ingelezen uit bestanden. Simulators vergaren in het algemeen zeer veel informatie tijdens een simulatieproces, dat doorgaans niet in real-time maar versneld plaatsvindt, en bieden de gebruiker achteraf een waaier van tools om deze data te analyseren. Onder de bekende simulators vinden we ns-2 [14], OPNET[1] en NCTUns [59].

8.2 NCTUns

Er werd voor twee tools bestudeerd in hoeverre ze konden worden ingezet bij de evaluatie van dit werk. Omdat getracht werd in real-time te wer-

[1] `http://www.opnet.com/`

ken met de applicaties, moest de functionaliteit van een emulator voorzien worden. Toch werd er niet gekozen voor NIST Net. Met deze emulator kunnen pakketverlies en enkele andere fenomenen die typisch zijn voor het Internet worden gesimuleerd, maar beschadiging van bits behoort niet tot de mogelijkheden. Daarom viel de keuze op NCTUns.

8.2.1 Emulatie

NCTUns is een simulator, maar kan ook fysieke apparaten opnemen in zijn simulatiemodel. Simulatie wordt daarbij tegen real-time snelheid uitgevoerd en data kan worden uitgewisseld met fysieke hosts door middel van de netwerkkaart van de simulatie server. Zo voorziet de simulator ook in emulatie functionaliteit.

Fysieke hosts worden in NCTUns *externe* hosts genoemd. Ze worden evenwaardig als gesimuleerde hosts opgenomen in het simulatiemodel. Binnen dit model krijgt elke host een fictief IP adres in het simulatie subnet 1.0.0.0/16. Voor externe hosts wordt dit adres via NAT (Network Address Translation) gekoppeld aan het echte IP adres van de host op het netwerk, zodat communicatie met het externe toestel mogelijk is. De NAT werkt bidirectioneel. Daarom kan ook een externe host een verbinding maken met elke andere host binnen het simulatiemodel. Het enige wat hiervoor moet worden voorzien is een extra regel in de routing tabel van de externe host, zodat alle verkeer voor het simulatie subnet naar de simulatie server wordt gerouteerd. Doordat het voor een externe host mogelijk is om te communiceren met een andere externe host in het simulatiemodel, kan NCTUns volwaardig worden gebruikt als emulator.

8.2.2 Voor- en nadelen

Het gebruik van de NCTUns simulator maakt het mogelijk om fysieke hosts te laten deelnemen in complexe en flexibele netwerkstructuren zonder dat deze fysiek moeten worden gebouwd. Een simulator verzamelt alle gegevens waarop vervolgens zeer uitgebreide analyse kan plaatsvinden. Verder beschikt NCTUns onder andere over een functie waarmee de simulatie opnieuw kan worden afgespeeld tegen een instelbare snelheid, zodat alles van zeer nabij kan worden geïnspecteerd. Bij het gebruik van een gewone emulator wordt *slechts* het gedrag van een netwerklink gecontroleerd. Alle vormen van verzamelen en analyseren van gegevens moeten gebeuren via andere tools.

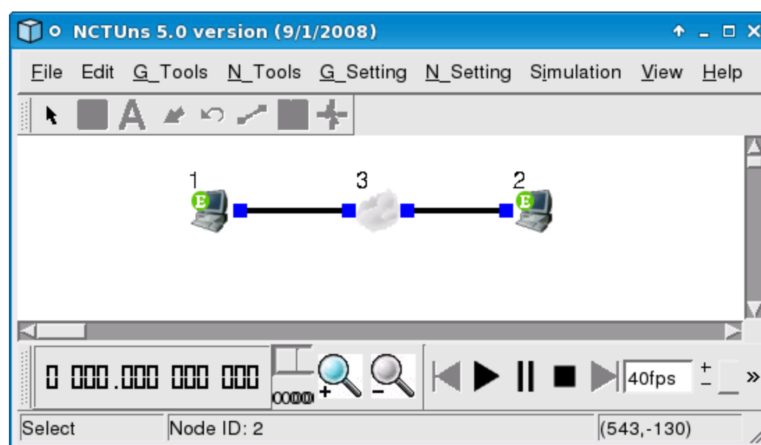
Er zijn echter ook nadelen verbonden aan een simulator. Tijdens de simulatie met NCTUns is het niet mogelijk om instellingen te wijzigen. Als de omstandigheden van het gesimuleerde netwerk moeten wijzigen, dan moet ofwel bij het opbouwen van het simulatiemodel worden aangegeven op welk tijdstip dit moet gebeuren, ofwel moet de simulatie onderbroken worden.

Dit maakt het gebruik van NCTUns omslachtiger dan het gebruik van een emulator als NIST Net, waar at-runtime parameters kunnen worden gewijzigd via een handige GUI.

8.3 Opstelling

8.3.1 Simulatiemodel

Het simulatiemodel voor NCTUns moet twee taken kunnen vervullen: pakketten moeten aan een bepaald ratio verloren gaan en de link moet aan een bepaald ratio bitfouten introduceren. Een eenvoudig model dat twee externe hosts verbindt, volstaat. De grafische modeleditor van NCTUns waarin dit model werd opgesteld, wordt weergegeven in figuur 8.1. Tussen de hosts zit een WAN node die pakketverlies kan simuleren. De betrouwbaarheid van de links kan worden ingesteld voor de simulatie van bitfouten.

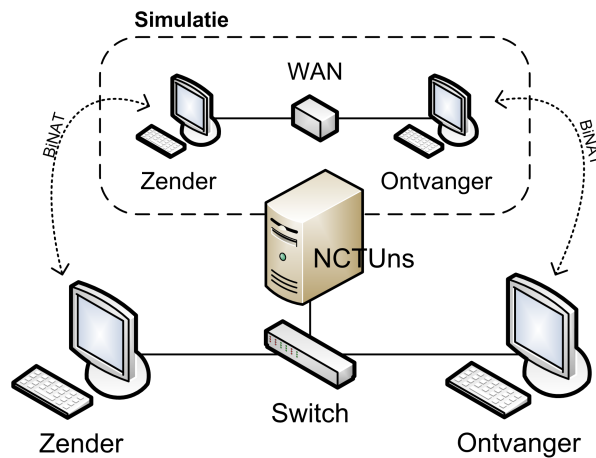


Figuur 8.1: *Het simulatiemodel in een screenshot van NCTUns. Nodes 1 en 2 zijn externe hosts. Node 3 simuleert een WAN tussen beide hosts.*

8.3.2 Rechtstreekse verbinding

Een rechtstreekse verbinding tussen twee instanties (in de figuren *clients* genoemd) van de ontwikkelde applicatie is eenvoudig te verwezenlijken: twee hosts, een zender en een ontvanger worden aan hetzelfde LAN gekoppeld. Met dit LAN is ook de NCTUns server verbonden. De opstelling wordt getoond in figuur 8.2.

Deze opstelling heeft één groot probleem: multicast verkeer kan niet door de NCTUns server gerouteerd worden. Dit wordt opgelost door op de zendende host een Multicast Tunnel Server (zie ook sectie 5.3.1) te installeren. De ontvanger kan vervolgens via unicast met dit hulpmiddel communiceren. Dit unicast verkeer kan via de simulatie server gerouteerd worden.

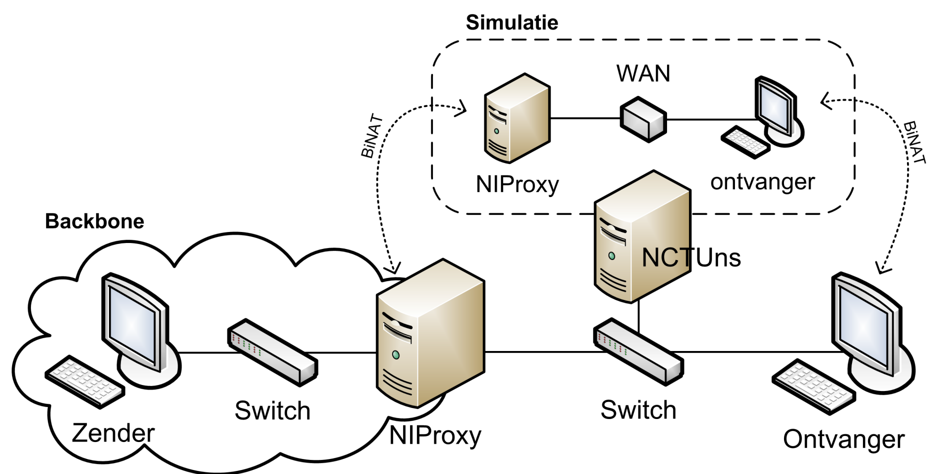


Figuur 8.2: *Hardware opstelling met rechtstreekse verbinding tussen zender en ontvanger, uitgebreid met een NCTUns server voor de simulatie van transmissiefouten.*

Het is heel eenvoudig om te wisselen tussen een situatie waarbij de clients rechtstreeks met elkaar communiceren of waarbij de NCTUns server wordt gebruikt. Beide clients zijn verbonden met hetzelfde LAN, dus de ontvanger kan met behulp van het IP adres van de zender rechtstreeks verbinden met de Multicast Tunnel Server. Om de NCTUns server te gebruiken, moet de ontvanger enkel verbinding maken met het fictieve IP adres dat de zender toegewezen krijgt in het simulatiemodel. Alle communicatie verloopt vervolgens via de NCTUns server. NAT zorgt ervoor dat alle verkeer de juiste fysieke host bereikt.

8.3.3 Opstelling met NIProxy

Een opstelling waarbij de NIProxy wordt gebruikt is complexer en vraagt een uitbreiding van de opstelling uit figuur 8.2. De zender wordt vervangen door de NIProxy. Deze zal data ontvangen van de WAN backbone, zoals beschreven in sectie 7.1. De backbone wordt gesimuleerd met een tweede LAN waarin de zender wordt geplaatst. De verbinding tussen de NIProxy en de ontvanger stelt de last mile voor. Transmissiefouten worden gesimuleerd door de NCTUns server. De opstelling wordt weergegeven in figuur 8.3.



Figuur 8.3: *Hardware opstelling met de NIProxy opgenomen in het netwerk.*

Hoofdstuk 9

Resultaten

9.1 IEEE 802.3 en 802.11 beperkingen

In dit werk werd veel gesproken over bitfouten die geïntroduceerd worden door de imperfectie van communicatielinks. Uiteraard werden er door de ontwikkelaars van protocollen voorzieningen getroffen om deze bitfouten op te sporen. Anders zou het onmogelijk zijn om betrouwbare communicatie te implementeren.

De beschermingstechnieken die werden geïmplementeerd in dit werk opereren in de applicatielaag van het OSI netwerkmodel [54]. Het is dus belangrijk dat pakketten met bitfouten de applicatielaag ook effectief bereiken.

Voor het transport wordt gebruik gemaakt van RTP over UDP/IP. In sectie 4.5 werd besproken dat de UDP controlesom moet worden uitgeschakeld, zodat dit transportprotocol een pakket met bitfouten niet verwijdert. De netwerklibraries die werden gebruikt, werden aangepast om dit mogelijk te maken. Wanneer de bitbescherming wordt geactiveerd in de applicatie, zal de UDP controlesom worden uitgeschakeld.

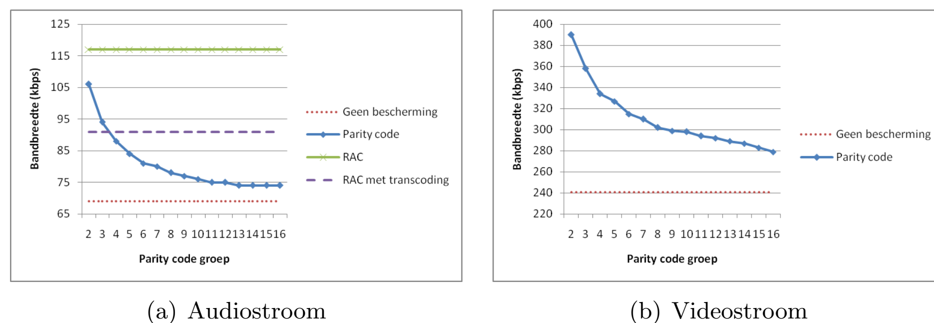
Bij het uitvoeren van tests bleek het uitschakelen van de UDP controlesom echter niet te volstaan. De tests verliepen over een LAN dat als datalinkprotocol kon kiezen uit de IEEE 802.3 (Ethernet) of de IEEE 802.11 (WLAN) standaard. Als fouten ontstaan in een netwerkframe, zullen deze protocollen het frame negeren. Dit gedrag kan niet gewijzigd worden. Dit probleem wordt ook aangekaart in [10]. De auteurs stellen een aanpassing van IEEE 802.11 voor waarbij de payload van de netwerkframes niet meer wordt opgenomen in de berekening van de CRC die de foutdetectie voorziet, zodat het protocol enkel zijn eigen headers beschermt.

De NCTUns software kan het effect van bitfouten simuleren. Toch worden de links gesimuleerd conform aan de IEEE 802.x standaarden. Bijgevolg bereiken beschadigde netwerkframes nooit hun doelhost, waardoor simulatie via een NCTUns server in hetzelfde bedje ziek is als het gebruik van een echt WLAN.

Men kan besluiten dat bescherming tegen bitfouten niet toepasbaar is op conventionele netwerken als Ethernet en WLAN. Daarom werd de RS bescherming niet opgenomen in de experimenten betreffende bandbreedte en efficiëntie.

9.2 Bandbreedte vereisten

Een eerste interessante vergelijking die kan worden gemaakt, betreft de hoeveelheid bandbreedte die nodig is om redundante FEC data te versturen. Figuur 9.1 zet voor de parity code de hoeveelheid benodigde bandbreedte uit tegenover het aantal mediapakketten dat wordt gecombineerd per redundant pakket. Het experiment werd apart uitgevoerd voor audio- en videostromen. Elke grafiek bevat de hoeveelheid bandbreedte die een niet beschermde stroom verbruikt als referentie. Op de grafiek betreffende audio werden tevens de vereisten voor Redundant Audio Coding aangebracht.



(a) Audiostroom (b) Videostroom

Figuur 9.1: *Bandbreedte gebruik voor verschillende mediatypes, technieken en instellingen.*

9.2.1 Vergelijking van technieken

Uit deze grafieken is zeer duidelijk te zien dat de parity code met een kleine groep gecombineerde pakketten veel bandbreedte vereist. Deze hoeveelheid neemt exponentieel af naarmate de grootte van de groep toeneemt.

Niet verwonderlijk is het feit dat Redundant Audio Coding zeer veel bandbreedte vereist. Men mag echter niet over het hoofd zien dat deze techniek meer verloren pakketten kan vervangen dan de parity code. Dit zal ook blijken uit de volgende sectie. Opmerkelijk is wel dat door het gebruik van piggybacking toch zichtbaar minder bytes nodig zijn ten opzichte van het twee keer verzenden van een pakket (117 kbps t.o.v. twee maal 69 kbps indien een kopie zou worden gemaakt).

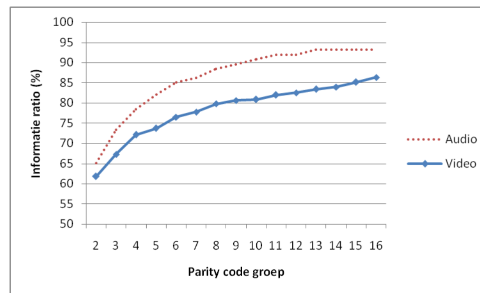
Indien de techniek de audiopakketten converteert naar een lagere kwaliteit (op de grafiek aangegeven als "RAC met transcoding"), situeert het

bandbreedte verbruik zich ter hoogte van een parity code die een klein aantal pakketten groepeert.

9.2.2 Vergelijking audio en video

Het zou oneerlijk zijn om bits te tellen bij het vergelijken van audio en video, omdat een videostroom zelfs zonder bescherming veel meer bandbreedte verbruikt. Toch vertoont het variëren van de grootte van de groep voor de parity code dezelfde exponentiële afbouw.

Wat wel werd vergeleken is het informatie ratio van de parity code in beide stromen. Figuur 9.2 geeft voor beide mediatypes weer hoeveel percent van de totale bandbreedte benut wordt voor mediadata. Het resterende gedeelte wordt opgeslorpt door redundante data.



Figuur 9.2: *Informatie ratio van de parity code voor audio en video.*

Opvallend is dat het informatie ratio voor audio altijd hoger ligt dan voor video. Hier is een verklaring voor: videopakketten hebben een zeer variërende pakketgrootte, wat bij een audiostroom minder het geval is. Daardoor is er meer padding nodig voor de bescherming van videopakketten. Padding is verspilling van bandbreedte. Deze grafiek bewijst dat het ontwikkelen van UEP technieken vooral videostromen ten goede zal komen.

9.3 Foutcorrectie capaciteit

In deze sectie wordt er gekeken hoe efficiënt de verschillende technieken zijn. Hiervoor werden twee parameters geëvalueerd. Enerzijds werd gekeken hoeveel pakketten na de poging tot herstel toch nog ontbraken (de residu packet loss rate (PLR)). Anderzijds werd ook berekend welk percentage van de ontvangen redundante data ook effectief bijdraagt tot het herstellingsproces. Uiteraard kan elke vorm van pakketverlies worden opgevangen door een pakket ettelijke keren te versturen, maar als slechts een klein percentage van de kopieën ook daadwerkelijk gebruikt wordt, is het extra bandbreedte verbruik dan nog wel te verantwoorden? Om op deze vraag te kunnen antwoorden, is het nodig om data over het gebruikte percentage redundante

informatie te verzamelen. Parity code en Redundant Audio Coding werden afzonderlijk geëvalueerd.

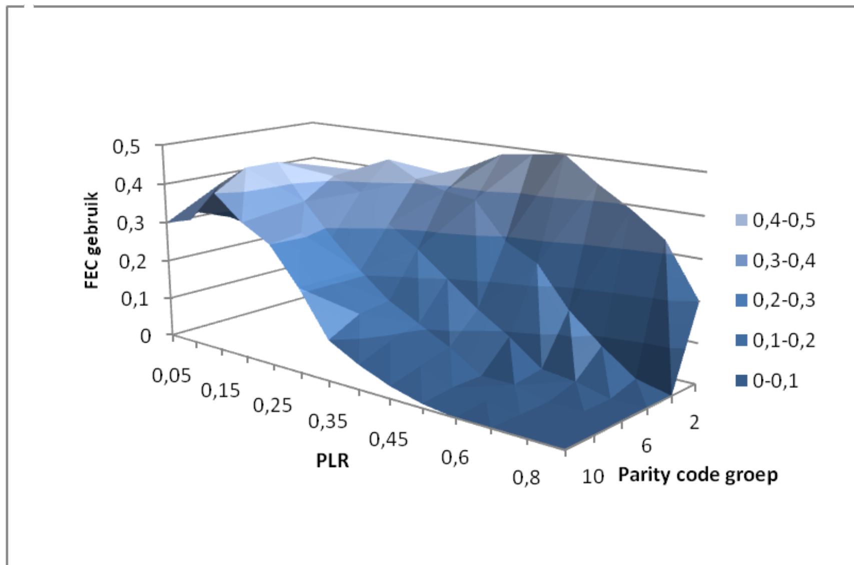
9.3.1 Parity code

Net als in de vorige sectie, werd de parity code in al zijn parameters geëvalueerd. Elke grootte voor de groep gecombineerde pakketten werd onder verschillende netwerkomstandigheden bekeken. Dit resulteert in driedimensionale grafieken. Figuur 9.3(a) toont het percentage FEC pakketten dat effectief werd benut door de ontvanger. Dit percentage is gemeten vanuit het standpunt van de ontvanger en wordt dus berekend op het aantal ontvangen pakketten. Figuur 9.3(b) geeft het residu PLR aan onder de verschillende omstandigheden. Dit is dus de hoeveelheid pakketten die verloren kunnen worden beschouwd *na* reconstructie.

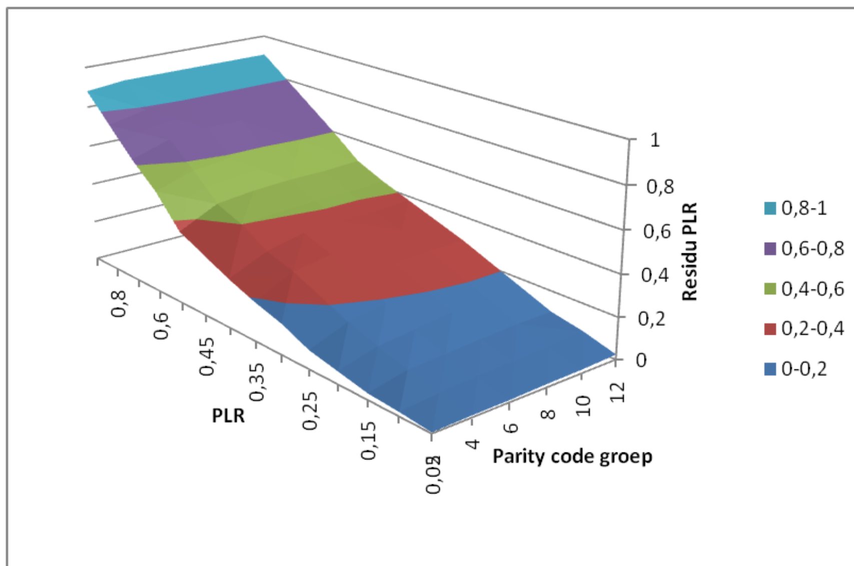
Om deze grafieken begrijpbaarder te maken, toont figuur 9.4 enkele doorsneden. Figuur 9.4(a) beeldt een doorsnede af die loodrecht staat op de diepteas. Dit betekent dat de groep grootte een vaste waarde heeft. In deze grafiek is deze waarde 2. De grafiek toont doorsneden van beide driedimensionale grafieken uit figuur 9.3. Zo kunnen de verbanden tussen FEC gebruik en residu PLR op één grafiek bestudeerd worden. Tevens wordt het PLR zonder toepassing van bescherming als referentie op de grafiek weergegeven, zodat het verschil met het residu PLR duidelijk wordt. De grafiek in figuur 9.4(b) is op dezelfde manier opgebouwd, maar ditmaal wordt de PLR als constante waarde genomen. De doorsnede bevindt zich op een pakketverlies van 20%.

Wat onmiddellijk opvalt in figuur 9.4(a) is dat het percentage pakketten dat gebruikt wordt bij de reconstructie in een boog varieert. Herinner u dat een parity pakket enkel wordt gebruikt als er exact één pakket ontbreekt van de gegroepeerde mediapakketten. De daling na het hoogtepunt wijst erop dat er meer dan één pakket per groep verloren gaat. Bijgevolg kan er minder redundante data benut worden. Aan de andere kant van het hoogtepunt is de daling te wijten aan een te veel aan redundante informatie. Als geen mediapakketten verloren gaan, wordt het parity pakket niet gebruikt. Op het hoogtepunt van de grafiek wordt het meest redundante data gebruikt. Dit is het punt waarop de extra bandbreedte voor FEC het best te verantwoorden valt. In het geval van een groep grootte 2 is dit het punt waarbij 50% PLR wordt geïntroduceerd door het netwerk. Opmerkelijk is ook het feit dat nooit meer dan 50% van de redundante data zal worden gebruikt. Dit blijkt uit figuur 9.3(a).

Een grafiek als in figuur 9.4(b) is de beste hulp bij de selectie van welke parameter het best past bij een bepaalde hoeveelheid pakketverlies. Daaruit valt vooral af te lezen dat niet de groep grootte met het meeste effectief gebruik van redundante data het beste scoort in het wegwerken van transmissiefouten. Uiteraard doen alle kleinere groepen het beter, maar het

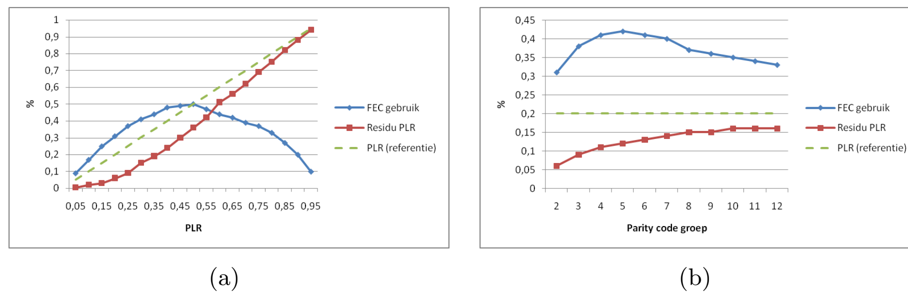


(a) Gebruik van redundante data



(b) Resterend pakketverlies

Figuur 9.3: *Efficiëntie evaluatie van de parity code. Opmerking: om de data overzichtelijk te kunnen presenteren, zijn de assen in beide grafieken anders opgesteld.*



(a) (b)

Figuur 9.4: *Details van de efficiëntie evaluatie van de parity code. (a) evaluatie met een groep grootte van 2; (b) evaluatie met een pakketverlies van 20%*

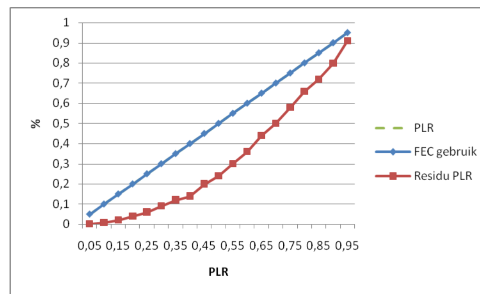
bandbreedte verbruik stijgt. Bij de selectie van een geschikte grootte zal steeds een afweging moeten gemaakt worden tussen hoeveel bandbreedte er mag gebruikt worden en welk resultaat hiermee bereikt moet worden.

Een algemene vaststelling is ook dat de grafiek in figuur 9.3(a) steeds lager en vlakker wordt naarmate de groep grootte toeneemt. Dit betekent dat er steeds minder redundante data effectief gebruikt wordt naarmate de groep grootte stijgt. Het experiment beschouwt enkel groepen tot grootte 12, terwijl de implementatie groottes tot 16 aanvaardt. Het gebruik van nog grotere groepen is in ieder geval niet interessant.

9.3.2 Redundant Audio Coding

De evaluatie van de Redundant Audio Coding is iets eenvoudiger dan die van de parity code. Een overzicht wordt gegeven in figuur 9.5. Opmerkelijk is de lijn die het FEC gebruik aanduidt. Deze maakt geen boogvorm zoals bij de parity code. Er is geen kantelpunt waarop er te weinig data wordt ontvangen waardoor de redundante data niet langer gebruikt kan worden, omdat elke kopie volledig onafhankelijk is van eender welk ander pakket. Ook valt het op dat deze lijn exact de PLR referentielijn bedekt. Een eenvoudige redenering verklaart dit: als $x\%$ van alle pakketten verloren gaat, is dit ook $x\%$ van de mediapakketten. Dit percentage kan dus mogelijk hersteld worden, waardoor ook $x\%$ van de ontvangen kopieën kan ingezet worden!

Wat efficiëntie ten opzichte van pakketverlies betreft, kan deze grafiek vergeleken worden met de parity code die het meeste verloren pakketten kan vervangen. Dit is de code met groep grootte 2, want bij deze wordt het meeste redundante data toegevoegd waarmee correcties kunnen worden uitgevoerd. Het resulterende PLR van deze code wordt afgebeeld in figuur 9.4(a). Redundant Audio Coding kan elk pakket vervangen, terwijl deze parity code slechts 1 pakket per groep van 2 kan herstellen. Daarom bereikt de lijn die de residu PLR aangeeft in de grafiek van Redundant Audio Coding lagere waardes. Dit is een interessant resultaat als ook het



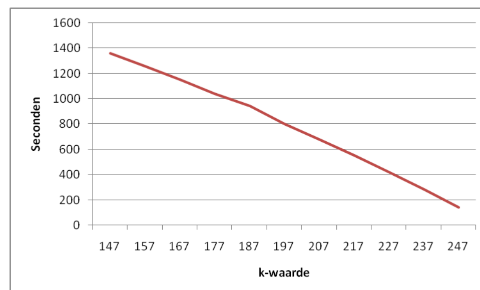
Figuur 9.5: *Efficiëntie evaluatie van Redundant Audio Coding*

bandbreedte verbruik erbij wordt gehaald (figuur 9.1(a)). Daar blijkt dat het verbruik voor Redundant Audio Coding met extra compressie lager is dan het verbruik van de parity code met groep groottes 2 en 3. Redundant Audio Coding weet dus meer fouten te corrigeren dan de parity code en heeft hier minder bandbreedte voor nodig.

9.4 Systeemvereisten

Bandbreedte en efficiëntie zijn belangrijke parameters bij de evaluatie van foutcorrectie technieken. Er moet echter ook aandacht besteed worden aan de systeemvereisten. Daar met real-time applicaties wordt gewerkt, is een techniek slechts bruikbaar als de hardware van een systeem ook in staat is om de bescherming in real-time toe te passen.

De berekeningen bij de Reed-Solomon techniek zijn de meest complexe in dit werk. Daarom werd gemeten hoeveel tijd de encoder nodig heeft om 1000 maal een codewoord te berekenen in het geval van een (255,k) RS code. k werd variabel gekozen. Het experiment werd uitgevoerd op een Intel® Pentium M 730 processor (1.60 Ghz). Verschillende waardes voor k kunnen heel variërende uitvoertijden opleveren: de (255,247) code had slechts 140 ms rekentijd nodig, terwijl de (255,147) code hier tot 1500 ms over doet.



Figuur 9.6: *CPU gebruik van de RS code*

Uit figuur 9.6 blijkt dat de benodigde processortijd lineair toeneemt naarmate de RS code meer redundante symbolen moet genereren. Processortijd is een belangrijk aspect bij de implementatie van de NIProxy. Deze netwerkcomponent moet zoveel mogelijk clients kunnen dienen. Een snelle verwerking van de individuele mediastromen is hierbij gewenst.

Bij de parity code en Redundant Audio Coding zijn er geen noemenswaardige problemen betreffende systeemvereisten. Ter vergelijking: het uitvoeren van 1000 XOR operaties met pakketten van 255 bytes duurt slechts drie milliseconden met de eerder aangehaalde hardware.

Hoofdstuk 10

Uitbreidingen

In hoofdstuk 6 werden enkele implementaties besproken van beschermingen tegen transmissiefouten. Uiteraard is er steeds ruimte voor verbetering van de ontwikkelde software. Dit hoofdstuk beschrijft enkele uitbreidingen die een meerwaarde kunnen bieden aan de bestaande implementatie.

10.1 Geavanceerde media codecs

Eerder werd aangehaald dat de H.263+ video codec en de Speex audio codec niet de meest geschikte data genereren om FEC technieken op toe te passen. Een uitbreiding van EMIPLIB met andere codecs zou meer mogelijkheden scheppen.

10.1.1 Unequal Error Protection

In het implementatiegedeelte werd nergens gebruik gemaakt van Unequal Error Protection (UEP). Noch de Speex encoder, noch de H.263+ encoder duiden duidelijk aan welke blokken bits belangrijker zijn dan anderen. Zonder deze informatie is UEP niet mogelijk.

In [55] wordt beschreven hoe een H.263(+) bitstroom kan worden gereorganiseerd om tot vier verschillende niveaus te vormen die afzonderlijk kunnen worden beschermd. Om dit effectief te implementeren, is een grondige kennis van de H.263 bitstroom nodig. Deze is echter zeer complex [23].

In de literatuurstudie werd een voorbeeld aangehaald van de AMR codec (sectie 4.6.2). Deze genereert drie groepen bits met elk een eigen niveau van belangrijkheid. Deze codec is echter niet vrij beschikbaar. Een beschikbare audio codec die zich wel uitstekend leent tot UEP is de iLBC (Internet Low Bitrate Codec) [2]. De documentatie beschrijft exact hoe de bytes van een iLBC pakket kunnen worden verdeeld volgens belangrijkheid: in een pakket met 20ms spraakinformatie zijn de eerste zes bytes het meest gevoelig voor bitfouten. De volgende acht bytes zijn minder gevoelig en de laatste 24 bytes

zijn het minst belangrijk.

10.1.2 Verbeterde Redundant Audio Coding

In de sectie over de implementatie van Redundant Audio Coding werd uiteengezet waarom de techniek niet erg geschikt is om te gebruiken in combinatie met de Speex codec. Enkele oplossingen werden voorgesteld, maar om de problemen echt op te lossen, moet een andere codec gebruikt worden waarvan de decoder niet afhankelijk is van een inter-pakket status. Deze codecs zijn echter schaars, want juist door het gebruik van inter-pakket predictie algoritmes zijn codecs in staat om aan een lage bitrate toch een goede afspeelkwaliteit aan te bieden.

iLBC is een codec die een pakket onafhankelijk van de andere pakketten kan decoderen [2]. Een decoder status wordt enkel bijgehouden om opvulling te voorzien wanneer een pakket ontbreekt en niet wordt vervangen. Toch is ook iLBC niet meer geschikt dan Speex, omdat de encoder geen kwaliteitsparameter heeft. Elk pakket heeft een vaste grootte van 38 bytes en kan niet verder gecomprimeerd worden. Zonder extra compressie is iLBC niet efficiënter dan Speex waarbij gewone kopieën worden gemaakt.

10.2 Adaptiviteit

In de literatuurstudie werd gesproken over vormen van bescherming die zich aanpassen aan de netwerkomstandigheden. Dit is een heel interessante uitbreiding op de huidige implementatie omdat de omstandigheden op een netwerk wel eens durven wijzigen. Bijvoorbeeld, als een extra client wordt toegevoegd in een multimediasessie, wordt meer data over het netwerk verwacht. Het zou kunnen dat een netwerkcomponent in deze situatie overbelast geraakt en niet langer alle pakketten kan verwerken. Een adaptief algoritme zal zich aanpassen aan deze situatie.

10.2.1 Meten van de netwerkomstandigheden

Om adaptiviteit te voorzien, moet een applicatie de netwerkomstandigheden kunnen waarnemen. Twee waarnemingen zijn belangrijk: hoeveel pakketten gaan er verloren en welk percentage bitfouten komt voor in de pakketten die aankomen? De ontvanger van een RTP mediastroom beschikt over voldoende informatie om deze waardes te berekenen. Enerzijds zijn RTP pakketten genummerd met een sequentienummer. Het ontbreken van een pakket is dus zeer eenvoudig op te merken. Anderzijds kan een RS decoder berekenen hoeveel symbolen in het RS codewoord foutief zijn ontvangen. De ontvanger heeft bijgevolg voldoende kennis over de netwerkomstandigheden om de zender bij te sturen.

10.2.2 Feedback kanaal

Redundante data wordt gegenereerd bij de zender. Het is echter de ontvanger die de netwerkomstandigheden berekent. De ontvanger moet zijn waarnemingen aan de zender kunnen signaleren. Zoals al werd aangehaald in sectie 4.9, voorziet het gebruik van RTP als transportmechanisme al in een handig communicatiekanaal waarmee zender en ontvanger informatie kunnen uitwisselen die niet tot de mediastroom behoort, met name het RTCP protocol. Dit protocol beschrijft RTCP APP pakketten, waarmee een applicatie zelf gedefinieerde data kan uitwisselen. Daar JRTPLIB de RTCP APP pakketten ondersteunt, kan het signaleren van de netwerkomstandigheden aan de zender vlot worden geïmplementeerd.

10.2.3 Implementatie van adaptieve algoritmes

Een zender zal met behulp van de informatie over de netwerkomstandigheden de parameters van zijn FEC encoders moeten aanpassen. Deze encoders kunnen nu al met verschillende parameters overweg, dus het aanpassen op zich vormt geen probleem. Om de juiste keuze te maken welke parameter bij welke netwerksituatie hoort, bieden analyses als deze in sectie 9 nuttige informatie. De parameters van de parity code werden al uitvoerig vergeleken onder verschillende netwerkomstandigheden. Een gelijkaardige analyse kan uitgevoerd worden voor de parameters van de RS code.

10.2.4 Adaptiviteit voor de NIProxy

Adaptieve algoritmes komen pas echt tot hun recht wanneer ze worden geïntegreerd in de NIProxy. In de huidige implementatie wordt elke mediastroom voor elke client op een gelijkaardige manier beschermd. Toch kunnen de verbindingen van de verschillende clients onderhevig zijn aan heel uiteenlopende netwerkomstandigheden. Met adaptieve algoritmes kan de NIProxy at-runtime individueel voor elke client de beste bescherming selecteren en indien nodig de bescherming aanpassen aan wijzigende omstandigheden. Deze individuele aanpak wordt ook gebruikt bij de functionaliteit die de bandbreedte van de clients beheert [65].

De NIProxy zou zelfs zo kunnen worden uitgebreid dat de selectie van FEC technieken samenhangt met bandbreedte beheer. Hiermee wordt bedoeld dat de NIProxy kan beslissen om een mediastroom die slecht wordt ontvangen te converteren naar een lagere kwaliteit, zodat extra bandbreedte vrijkomt om meer redundante data te kunnen versturen als bescherming tegen de transmissiefouten. Een gelijkaardig idee werd in de literatuurstudie in sectie 4.9.2 aangehaald.

Hoofdstuk 11

Conclusie

In dit werk werd aangetoond dat communicatie over een netwerk kwalitatief kan worden verbeterd met behulp van Forward Error Correction (FEC) technieken. Door toevoeging van redundante informatie kunnen transmissiefouten worden opgevangen aan de ontvangstkant. De ontvanger zal met de extra gegevens in staat zijn om fouten op te sporen en te corrigeren indien nodig. Door met deze methodes de communicatiekanalen van real-time multimedia toepassingen te beschermen, ondervindt de gebruiker minder last van slechte netwerkomstandigheden.

Traditioneel wordt hertransmissie gebruikt om beschadigingen door het netwerk te compenseren. Omdat met FEC technieken beter rekening gehouden kan worden met de tijdsbeperkingen die eigen zijn aan real-time mediastromen, vormen deze een meer bruikbaar alternatief.

Shannon bewees dat er voor elk communicatiekanaal een grens bestaat die bepaalt hoeveel data betrouwbaar over het kanaal kan worden gestuurd. In dit werk werden de belangrijkste FEC codes overlopen. Beginnend bij eenvoudige codes, die de lezer een duidelijk beeld gaven van hoe met FEC een kanaal beschermd kan worden, werd er verder gebouwd naar complexe technieken waarvan bewezen is dat ze Shannon's limiet zeer dicht weten te benaderen.

Om foutcorrectie te integreren in mediastromen, werden methodes aangehaald die toegespitst zijn op het beschermen van RTP dataverkeer. RTP communicatie is onderhevig aan zowel pakketverlies als aan beschadiging van bits in pakketten die toch arriveren bij de ontvanger. Oplossingen voor beide problemen werden aangeboden. Ook de gevolgen van deze technieken op het netwerk en op compatibiliteit werden onderzocht. In beide gevallen zijn er gunstige mogelijkheden. Sommige technieken laten toe dat ontvangers zonder FEC functionaliteit toch een beschermde mediastroom kunnen decoderen. De extra belasting van het netwerk door toevoeging van FEC data weegt in vele gevallen niet op tegen de verbeteringen die door FEC bereikt worden. Verder bestaan er UEP technieken, die het bandbreedte

verbruik door FEC informatie beperken terwijl ze toch de belangrijkste informatie uit de mediastromen weten te beschermen. Adaptieve technieken laten toe om de bescherming van het dataverkeer op elk moment precies af te stellen op de noden van het netwerk.

Uit dit aanbod van mogelijkheden ter bescherming van mediastromen werden enkele technieken in de praktijk uitgewerkt. Twee technieken werden geïmplementeerd om pakketverlies op te vangen, terwijl met behulp van Reed-Solomon codes werd getracht bitfouten op te vangen. Hieruit kan vooral geconcludeerd worden dat een praktische implementatie minder voor de hand liggend is dan de theorie doet uitschijnen en dat er heel wat technische obstakels moeten worden overwonnen om op een efficiënte manier het probleem van transmissiefouten aan te pakken. Het beschermen tegen bitcorruptie bleek zelfs onmogelijk te zijn op de conventionele Ethernet en WLAN netwerken.

Toch mag ook gesteld worden dat het gebruik van een techniek als de parity code wel degelijk de gevolgen van netwerkstoringen weet te verminderen. Verder bleek dat het technisch mogelijk is om deze bescherming met behulp van de NIProxy in het netwerk zelf te plaatsen, zodat niet langer de zender, maar het netwerk zelf transmissiefouten kan proberen te verbergen voor de eindgebruiker.

Er kan worden geconcludeerd dat FEC technieken geschikt zijn om netwerkcommunicatie van multimedia toepassingen te beschermen tegen transmissiefouten. Hierdoor zal de gebruiker van deze toepassingen een verhoogde Quality-of-Experience ervaren, wat heel wat meerwaarde geeft ten opzichte van applicaties zonder deze bescherming.

Bibliografie

[1] Nadjib Achir, Kave Salamatian, and Guy Pujolle. Object-based unequal loss protection for multiobject video delivery. In *4th EURASIP Conference focused on Video/Image Processing and Multimedia Communications*, volume 1, pages 317–322, July 2003.

[2] Soren Vang Andersen, Alan Duric, Henrik Astrom, Roar Hagen, W. Bastiaan Kleijn, and Jan Linden. Internet low bit rate codec (iLBC). Request for Comments 3951, Internet Engineering Task Force, December 2004.

[3] Mario Baldi and Yoram Ofek. End-to-end delay analysis of videoconferencing over packet-switched networks. *IEEE/ACM Transactions on Networking*, 8(4):479–492, August 2000.

[4] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley. Adaptive FEC-based error control for Internet telephony. In *Proc. of IEEE Infocom*, volume 3, pages 1453–1460, New York, NY, March 1999.

[5] Jean-Chrysostome Bolot and Andrés Vega-García. Control mechanisms for packet audio in the Internet. In *Proc. of IEEE Infocom '96*, volume 1, pages 232–239, San Fransisco, CA, April 1996.

[6] Raj Chandra Bose and Dwijendra Kumar Ray-Chaudhuri. On a class of error-correcting binary codes. *Information and Control*, 3:68–79, 1960.

[7] R. Braden, D. Borman, and C. Partridge. Computing the Internet checksum. Request for Comments 1071, Internet Engineering Task Force, September 1988.

[8] Peter J. Cameron. Galois fields, 2003.

[9] Mark Carson and Darrin Santay. NIST Net: A Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, July 2003.

[10] Ian Chakeres, Hui Dong, Elizabeth Belding-Royer, Allen Gersho, and Jerry Gibson. Allowing errors in speech over wireless LANs. In *4th*

Workshop on Applications and Services in Wireless Networks, pages 142–151, 2004.

[11] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The JPEG2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, November 2000.

[12] Jonny Daenen. Beschermen van netwerktrafiek tegen transmissiefouten. Bachelor eindwerk, Universiteit Hasselt, 2007.

[13] Girla Deepak. A class of non-binary LDPC codes. Master's thesis, Texas A&M University, May 2003.

[14] Kevin Fall and Kannan Varadhan. *The* ns *Manual*. The VINT Project, Januari 2009.

[15] Fernando Silveira Filho, Edson H. Watanabe, and Edmundo de Souza e Silva. Adaptive forward error correction for interactive streaming over the Internet. In *Proceedings of the IEEE Globecom*, pages 1–6, San Francisco, CA, November 2006.

[16] Pascal Frossard and Olivier Verscheure. Joint source/fec rate selection for quality-optimal MPEG-2 video delivery. *IEEE Transactions on Image Processing*, 10(12):1815–1825, December 2001.

[17] Robert G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1960.

[18] Justin Goshi, Alexander E. Mohr, Richard E. Ladner, and Eve A. Riskin. Unequal loss protection for H.263 compressed video. In *Data Compression Conference*, pages 73–82, March 2003.

[19] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, April 1950.

[20] Vicky Hardman, Martina Angela Sasse, Mark Handley, and Anna Watson. Reliable audio for use over the Internet. In *Proc. INET '95*, pages 171–178, Honolulu, HI, June 1995.

[21] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

[22] International ISBN Agency. *ISBN Users' Manual*, fifth edition, 2005.

[23] ITU-T. *Video coding for low bit rate communication: ITU-T Recommendation H.263*, January 2005.

[24] Jennifer D. Key. *Applied Mathematical Modeling: A Multidisciplinary Approach*, chapter Some error-correcting codes and their applications, pages 291–314. Chapman & Hall/CRC Press, 1999.

[25] Rob Koenen. MPEG-4 overview. ISO/IEC JTC1/SC29/WG11 N4668, International Organisation for Standardisation (ISO), 2002.

[26] John Koeter. What's an LFSR? Texas Instruments Incorporated, December 1996.

[27] Philip Koopman. 32-bit cyclic redundancy codes for Internet applications. In *International Conference on Dependable Systems and Networks*, Washington DC, July 2002.

[28] Lars-Ake Larzon, Mikael Degermark, and Stephen Pink. Efficient use of wireless bandwidth for multimedia applications. In *Mobile Multimedia Communications (MoMuC)*, pages 187–193, San Diego, CA, 1999.

[29] Lars-Ake Larzon, Mikael Degermark, Stephen Pink, Lars-Erik Jonsson, and Godred Fairhurst. The lightweight user datagram protocol (UDP-Lite). Request for Comments 3828, Internet Engineering Task Force, July 2004.

[30] Adam H. Li et al. RTP payload format for generic FEC. Request for Comments 5109, Internet Engineering Task Force, December 2007.

[31] Adam H. Li, Jay Fahlen, Tao Tian, Luciano Bononi, So-Young Kim, Jeong-Hoon Park, and John Villasenor. Generic uneven level protection algorithm for multimedia datatransmission over packet-switched networks. In *Tenth International Conference on Computer Communications and Networks*, pages 340–346, 2001.

[32] Günther Liebl, Thomas Stockhammer, and Frank Burkert. Modeling and simulation of wireless packet erasure channels. In *10th Virginia Tech/MPRG Symposium on Wireless Personal Communications*, pages 203–214, 2000.

[33] David MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, fourth edition, 2005.

[34] Madhavi Marka and James E. Fowler. Unequal error protection of embedded multimedia objects for packet-erasure channels. In *Proceedings of the International Workshop on Multimedia Signal Processing*, pages 61–64, St. Thomas, US Virgin Islands, December 2002.

[35] Microsoft Developer Network (MSDN). *AVI File Format.* Microsoft Corporation, 2009.

[36] Alexander E. Mohr, Eve A. Riskin, and Richard E. Ladner. Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction. *IEEE Journal on Selected Areas in Communications*, 18(6):819–828, June 2000.

[37] Todd K. Moon. *Error Correction Coding.* John Wiley & Sons, Inc., 2005.

[38] Samuel P. Morgan. Richard Wesley Hamming (1915-1998). *Notices of the American Mathematical Society*, 45:972–977, 1998.

[39] Mike Nachtegael and Jeroen Buysse. *Wiskundig Vademecum.* Uitgeverij Pelckmans, 2001.

[40] Didier Nicholson, Catherine Lamy-Bergot, Xavier Naturel, and Charly Poulliat. JPEG 2000 backward compatible error protection with reed-solomon codes. *IEEE Transactions on Consumer Electronics*, 49(4):855–860, November 2003.

[41] Chinmay Padhye, Kenneth J. Christensen, and Wilfrido Moreno. A new adaptive FEC loss control algorithm for Voice over IP applications. In *Proc. of IEEE International Performance, Computing and Communication Conference*, pages 307–313, 2000.

[42] Tom Penick. 1's complement and 2's complement arithmetic, 1998.

[43] Colin Perkins. *RTP - Audio and Video for the Internet.* Addison-Wesley, 2003.

[44] Colin Perkins and Jon Crowcroft. Effects of interleaving on RTP header compression. In *Proc. of IEEE Infocom*, volume 1, pages 111–117, 2000.

[45] Colin Perkins, Isidor Kouvelas, Orion Hodson, Vicky Hardman, Mark Handley, Jean-Chrysostome Bolot, Andres Vega-Garcia, and Sacha Fosse-Parisis. RTP payload for redundant audio data. Request for Comments 2198, Internet Engineering Task Force, September 1997.

[46] Irving S. Reed and Gus Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960.

[47] Curtis Roads and John Strawn. *The computer music tutorial*, chapter Basic Concepts of Signal Processing, pages 387–449. MIT Press, 1996.

[48] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A transport protocol for real-time applications. Request for Comments 3550, Internet Engineering Task Force, July 2003.

[49] Steven J. Searle. A brief history of character codes, 1999.

[50] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[51] Amin Shokrollahi. LDPC codes: An introduction, April 2003.

[52] Johan Sjoberg, Magnus Westerlund, Ari Lakaniemi, and Qiaobing Xie. Real-Time Transport Protocol (RTP) payload format and file storage format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) audio codecs. Request for Comments 3267, Internet Engineering Task Force, June 2002.

[53] Bernard Sklar. *Digital Communications: Fundamentals and Applications*, chapter Reed-Solomon Codes, pages 437–460. Prentice-Hall PTR Publishing, 2nd edition, January 2001.

[54] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall PTR Publishing, fourth edition, 2005.

[55] J. Tavares and A. Navarro. Improving error robustness of H.263 bitstreams. In *Conf. on Telecommunications - ConfTele*, volume 1, Figueira da Foz, Portugal, April 2001.

[56] P.N. Tudor. MPEG-2 video compression. *Electronics & Communication Engineering Journal*, 7(6):257–264, December 1995.

[57] Jean-Marc Valin. Speex: A free codec for free speech. In *Linux.Conf.Au*, 2006.

[58] Frederik Vanhaverbeke, Marc Moeneclaey, Koen Laevens, Natalie Degrande, and Danny De Vleeschauwer. On the application of FEC to counter packet loss caused by buffer overflow. In *Symposium on Information Theory in the Benelux*, pages 27–34, 2007.

[59] Shie-Yuan Wang, Chih-Liang Chou, and Chih-Che Lin. *The GUI User Manual for the NCTUns 5.0 Network Simulator and Emulator*. National Chiao Tung University, Taiwan, September 2008.

[60] Yao Wang, Jörn Ostermann, and Ya-Qin Zhang. *Video Processing and Communications*, chapter Waveform-based Video Coding, pages 263–313. Prentice Hall, 2002.

[61] Henry S. Warren. *Hacker's Delight*. Addison-Wesley, 2003.

[62] Henry S. Warren. Cyclic Redundancy Check. Nieuw hoofdstuk voor [61], 2008.

[63] Maarten Wijnants, Tom Jehaes, Peter Quax, and Wim Lamotte. Efficient Transmission of Rendering-Related Data Using the NIProxy. In *Proceedings of the IASTED International Conference on Internet and Multimedia Systems and Applications (EuroIMSA 2008)*, Innsbruck, Austria, March 2008.

[64] Maarten Wijnants and Wim Lamotte. Audio and Video Communication in Multiplayer Games through Generic Networking Middleware. In *Proceedings of the 7th International Conference on Computer Games (CGAMES 2005)*, pages 52–58, Angoulême, France, November 2005.

[65] Maarten Wijnants and Wim Lamotte. The NIProxy: a flexible proxy server supporting client bandwidth management and multimedia service provision. In *Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, Helsinki, Finland, June 2007.

[66] Jonathan S. Yedidia, William T. Freeman, and Yair Weis. *Exploring Artificial Intelligence in the New Millenium*, chapter Understanding Belief Propagation and its Generalisations, pages 239–271. Elsevier Science & Technology Books, January 2003.

[67] W. Zhao, R. Chellappa, A. Rosenfeld, and P.J. Philips. Face recognition: A literature survey. *ACM Computing Surveys (CSUR)*, 35(4):399–458, December 2003.

Bijlagen

Bijlage A

Het Galois veld $GF(2^3)$

Symbool	Veelterm	Bits	Symbool	Veelterm	Bits
0	0	000	α^3	$1+x$	110
α^0	1	100	α^4	$x+x^2$	011
α^1	x	010	α^5	$1+x+x^2$	111
α^2	x^2	001	α^6	$1+x^2$	101

Tabel A.1: *Overzicht van $GF(2^3)$. De voorstelling als veelterm is zoals beschreven in vergelijking 3.12.*

$+$	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
0	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	α^0	0	α^3	α^6	α^1	α^5	α^4	α^2
α^1	α^1	α^3	0	α^4	α^0	α^2	α^6	α^5
α^2	α^2	α^6	α^4	0	α^5	α^1	α^3	α^0
α^3	α^3	α^1	α^0	α^5	0	α^6	α^2	α^4
α^4	α^4	α^5	α^2	α^1	α^6	0	α^0	α^3
α^5	α^5	α^4	α^6	α^3	α^2	α^0	0	α^1
α^6	α^6	α^2	α^5	α^0	α^4	α^3	α^1	0

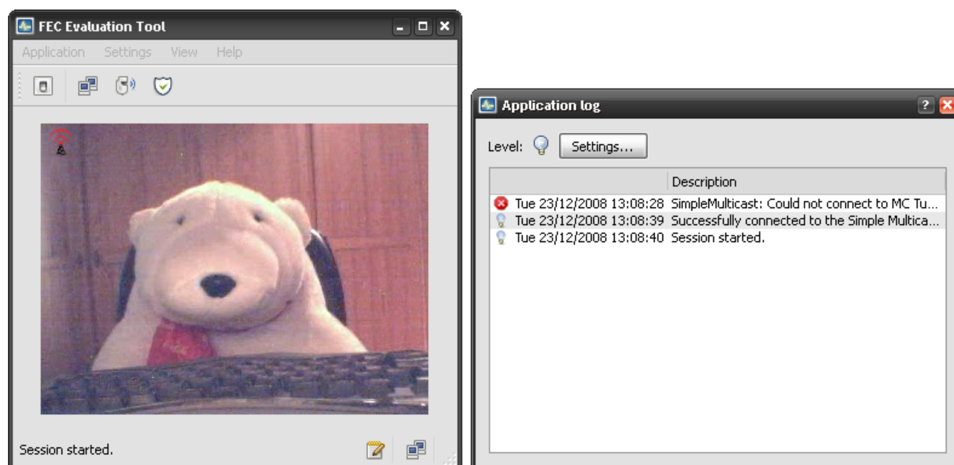
Tabel A.2: *Optellingstabel van $GF(2^3)$.*

$\times$	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
0	0	0	0	0	0	0	0	0
α^0	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^1	0	α^1	α^2	α^3	α^4	α^5	α^6	α^0
α^2	0	α^2	α^3	α^4	α^5	α^6	α^0	α^1
α^3	0	α^3	α^4	α^5	α^6	α^0	α^1	α^2
α^4	0	α^4	α^5	α^6	α^0	α^1	α^2	α^3
α^5	0	α^5	α^6	α^0	α^1	α^2	α^3	α^4
α^6	0	α^6	α^0	α^1	α^2	α^3	α^4	α^5

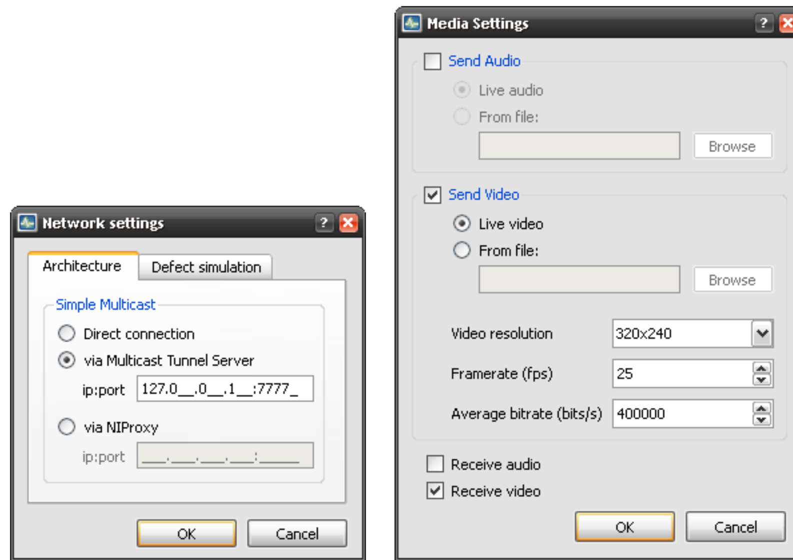
Tabel A.3: *Vermenigvuldigingstabel van* $GF(2^3)$.

Bijlage B

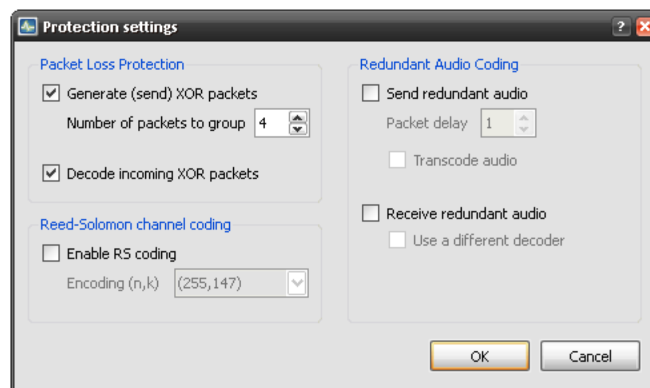
GUI screenshots



Figuur B.1: *Screenshot van de applicatie. Links wordt het hoofdvenster weergegeven. Rechts de applicatielog.*



(a) Netwerkinstellingen (b) Media instellingen



(c) FEC instellingen

Figuur B.2: *De vensters waarmee de applicatie kan worden ingesteld.*

Bijlage C

EMIPLIB componentkettingen

In deze bijlage wordt de lezer verondersteld vertrouwd te zijn met de interne werking van EMIPLIB. De componentkettingen die verantwoordelijk zijn voor de invoer en uitvoer van audio en video in de applicatie, besproken in hoofdstuk 5, worden toegelicht. Er zijn vier kettingen: voor zowel audio als video is er een aparte invoer- en uitvoerketting.

De applicatie is een Windows applicatie. Bijgevolg bevatten de kettingen enkel componenten voor dit platform.

C.1 `AudioInputChain`

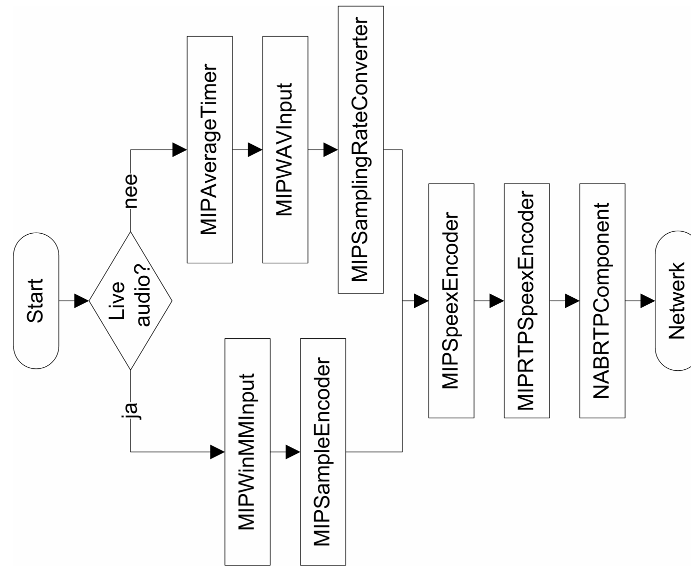
De invoerketting voor audio (figuur C.1) kan van twee verschillende bronnen informatie verzamelen: live audio wordt via de Windows Multimedia API verworven, terwijl ook Waveform bestanden kunnen ingelezen worden. Na het comprimeren met de Speex encoder worden de audiopakketten aan de RTP laag doorgegeven.

C.2 `AudioOutputChain`

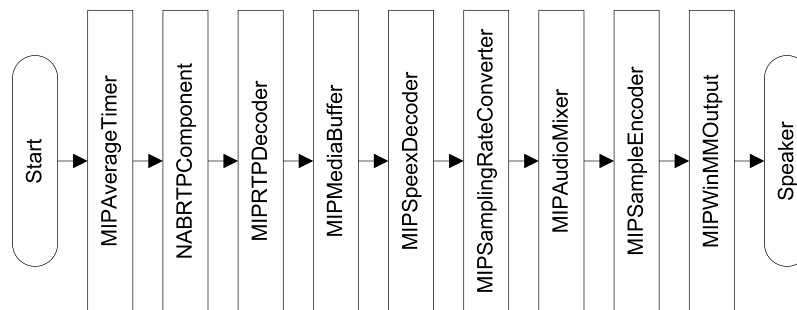
De uitvoerketting voor audio (figuur C.2) bevat niets speciaal. Enkel de basiselementen voor het decoderen en afspelen van Speex audiopakketten zijn aanwezig. De `MIPRTPDecoder` bevat een extra `MIPRTPPacketDecoder` voor onbekende *Payload Types*. Deze worden genegeerd.

C.3 `VideoInputChain`

De invoerketting voor video (figuur C.3) kan net als zijn audio variant van twee verschillende bronnen informatie verzamelen. Live videobeelden wor-



Figuur C.1: *Invoerketting voor audio*

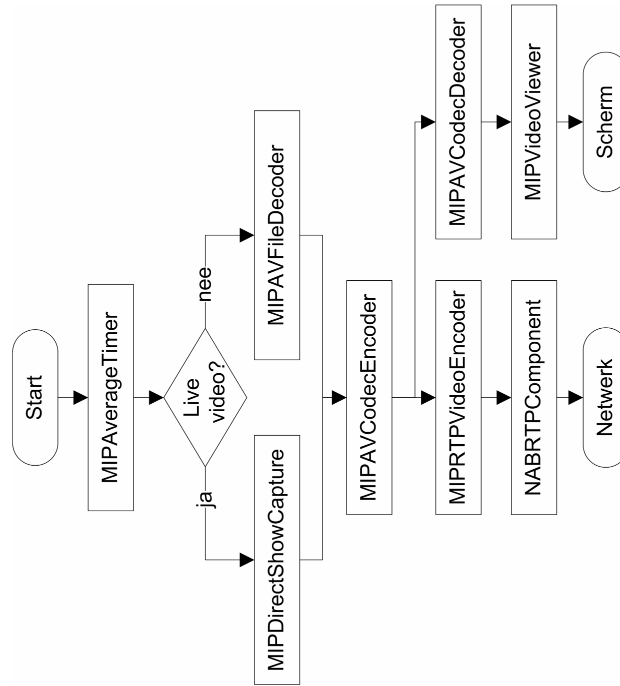


Figuur C.2: *Uitvoerketting voor audio*

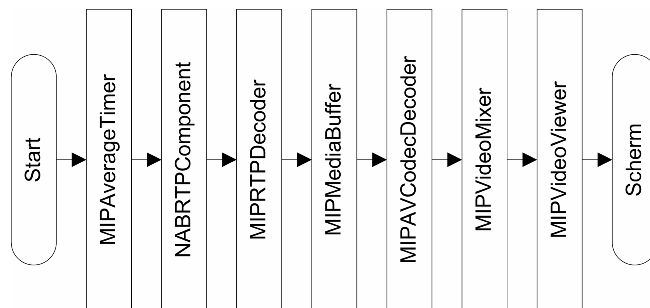
den via de Microsoft DirectShow API ingelezen. `MIPAVFileDecoder` is een deel van de uitbreiding van EMIPLIB die werd ontwikkeld (zie sectie 5.3.2) om ook beelden uit een bestand te kunnen gebruiken. Beelden worden vervolgens gecomprimeerd met de H.263+ encoder en aan de RTP laag doorgegeven. Er is een bypass voorzien om de beelden ook op het scherm weer te geven, zodat de zender ziet wat hij uitzendt. Deze bypass is bewust na de video encoder geplaatst, zodat de beelden die getoond worden dezelfde decoder artefacten hebben als bij de ontvanger (in de veronderstelling dat alle pakketten arriveren). Dit vereenvoudigt het vergelijken van de beelden van zender en ontvanger. De `MIPVideoViewer` component vormt de koppeling tussen EMIPLIB en de GUI.

C.4 `VideoOutputChain`

De uitvoerketting voor video (figuur C.4) heeft dezelfde basisfunctionaliteit als de `AudioOutputChain`, maar dan voor video. RTP pakketten die geen videobeelden bevatten worden genegeerd. De `MIPVideoViewer` component zorgt ervoor dat de GUI de gedecodeerde beelden aan de gebruiker toont.



Figuur C.3: *Invoerketting voor video*



Figuur C.4: *Uitvoerketting voor video*