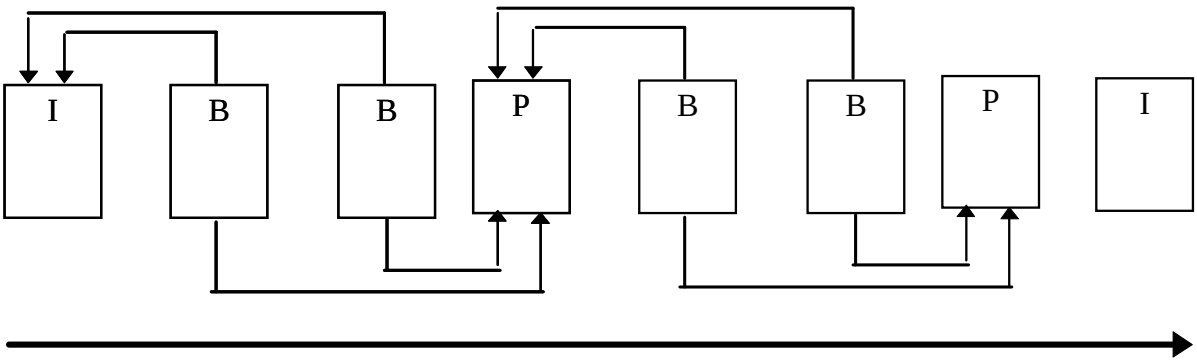
NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
FAKULTET FOR ELEKTROTEKNIKK OG TELEKOMMUNIKASJON

HOVEDOPPGAVE

Videotjener med VCR-funskjonalitet for MPEG-1



Roger Koteng

20. desember 1996

Sammendrag

Denne rapporten presenterer resultatet av en hovedoppgave utført ved Institutt for datateknikk, Norges teknisk-naturvitenskapelige universitet.

Hensikten med oppgaven har vært å finne metoder for å få til VCR-funksjonalitet i MPEG-1 video. Med VCR-funksjonalitet menes de mest vanlige funksjonene man finner på en vanlig videospiller for bevegelse i en video, slik som spoling. Basert på disse metodene er det konstruert en videotjener. En prototype av videotjeneren er implementert for å teste ut de forskjellige metodene, og målinger som sammenligner metoden blir presentert.

For å finne disse metodene for VCR-funksjonalitet er MPEG-1 standarden studert. Man har også sett på rapporter om andres måter å løse dette. Tre hovedmetoder for spoling er funnet. Likeså metoder for avspilling bakover , saktefilm og tilfeldige hopp i videoen.

En ny videotjener som tar utgangspunkt i metodene som er funnet, er konstruert. Noen av komponentene brukt i videotjeneren er basert på en tidligere hovedoppgave [Langørgen-94].

En prototype av videotjeneren er implementert, med noen forenklinger i forhold til konstruksjonen. All VCR-funksjonalitet er implementert i videotjeneren slik at det har vært mulig å måle de forskjellige spolemetodene.

Målingene viser at bruk av spolefil ved spoling er den beste metoden for å støtte VCR-funksjonalitet i en videotjener, når man tar hensyn til de mest kritiske ressursene der. De mest kritiske ressursene i en videotjener er funnet til å være diskforbruk og nettverksbelastning. Spoling med spolefil vil kunne gi en god spolekvalitet ved ideelle spolehastigheter for den spesielle filen, men dårligere ved andre hastigheter. Særlig spoling med lave hastighet kan gi dårlig spolekvalitet fordi rammeraten vil være lav. For å bedre spolingen i dette området kan man benytte flere spolefiler, der hver spolefil dekker hvert sitt spolehastighetsområde.

Forord

Denne rapporten er resultatet av en diplomoppgave utført ved institutt for datateknikk, Norges teknisk-naturvitenskapelige universitet. Arbeidet som er utført er tilknyttet LAVA-prosjektet.

Oppgaven går ut på å finne metoder for å få til VCR-funksjonalitet i MPEG-1-video. Dette skal så realiseres i en videotjener. Rapporten inneholder beskrivelse av metoder som er funnet, konstruksjon av videotjener og målinger gjort på en implementert prototype av videotjeneren.

Jeg vil takke veileder Olav Sandstå og faglærer Roger Midtstraum, for alle råd og tips jeg har mottatt under arbeidet med oppgaven. Spesielt takk til Olav for han innsats og entusiasme under hele oppgavens varighet.

Trondheim, 20. desember 1996

_____

Roger Koteng

Innhold

1. Innledning

Behovet for overføring av video via nettverk over lange distanser, blir stadig større. Dette vises best ved at det forskes mer enn noen gang på denne typen teknologi. Dessverre har vi ennå ikke noen gode løsninger for dette over dagens nett. Hovedgrunnen til det er at videodata er svært store og krever derfor høy båndbredde for å oppnå tilfredsstillende overføringer. Man har muligheter for å sende bilder og lyd over det vanlige telefonnettet, men kvaliteten på videoen blir dårlig, på grunn av den lave båndbredden i systemet. Bruken av denne typen overføringer er i dag begrenset til overføring av levende bilder f.eks. i forbindelse med videokonferanser, der man kan greie seg med den kvaliteten som tilbys. Denne kvaliteten er generelt ikke god nok for overføring av video, der faste rammerater og båndbredder ofte er bestemt.

Med de senere års utvikling av nettverksteknologi, har muligheten for overføring av video over nett i sanntid blitt mulig. Der er særlig den nye nett teknologien ATM som gjør det mulig å få til dette. Den har en overføringskapasitet på 155 Mbit/s eller mer per forbindelse, og har samtidig muligheter for å reservere båndbredde på nettet slik at man sikrer stabile leveringstider for data. Reservering av båndbredde kan bli en nødvendighet i fremtidens overfylte nettverk, for å sikre de faste bitrater som er nødvendig for video. Dessverre finnes det ennå ikke store nettverk basert på denne teknologien tilgjengelig for allmennheten i dag.

For overføring av video over nett, er det i de fleste tilfeller nødvendig å komprimere videodataene så mye som mulig uten at det går for mye utover bildekvaliteten. Ved å komprimere videoen, kan man overføre video der båndbredde-begrensningen på nettet ellers ville hindre det. For komprimering av video med bitrater mellom 300 kbit/s og 1.5 Mbit/s, er det en ledende standard i dag. Denne standarden er MPEG-1. Den er ledende først og fremst fordi den gir den beste kompresjonen i forhold til bildekvalitet og bitrate. For å gi best mulig komprimering er den optimalisert for normal avspilling. Dette gjør den lite egnet til andre avspillingsformer som f.eks spoling og baklengs avspilling, fordi den bruker komprimeringsmetoder som vanskeliggjør disse.

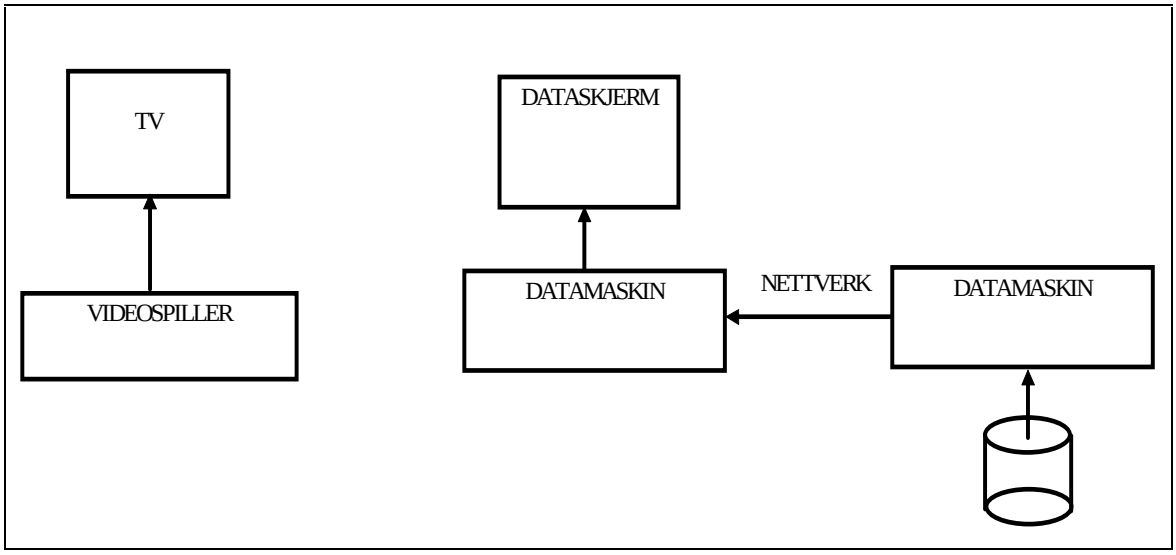
Ved å kombinere bruk av ATM og MPEG-1 i en videotjener kan man lever mange samtidige videoer over nettet. Dette kan f.eks. benyttes i video on demand tjenester, der man leverer videofilmer til private hjem etter deres ønske. Det kan også benyttes i et sentralisert videoarkiv der flere skal ha tilgang til videoene fra ulike lokasjoner. I disse tjenestene vil man ha behov for å kunne bevege seg fritt i en video.

Denne hovedoppgaven tar for seg ulike metoder for å oppnå VCR-funksjonalitet i MPEG-1-kodet video. Den tar også for seg utviklingen av en videotjener som benytter dette ved leveranser av video over et ATM-nettverk. Denne oppgaven har ikke som mål å lage en høykapasitets videotjener der det eneste målet er å levere flest mulig videostrømmer til flest mulig brukere. I denne oppgaven skal det lages en videotjener der det fokuseres på bruk av MPEG-1 videokomprimering og at man kan tilby de mest vanlige funksjonene som finnes på en normal videomaskin for bevegelse i videomaterialet. Dette betegnes som VCR-funksjonalitet.

Oppgaven er utført ved Institutt for datateknikk ved NTNU i Trondheim høsten 1996.

2. Problemspesifikasjon

Vi vil her forklare hva som skal løses i denne oppgaven, og hva som gjør denne så vanskelig at den er verdig en hovedoppgave.



Figur 2.1 Leveranse av video.

Målet med en slik oppgave er å kunne presentere video på en dataskjerm fra en videotjener over et nett, på en måte som ligner presentasjon av video på en TV fra en videomaskin (se Figur 2.1). Man kan si at man bytter ut TV'en med en dataskjerm, videomaskinen med en eller flere datamaskiner som utgjør en videotjener, og ledningen i mellom TV og videoen med et datanettverk. Denne beskrivelsen er nok noe forenklet, men det illustrerer hva som skal gjøres. Hvorfor skal man da lage et slik system, når det virker mye enklere med TV og video? Hva er fordelen med det nye systemet? En av fordelene med det nye systemet er at man kan levere flere videofilmer samtidig til ulike brukere. Man har som bruker fordelen av å kunne velge i et stort utvalg av filmer, og motta bilder fra disse med engang valget er gjort.

Følgende oppgaver skal løses:

Avspilling av MPEG-1-video over ATM-nettverk skal oppnåes.
VCR-funksjonalitet over ATM-nettverk for MPEG-1-video skal konstrueres.
Belastning på nett skal ikke øke ved endring av avspillingsform.

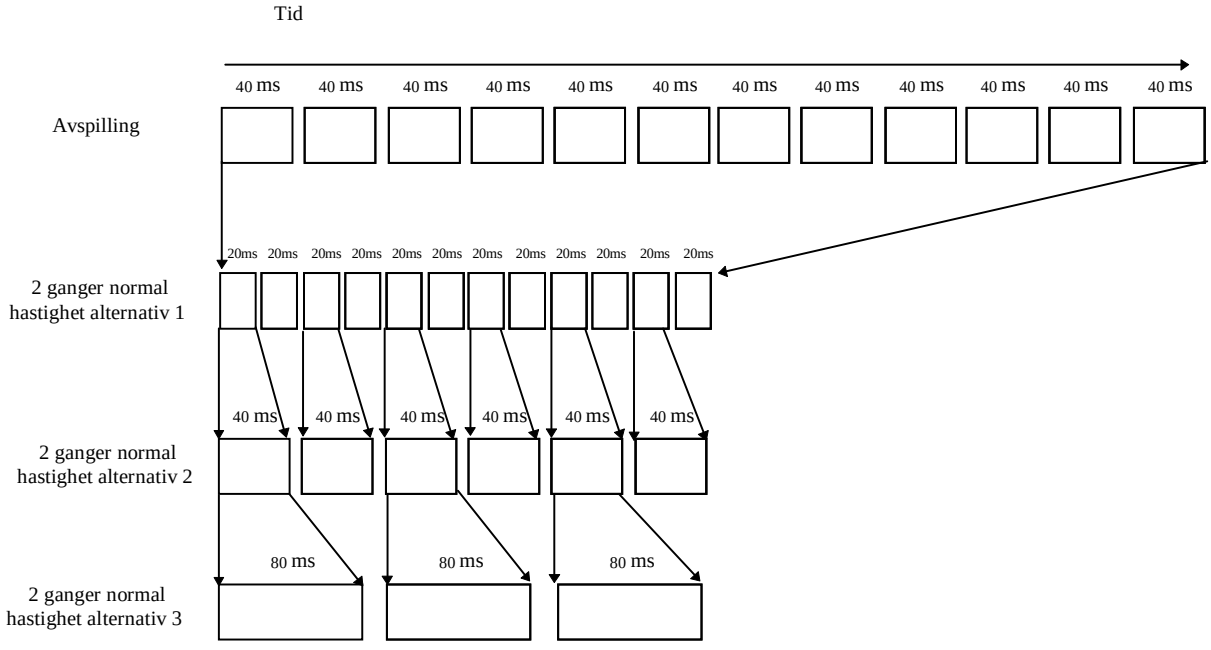
Med VCR-funksjonalitet menes de mest vanlige funksjoner som finnes på en videomaskin. Dette innbefatter:

- Vanlig avspilling av video med lyd.
- Spoling av video i ulike hastigheter, i begge retninger, uten lyd.
- Sakte avspilling i begge retninger uten lyd.
- Bilde for bilde.
- Stillbilde.
- Tilfeldige hopp i videoen.

2.1 VCR-funksjonalitet i komprimert video

Hva er det som gjør MPEG-1 til en god komprimeringsstandard for TV-overføringer, men lager vanskeligheter så fort man snakker om andre avspillingsmuligheter enn vanlig avspilling og bilde for bilde. Hvis du har en MPEG-1 avspiller tilgjengelig kan du se hvilke VCR-muligheter du har på denne. Det er stor sannsynlighet for at du ikke finner muligheter for hurtig spoling fremover og bakover, sakte film, revers osv. Grunnen til dette er at MPEG-1 er laget for at hvert bilde i filmen skal vises ved avspilling. Prøver man å hoppe over et bilde er sjansene store for at man får noen merkelig forstyrrelser i filmen som arter seg som små ruter med utydelig informasjon på skjermen. Bildene i MPEG-1 er nemlig ikke kodet hver for seg slik at man bare kan plukke ut hvert tiende bilde og sette dette sammen til en videostrøm og derved få til ti ganger spoling. Man har brukt en komplisert algoritme for kodingen for å få bildene komprimert mest mulig og har derved ødelagt noen av muligheten som finnes ved andre komprimeringsstandarder. Motion-JPEG er en slik standard som har samme muligheter som ikkekomprimert video til å bevege seg fritt i film-materialet og å hoppe inn på hvilken som helst ramme uten at man får problemer med dekoding og visning. I MPEG-1 må man ved hopp i filmen, treffe nøyaktig på riktig plass for å oppnå et hopp som kan godtas av en dekoder, og man kan normalt bare hoppe til bestemte punkter i filmen.

2.1.1 Hvordan oppnå spoling



Figur 2.2 Muligheter for å få til spoling.

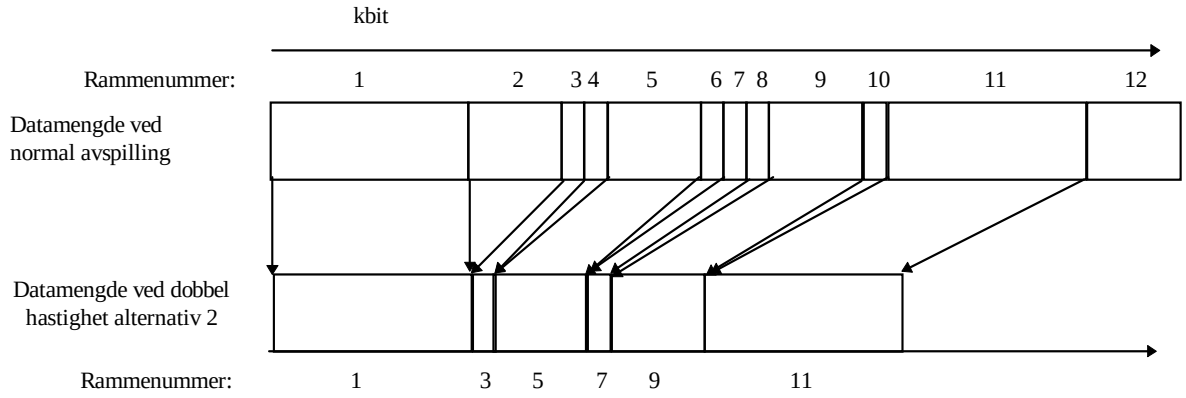
Øverst i Figur 2.2 ser vi hvordan rammer vises i tid for vanlig avspilling. Hver ramme vises i 40 ms. Det tilsvarer en bildefrekvens på 25 rammmer/s, som er samme bildefrekvens som i et video-system av type PAL. Figur 2.2 viser også tre forskjellige alternativer for å få til spoling med dobbel hastighet.

Det første alternativet viser den metoden som benyttes på vanlige videomaskiner. Man viser alle bildene i halvparten av tiden i forhold til normalt og oppnår derved en spoleeffekt. Fra en

videomaskin blir det ofte forstyrrelser på bildet av dette fordi man ikke klarer å styre bildehode like nøyaktig ved høyere hastigheter som ved normal avspilling. Denne metoden øker datamengden som blir overført per sekund, og vil derfor øke belastningen på harddisk og nettverk i en videotjener. Den viser all informasjon i videoen og dette gjør denne metoden til den beste hvis vi kun tenker på kvalitet.

Alternativ to er kanskje den mest logiske metoden å bruke for spoling i digital video. Her vises annenhvert bilde. Det gjør at man kan vise hvert bilde like lenge som ved normal avspilling og likevel oppnå en spoleeffekt. Datamengden som skal presenteres er like stor som ved vanlig avspilling, men noe av informasjonen i videoen går tapt, siden ikke alle bildene vises. Dette er den optimale metoden å bruke for en videotjener, hvis man ikke trenger å ta hensyn til komprimeringsmetode, fordi den gir den samme belastningen på nettverket som ved vanlig avspilling.

Alternativ tre viser en metode der man kun viser hvert fjerde bilde for spoling med dobbel hastighet. Med denne metoden vises hvert bilde dobbelt så lenge som ved normal avspilling. Vi vil nå oppnå at datamengden som presenteres ved spoling er mindre enn ved vanlig avspilling. Hva er poenget med dette da? Hvis man allikevel har mulighet for å sende mer informasjon en det man gjør hvorfor senke kvalitetsopplevelsen til seeren. Løsningen ligger i komprimesringsmetoden som benyttes. Vi forutsatte over at hver ramme ble representert med like stor data mengde. Dette er ikke alltid riktig. Særlig når rammene har blitt komprimert. Det vises i Figur 2.3, hvor størrelsen på hver ramme i kilobit illustreres. Rammestørrelsene som vises her er typiske for MPEG-1 video. Vi ser at selv om man plukker ut annen hver ramme vil ikke datamengden nødvendigvis bli halvparten så stor. Plukker man derimot ut færre rammer er sannsynligheten større for å halvere datamengden, men helt sikker kan man ikke være.



Figur 2.3 Rammer i forhold til rammestørrelse.

2.1.2 Bildeavhengigheter

For å øke komprimeringen kan man ved bildekomprimering utnytte at etterfølgende bilder ofte er svært like. Det får man til ved å kun kode forskjellene på bildet i forholde til det foregående bildet, isteden for å kode hele bildet. Man kan da oppnå en høyere komprimering. Dette vil imidlertid lage problemer ved spoling. Hvis vi ser på Figur 2.2 ser vi at vi nå vil få problemer med å bruke alternativ 2 og 3, som fra tidligere har sett ut som de mest attraktive

metodene. Vi risikerer å plukke ut rammer som er kodet i forhold til rammer som ikke plukkes ut. Disse rammene kan dermed ikke dekodes, og det resultatet blir slett ikke slik vi hadde tenkt at det skulle bli.

2.2 Videotjener

For en videotjener er det essensielt at levering av video er så lite ressurskrevende som mulig. I tillegg til at det skal være lite ressurskrevende er det også nødvendig at man har en jevn bruk av ressurser i ulike faser av levering, slik at det er lett å beregne hvor mange leveranser et system tåler uten å ta hensyn til når leveranser oppstår, og hva som kreves av hver enkelt leveranse. En lite resurskrevende video er, for det første en video som er så liten i datamengde som mulig. Det gjør at man kan lese mindre data fra disk per leveranse, og mindre data blir sendt over nett per leveranse. I tillegg bør videoen være lett å manipulere hvis man ønsker å levere video på forskjellige måter. For å være lett å manipulere bør rammene i videoen være uavhengig av hverandre. MPEG-1 er den komprimeringsstandarden som gir minst datamengde for de fleste videofilmer. Den er imidlertid ikke like enkel å manipulere som andre komprimeringsstandarder. Det blir derfor en oppgave her å finne en måte å manipulere en MPEG-1 video uten å bruke for mye resurser i videotjeneren, samtidig som kvaliteten på leveransen til klienten ikke blir for dårlig.

2.3 Beskrivelse av hva som skal gjøres

Vi har i delkapitel 2.1 pekt på problemer som kan oppstå for komprimert video, i forhold til ukomprimert video. Vi skal nå definere problemene med VCR-funksjonalitet spesifikt mot MPEG-1-video.

MPEG-1-video inneholder alle de problemelementene som er beskrevet i delkapittel 2.1. Den inneholder rammer med ulike størrelse, der størrelsen varierer med opptil 6 ganger. Den har også avhengigheter mellom ulike rammer i videoen. Vi ønsker med dette utgangspunktet å finne metoder for å unngå fellene, som vist over. Man skal klare å finne nye metoder for få til spoling. Hvis flere ulike metoder finnes, skal disse settes opp mot hverandre. Kriterier for sammenligning er blant annet bildekvalitet og rammehastighet.

For å teste ut de ulike metodene for VCR-funksjonalitet skal det konstrueres og implementeres en videotjener. Videotjeneren skal lages etter prinsipper om en jevnest mulig belastning av ressurser, minst mulig ressursbruk per videoleveranse og et utvalg av de metodene som er funnet skal implementeres i den. Dette gir et nytt kriterier for sammenligning av metoder nemlig ressursforbruk i videotjeneren. Dette kan vise seg å være det viktigste kriteriet for levering av video med VCR-funksjonalitet fra en videotjener.

3. Bakgrunn

I dette kapittelet vil den teknologien som er funnet tilgjengelig for bruk i forbindelse med en videotjener presenteres. Dette omfatter ulike typer/formater av digital video, hvilke nettverksløsninger som er tilgjengelig, hvilke løsninger som er utprøvet og hvilke erfaringer andre har fått ved design av en videotjenere. Det vil først bli vist ulike måter å lagre digitalvideo, så vil vi se på audio, deretter vil vi nevne de ulike nettverksteknologier som kan benyttes for til slutt å komme med noen eksempler på videotjenere som brukes i dag.

3.1 Digitale bilder

Figur 3.1 Fra tv-serien Schrødingers katt. Oppløsningen er 100x80 punkter og 24 bits fargekode.

(Prøv å myse for å se hva bildet forestiller.)

For å overføre et bilde til en datamaskin må bildet digitalisere. Bildet blir delt opp i punkter der hvert punkt representeres ved en bitkode. Denne oppdelingen kan sees på Figur 3.1. Bitkoden forteller hvilken farge og intensitet dette ene punktet skal ha. Denne koden har normalt en størrelse på fra 8 bit til 24 bit. For et naturtro fotografi bør bildekoden være 16 bit eller mer. Dette for å gi nok fargenyanser til at øyet vårt godtar det som et fotografi. En 16 bits kode utgjør ca.65000 farger/fargenyanser. I tillegg teller antall punkter som benyttes i bildet. Jo flere punkter som benyttes til å beskrive bilde jo nøyaktigere og klarere blir bildet.
Det er vanlig å kalle denne samlingen av punkter et bilde utgjør for en punktmatrise. Størrelsen på denne punktmatrisen varierer alt etter hvor stort bildet skal være og hvor høy kvalitet man ønsker på bildet. For å kunne gi et detaljert bilde som kan sammenlignes med bildet på en TV-skjerm (PAL) vil punktmatrisen være på 720x575 punkter. Det gir 414.000

enkeltpunkter. Ved å benytte 24 bit eller 3 byte fargekode til hvert punkt betyr det at man trenger ca 1,2 MByte for å lagre et TV-bilde i PAL-kvalitet på en datamaskin. I Figur 3.2 ser vi noen eksmpler på TV/video-standarder og hvor stor lagringskapasitet disse trenger.

TV/Video-standard	*Oppløsning*	*datavolum/ bilde v.24 bit*	*Bildefrekvens*	*Datamengde pr. sekund*
PAL(Europa) og SECAM (Frankrike)	maks 830x625 standard 720x575	standard 1.242.000 byte	25 bilder/s (50 Hz interlaced)	standard 31.050.000 byte
NTSC (USA og Japan)	maks 700x525 standard 600x480	standard 864.000 byte	30 bilder/s (60 Hz interlaced	25.920.000 byte
HDTV (High Definition TeleVision) NHK	maks 2220x1250 standard 1920x1080	standard 6.220.800 byte	25 bilder/s (europa) 30 bilder/s (Nord amerika)	152.520.000 byte 186.624.000 byte
VHS	standard 300x240	216.000 byte	Avhengig av TV-standard	v/25 bilder/s 5.400.000 byte
S-VHS	standard 565x425	720.375 byte	Avhengig av TV-standard	v/25 bilder/s 18.009.375 byte

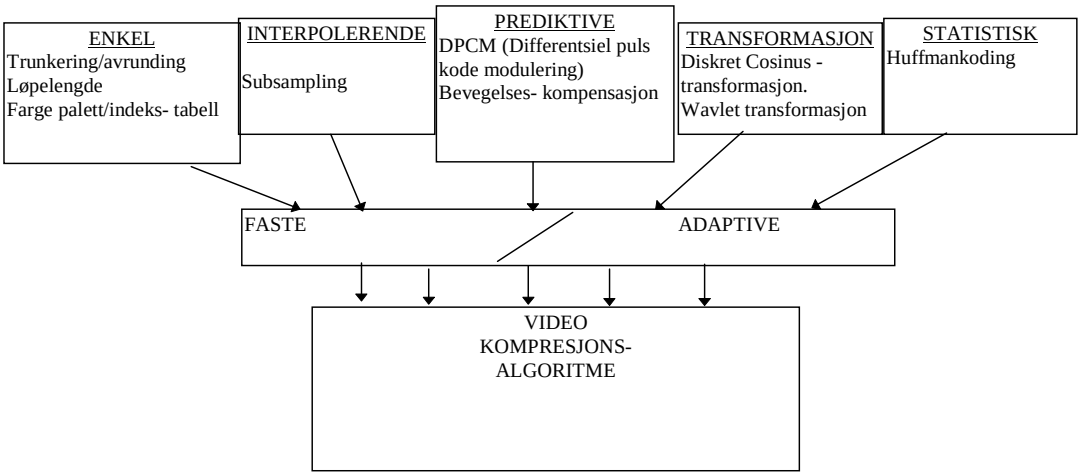
Figur 3.2 Oversikt over TV-/video-standarder bla. fra [Hodge-94].

3.2 Digital video

Ved digital video vil digitale bilder bli vist etter hverandre i en hastighet mellom 20 og 30 bilder i sekundet alt etter hvilket videoformat bildene er hentet fra (PAL gir 25 bilder i sekundet, NTSC gir 30 bilder i sekundet og vanlig kinofilm gir 24 bilder i sekundet). Hvis man skal lage en sekvens av bilder med PAL-kvalitet så vil hvert enkeltbilde okkupere ca. 1,2 MByte lagringsplass som vist i Figur 3.2. PAL-kvalitet tilsier 25 bilder i sekundet. Dette gir en datamengde på ca. 30 MByte/s. En langfilm på 90 minutter, ville da utgjøre ca. 162 GByte med bildedata. I tillegg kommer lyddataene som utgjør ca. 1 GByte. Med dagens lagringsteknologi vil dette kreve et harddisksystem med mellom 10-100 disker avhengig av kapasiteten på hver enkelt harddisk.

Kostnaden for et slikt system ville ligge på (sept 1996) ca. 1000,-/GByte. Det betyr 162.000,- for å lagre en film. Man kunne lagret filmen billigere på digital bånd der prisen ligger omlag på 50 kr/GByte, men dette krever en forhåndsinvestering i båndspillere som gir en pris langt dyrere enn 162.000,- for *en* film, men gir lavere kostnad ved lagring av flere filmer. Selv med nettoprisen på 8.100,- pr. film (kun båndkostnaden) er meget høy. Alle disse beregninger er sett med dagens priser. Prisene vil nok synke etter som man finner andre og billigere teknologier for lagring av data.

3.3 Komprimering av bildedata



Figur 3.3 Oversikt over komprimeringsteknikker hentet fra [Koegel-94].

Hvorfor man ønsker å komprimere bildedata er forklart over. Det er alt for store datamengder tilknyttet en ikke komprimert video. Det koster også altfor meget å lagre disse mengdene med data. I tillegg vil overføringsbegrensninger i nettverk umuliggjøre praktisk overføring av video over de nettverksstandardene som foreligger i øyeblikket.

Grunnen til at det er mulig å komprimere et bilde slik at det tar mindre plass er at bildet inneholder redundant informasjon. Man kan også fjerne bildedata som er visuelt ubetydelig. Det betyr at det menneskelige øyet ikke legger merke til at vi fjerner disse data. Spesielt er dette mulig ved levende bilder da øyet er mest opptatt av bevegelser i bilde og ikke små detaljer. Dette kommer vi tilbake til senere.

Komprimering av bildedata er det forsket mye på i mer en 25 år. Det er derfor i følge [Kogel-94] vanskelig å tenke seg at det kommer noen nye og revolusjonerende teknikker som er mye bedre enn de som finnes i dag. Som vist i Figur 3.3 er ofte en bildekompresjonsalgoritme bygd opp av flere komprimeringmetoder. Vi vil her gjennomgå de mest brukte metodene for bilde/video-komprimering.

3.3.1 Tapsfri vs. ikke tapsfri komprimering

Man kan dele opp komprimeringsmetoder i to hovedklasser. De kalles tapsfri og ikke-tapsfri komprimering. Ved tapsfri komprimering vil dataene som komprimeres for så å dekomprimeres være identisk med de originale dataene. Ved ikke-tapsfri komprimering blir visse data borte. For vanlig tekst og f.eks. maskinkode kan man ikke benytte annet en tapsfri komprimering. Her er det essensielt at dataene ikke forandres eller blir borte under komprimering. For bildedata derimot kan man benytte seg av noen typer ikke-tapsfri komprimering, fordi de detaljene som forsvinner er såpass små at de betraktes som visuelt ikke signifikant og vil i de aller fleste tilfeller ikke legges merke til at er borte.

Ved tapsfri komprimering kan man oppnå en komprimeringgrad på ca 3 ganger for bildedata[Gonzalez-92]. Ved ikke-tapsfri komprimering for et bilde, kan man komme ned i faktorer på mer enn 20 ganger [Gonzalez-92]. Hvis man i tillegg tar med komprimering ved

hjelp av likheter i bilder for en video, kan man tjene ytterligere 2/3 deler i forhold til et enkeltstående bilde. Det vil da gi en mulig komprimering på over 60 ganger. Siden tapsfri komprimering har en så liten komprimeringsgrad vil ikke dette være aktuelt å bruke det for komprimering av video.

3.3.2 Løpelengdekoding (tapsfri) [Murray-94]

Den enkleste komprimeringsmetoden for data er løpelengdekoding. Der setter man sammen like etterfølgende verdier til denne verdien ganget med antall repetisjoner av verdien. Dette fungerer bra i ensformige datamengder som en tegning/bilde der mesteparten av tegningen er i en farge, eller i en tekst der teksten består av mange like tegn.

Eksempel :

AAAABBBBBCCCD lagres som A4B5C3D1

3.3.3 Huffmankoding (tapsfri) [Murray-94]

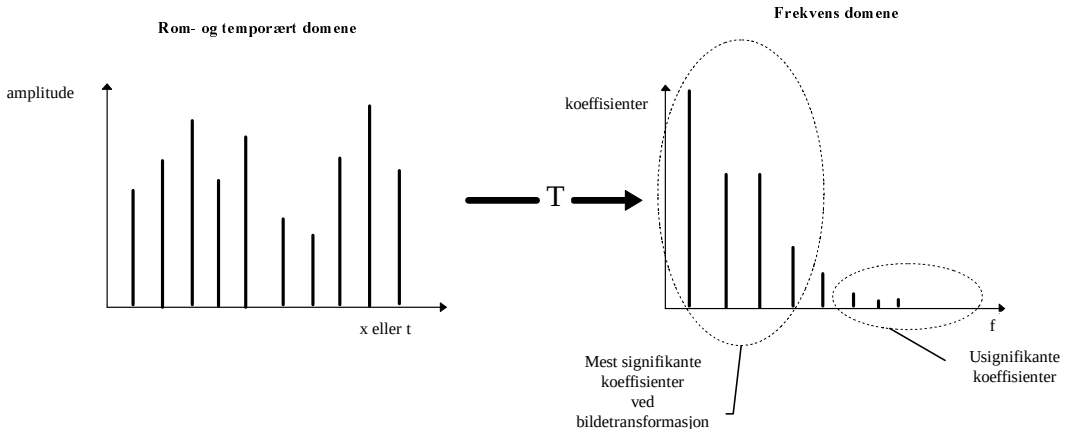
En litt mer avansert metode er huffmankoding. Ved huffmankoding har man en tabell med forutbestemte datakombinasjoner som statistisk skal være de hyppigst forekommende i den type data som skal komprimeres eller man bygger opp denne tabellen mens man koder dataene. Hver kombinasjon har et nummer, slik at man lagrer dette nummeret i stedet for datakombinasjonen. De mest vanlige kombinasjoner ligger på lavest nummer og trenger derfor minst lagringskapasitet.
Dette gir en mer fleksibel kodingsmetode enn løpelengde koding og kan derfor benyttes på flere typer data.

3.3.4 Fotografi- og videospesifikk koding

For bildedata som fotografier og video, er ikke metodene over alene effektive nok. Dette kommer av at et fotografi er så fullt av fargenyanser og datakombinasjoner som er uforutsigbare og lite repetitive. Det er derfor utviklet andre komprimeringsmetoder som har vist seg å være spesielt effektive for fotografier og video. Det finnes nemlig redundans i bilde data som gjør bilde komprimerbart. Denne redundansen kan deles opp i tre typer [Hilton-94]:

- Rom-redundans
  I nesten alle naturlige bilder vil verdien til nabopunkter være sterkt korelaterte.
- Spektral redundans
  I bilder bestående av mer en et spektralt bånd vil verdiene til den samme piksel-lokasjonen ofte være korelaterte.
- Temporal redundans
  Etterfølgende rammer i en videosekvens vil ofte forandre seg lite.

Fjerning av rom- og spektral redundans gjøres ofte ved hjelp av transformasjon. De mest kjente av disse er diskret cosinus transformasjon og wavelet-transformasjon. Komprimering ved hjelp av transformasjon skjer ved at dataene flyttes fra det initielle rom/temporale-domene til et abstrakt domene som passer bedre for kompresjon [Fluckiger-95].



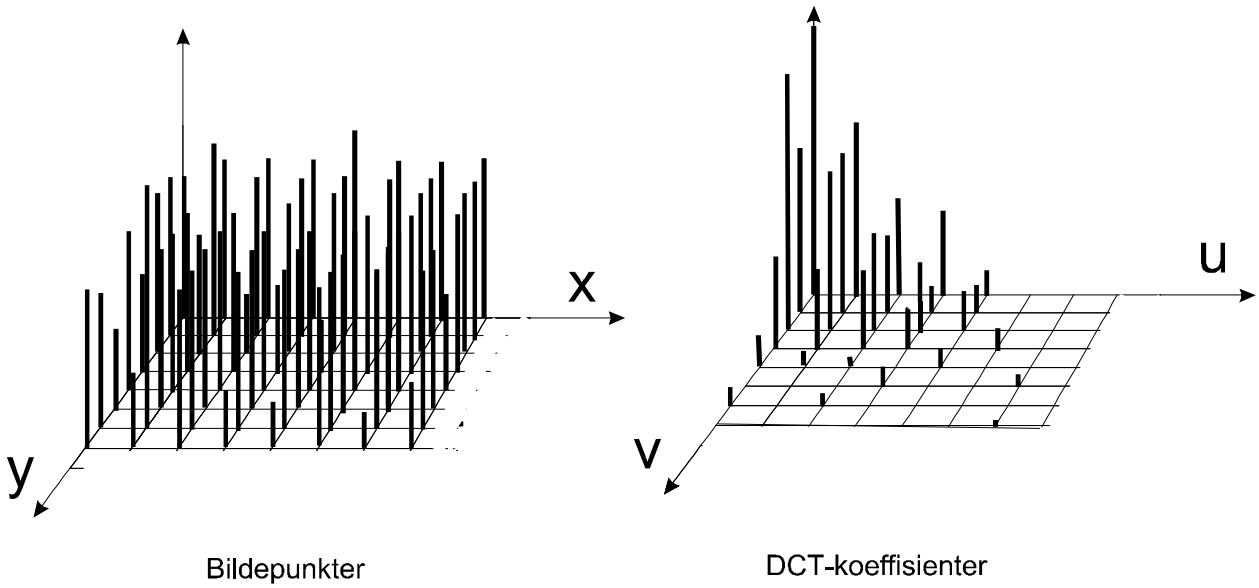
Figur 3.4 Prinsipp for transformasjonskoding [Fluckiger-95].

3.3.4.1 Diskret cosinus transformasjon [Fluckiger-95]

Komprimering ved hjelp av diskret cosinus transformasjon (DCT) består av følgende tre trinn:

- Selve transformasjonen
- Kvantisering
- Koding

Først deles bildet opp i blokker av 8x8 punkter. Hver blokk transformeres fra romdomene til et frekvensdomene.



Figur 3.5 Transformasjon av bildepunkter til DCT-koeffisienter.

Vi kan si at transformasjon tar punktverdier og overfører disse til frekvenskomponenter. Komponenten for ingen romfrekvens kalles DC-koeffisienten og er snittverdien av alle punktene i blokken. De resterende 63 verdier er AC-koeffisitenter og representerer amplituden og økende horisontale og vertikale frekvenser i blokken. Siden nabopunktene i et bilde ofte likner på hverandre eller varierer svakt vil mesteparten av signalenergien tvinges inn i lavere

frekvens-koeffisienter. Høyere frekvenskoeffisienter får derfor verdier lik null eller i nærheten av null.

Transformasjonen skjer med følgende formel [MPEG-92]:

$$F(u,v) = \tfrac{1}{4} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos(\pi(2x+1)u/16)\cos(\pi(2y+1)v/16)$$

der : x,y,u,v = 0,1,2 .... ,7
C(u) og C(v) = $1/\sqrt{2}$ for u,v=0
og 1 ellers.

Den inverse transformasjonen skjer ved følgende formel [Gonzalez-92]:

$$f(x,y) = \sum_{u=0}^{7} \sum_{v=0}^{7} \tfrac{1}{4} C(u)C(v) F(u,v) \cos(\pi(2x+1)u/16)\cos(\pi(2y+1)v/16)$$

der : x,y,u,v = 0,1,2 .... ,7
C(u) og C(v) = $1/\sqrt{2}$ for u,v=0
og 1 ellers.

Transformasjonen i seg selv gir ingen komprimering av dataene, men gir større mulighet for videre komprimering. Ofte benyttes kvantisering etter transformasjon for å oppnå komprimering.
Man benytter da en ferdig kvantiseringstabell på 8x8 verdier.

1	1	1	1	1	4	8	16
1	1	1	1	4	4	8	16
1	1	1	2	4	4	8	16
1	8	8	8	8	16	16	16
2	8	8	8	8	16	16	32
4	8	8	8	16	16	16	32
4	8	8	8	16	16	32	32
8	8	8	16	16	32	32	64

Figur 3.6 Eksempel på en kvantiseringstabell.

Hver verdi fra DCT deles med tilsvarende verdi i kvanitiseringstabellen. Resultatet av deling rundes av, og beholdes hvis den nye verdien er større en 1, hvis ikke settes den til null. Verdiene i tabellen er lavest i nærheten av DC-koefisienten og høyre jo lenger fra DC-koefisienten du kommer. Det gjør at koefisienter for høyre frekvenser oftest blir lik null etter kvantisering. Ved så å sikk-sakk skanne de nye verdiene fra øverste venstre hjørne til nederste høyre hjørne, vil de verdier som mest sannsynlig er lik null ligge sist i skanningen. Dette gjør at det lettere å komprimere dataene med f.eks. løpelengde koding.

3.3.4.2 Wavelet transformasjon (Hentet fra [Hilton-94])

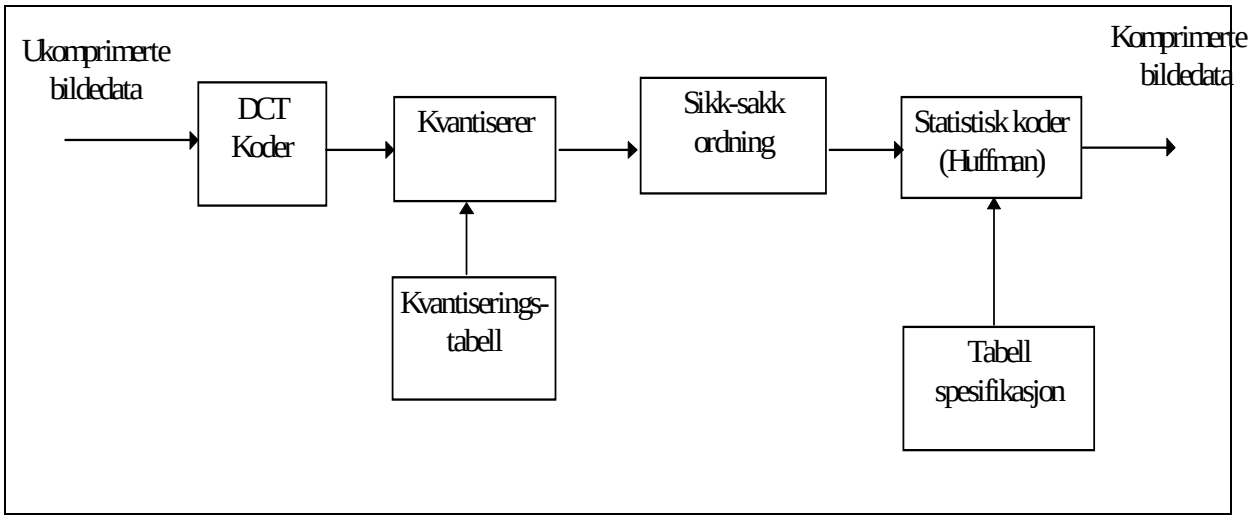
I de senere årene er wavlet transformasjon blitt utforsket for bildekomprimering. Det er en teknikk som ligner mye på diskret cosinus transformasjon. Man gjør her en lignende transformasjon av bildet til et frekvensdomene, deretter deles bildet opp i flere delbilder hvor hvert delbilde representer et spesielt frekvensbånd for bildet. Jo flere ganger du deler opp bildet jo smalere frekvensbånd representerer hvert bilde. Denne oppdelingen gir ingen kompresjon av bildet. Den konsentrerer imidlertid bildets punktverdier til noen få koeffisienter. Med denne koeffisient konsentrasjonen kan man kvantisere disse på lignende måte som ved DCT.
Det har vist seg at wavlet transformasjonen vil gi bedre resultater ved komprimeringsrater høyere en 30 til 1 i forhold til DCT (JPEG). I følge [Hilton-92] vil DCT (JPEG) være den beste metoden ved komprimeringsrater under 25 til 1. Det finnes ennå ingen komprimeringstandarder som benytter Wavelet-teknikken.

3.4 Bilde- og videokompresjonsstandarder

Her vil vi presentere de mest populære kompresjonsstandarder for bilde og video. En av de viktigste kriterier for om en kompresjonsstandard for video kan brukes er, i tillegg til at den gir en høy nok komprimering, hvor lang tid det tar å dekomprimere et bilde. Man må kunne dekomprimere bilder slik at man rekker å vise det antall bilder i sekundet som videoklippet tilsier. Vi vil først se på JPEG-standarden for bildekompresjon for deretter å se på MPEG-standarden som er den mest brukte standarden sammen med Apple QuickTime og AVI, for komprimering av video.

3.4.1 JPEG

JPEG er en kompresjonsstandard for fotografier [JPEG]. Standarden har fått navn etter komiteen som lagde standarden (Joint Photographic Expert Group). Standarden bygger på en serie kompresjons-teknikker. Der DCT er en av de viktigste faktorene.



Figur 3.7 Sekvensiell JPEG-koding.

Det finnes fire forskjellige typer av JPEG-komprimering [Koegel-94]:

- Sekvensiell koding
  Her skannes bilde en gang, fra øverste venstre hjørne til nederste høyre hjørne. Denne metoden er den mest vanlige.
- Progressiv koding
  Her skannes bilde flere ganger, først veldig grovt så finere og finere. Man oppnår da at bilde ved dekoding bygges opp først av grove brikker så finere og finere.
- Tapsfri koding
  Her benyttes ikke transformasjon, men en prediktor f.eks. DPCM og en statistisk koder (huffman).
- Hierarkisk koding
  Her kodes bildet med flere oppløsninger og man kan hente fram bildet i forskjellige oppløsninger hver for seg. Dette har ofte blitt brukt i forbindelse med internett der man
  først har bildet med dårlig oppløsning for så å gi bildet høyere og høyere oppløsning
  etterhvert. På den måten slipper man å vent lenge for å finne ut hva bildet er av, og kan
  gå videre hvis man ikke er interessert. Man kan også lage meget korte videosekvenser
  på denne måten.

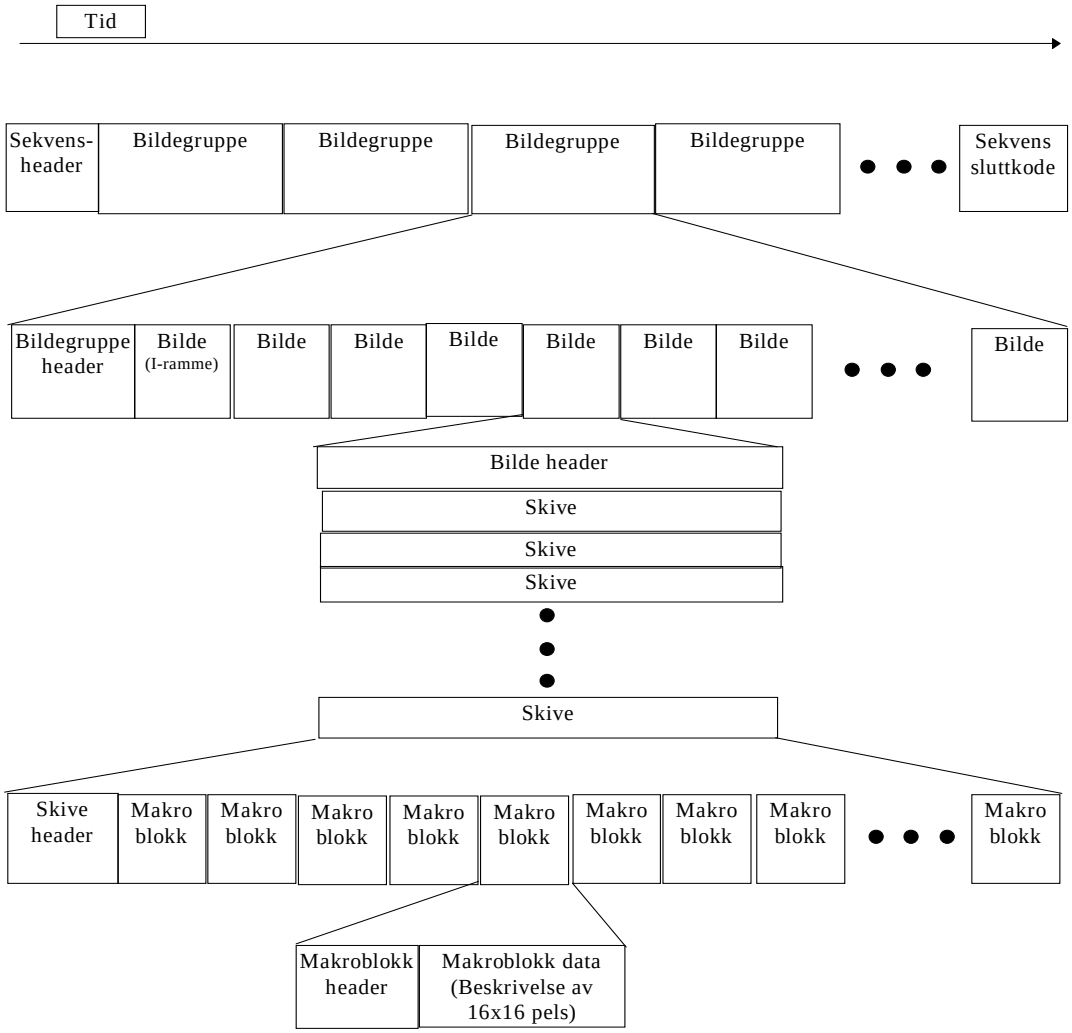
Alle unntatt tapsfrikoding benytter DCT som grunnlag for kodingen.

Ved å legge flere sekvensiell kodede JPEG-bilder etter hverandre kan man oppnå bevegelige bilder. Dette kalles M-JPEG (Motion-JPEG) og benyttes i noen digitale videosystemer. Det er ingen standard i seg selv, men benyttes i andre standarder for video.

3.4.2 MPEG-1 (video)

MPEG-1-standarden er den mest brukte for komprimering av videoklipp [MPEG-92]. Navnet sitt har den fått fra komiteen som bestemte standarden (Moving Picture Expert Group). Den var i starten ment som en standard for video lagret på CD-rom. Den ble senere endret til en standard for koding av audio/video med bitrater opptil 1,5 Mbit/s.

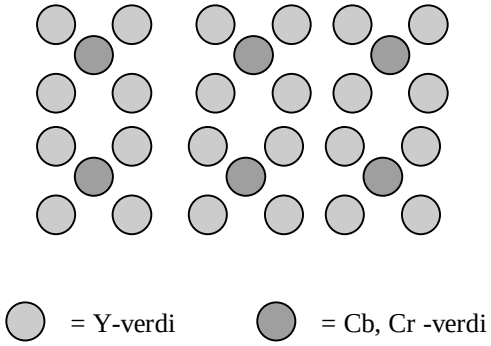
MPEG-1 benytter noe av den samme komprimeringsteknikken som blir benyttet i JPEG. Det er i tillegg tatt hensyn til temporal redundans mellom etterfølgende bilder. Dette gjør at et MPEG-1 komprimert videoklipp vil være ca 1/3 av størrelsen til en JPEG-komprimert ekvivalent.



Figur 3.8 Standard oppbygning av en MPEG-1 video strøm.

Figur 3.8 viser oppbygningen av MPEG-1-komprimert video. Den starter alltid med en sekvens-header der bla. opplysninger om bildestørrelse, rammerate og bitrate inngår. En sekvens avsluttes alltid av en sekvensslutt kode. Imellom disse ligger bildene i videoen. Disse er delt opp i bildegrupper som vanligvis inneholder et bestemt antall bilder. Antall bilder i en bildegruppe ligger vanligvis mellom 10 og 20 bilder. En bildegruppe kan være lukket eller åpen. Hvis den er lukket betyr det at alle bildene i bildegruppen kan dekodes uavhengig av bilder i andre bildegrupper.

Hvert bilde består av et antall skiver (se Figur 3.8). Størrelsen på skivene kan variere, men det er vanlig å la hver skive inneholde informasjon for minst en lengde av bildet. Hver skive består igjen av en eller flere makroblokker. Makroblokker er byggesteinene i en MPEG-1 video. Hver makroblokk gir en beskrivelse av et 16 x 16 punkts område i bilde. Hvordan denne beskrivelsen er fremlagt avhenger av hvilken type makroblokken er.

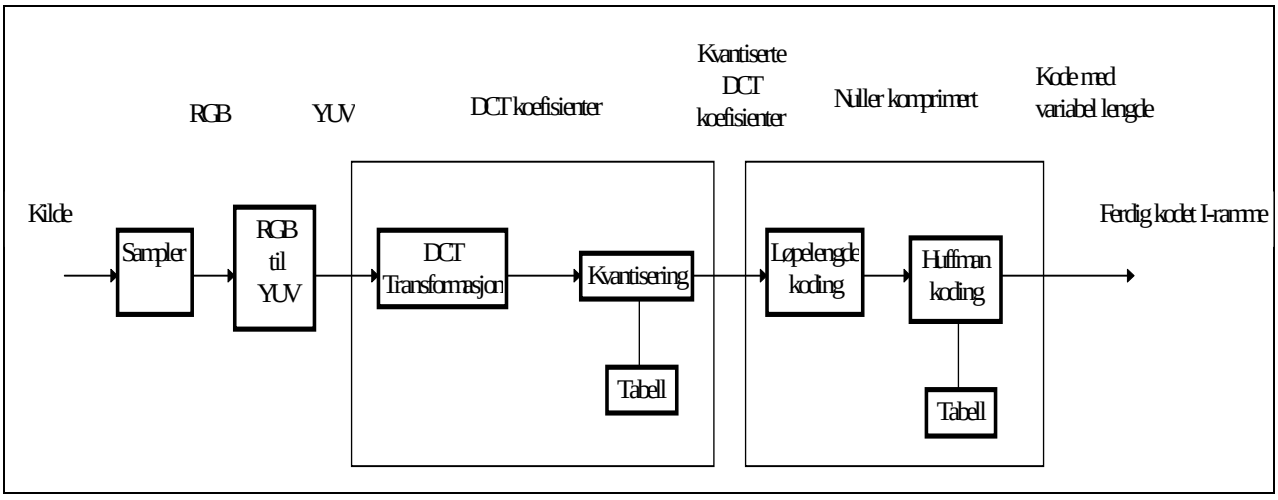


Figur 3.9 Relativ plassering av luminans og krominans i en makroblokk.

Dataene som skal gjenskapes av en makroblokk er 4, 8 x 8 punkts blokker med luminans og 2 ,8x8 punkts blokker med krominans. Grunnen til at det er færre blokker med krominans (farger) enn med luminans (lys/mørke), er at man utnytter unøyaktigheter i den menneskelige synssansen. Det er bevist at synsansen er flinkere til å skille mellom lys og mørke enn farger.

MPEG-1 video har fire forskjellige bildetyper. Disse kalles I-, P- ,B- og D-rammer.

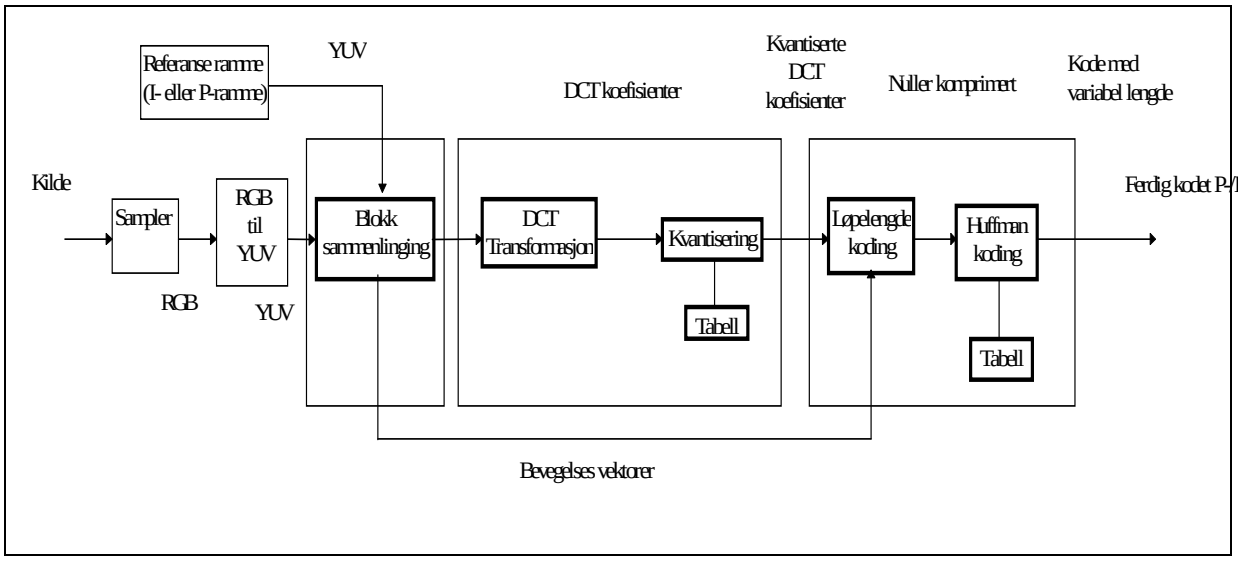
3.4.2.1 I-rammer



Figur 3.10 Koding av I-rammer.

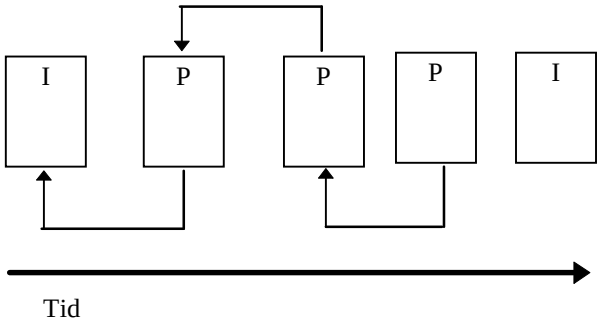
En I-ramme (Intra-ramme) er i utgangspunktet kodet likt som et JPEG-bilde. Det er komprimert uavhengig av andre bilder og kan derfor også dekodes selvstendig. Disse benyttes i MPEG-1 for å oppnå tilfeldige aksesspunkter i videoen. MPEG-1-standarden [MPEG-92] anbefaler ca 2 I-rammer per. sekund video. En I-ramme kan ha to forskjellige typer makroblokker. Forskjellen ligger i at den ene benytter en standard kvantiseringsskalering mens den andre benytter en egen 5 bits kvantiseringsskalering. Skaleringen kan ha verdiene 1-31. Begge typene benytter de samme metodene for koding som vist i Figur 3.7. Hver makroblokk består da av seks kodede 8x8 punkts blokker.

3.4.2.2 P-rammer



Figur 3.11 Koding av P- og B-rammer.

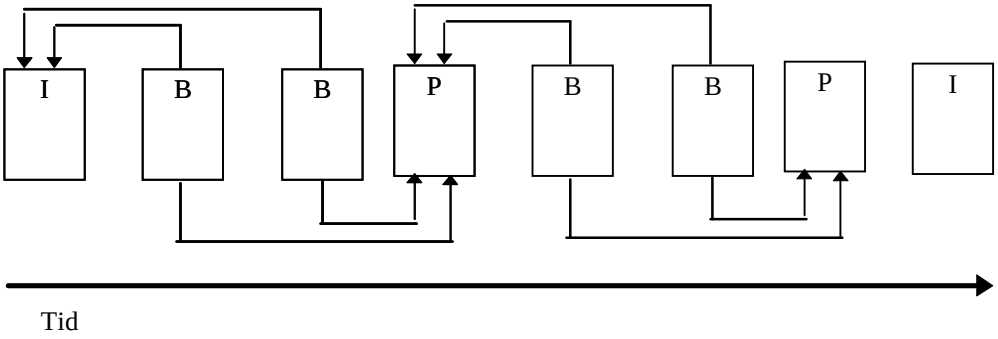
En P-ramme (Predektiv-ramme) er kodet i forhold til nærmeste I- eller P-ramme som vist i Figur 3.12. En P-ramme har mulighet for å bruke åtte forskjellige makroblokktyper. I fem av disse benyttes bevegelseskompensasjon. Forskjellene på de fem ligger i om kvantiseringsskalaen skal endres eller ikke, og om feilkorreksjonen til blokkene skal kodes. To av de er de samme makroblokktypene som benyttes i I-rammer. Den siste er av type hopp makroblokk. Hvilken makroblokk type som benyttes avhenger av datamengden hver makroblokk type representerer for den aktuelle bildebiten.



Figur 3.12 Kodet avhengighet mellom P- og I-rammer.

Bevegelses kompensasjon gjøres ved at man ved koding av P-rammer sammenligner makroblokken som skal kodes med makroblokker i foregående I-/P-ramme. Hvis man finner en som er lik, lagres det en vektor som representerer avstanden mellom makroblokkene i høyde og bredde. Når blokken ikke er helt like lagres det i tillegg feilkoeffisienter som retter opp forskjellene på makroblokkene. Disse feilkoeffisienten kan kodes ved hjelp av DCT hvis nødvendig. Hvis makroblokkene er like og de ligger på samme sted på begge bilder vil det i stedet bli lagret en hopp makroblokk, som brukes når bildene som sammenlignes er like i det område makroblokken representerer. Hvis man ikke finner en makroblokk som er lik eller ligner, vil blokken kodes på samme måte som en av makroblokkene i en I-ramme.

3.4.2.3 B-rammer



Figur 3.13 Kodet avhengighet mellom B-rammer og I-/P-rammer.

En B-ramme (Bi-direksjonal-ramme). Benytter seg av bevegelseskompensasjon på samme måte som en P-ramme, men det lagres ingen feilkoeffisienter. Den kodes i forhold til I-rammer og P-rammer. Makroblokkene kodes mot nærmeste I-/P-ramme bakover i tid eller fremover i tid. B-rammer har mulighet for tolv forskjellige makroblokker. De typene som er i tillegg til de som benyttes i P-rammer er de som benytter bakover bevegelsesvektor. Det vil si de har mulighet for å sammenligne blokkverdier med makroblokker fra et fremtidig bilde. Det gir mulighet for tre forskjellige prinsipper for bevegelses kompensasjon ved hjelp av bevegelsesvektorer. Den ene benytter bevegelse kompensasjon i forhold til et tidligere bilde, den andre benytter bevegelseskompensasjon i forhold til et fremtidig bilde og den tredje benytter bevegelseskompensasjon der bevegelses vektorer peker til en makroblokk i et tidligere bilde og en annen vektor peker til en makroblokk i et fremtidig bilde og snittet av de to makroblokkene blir brukt.

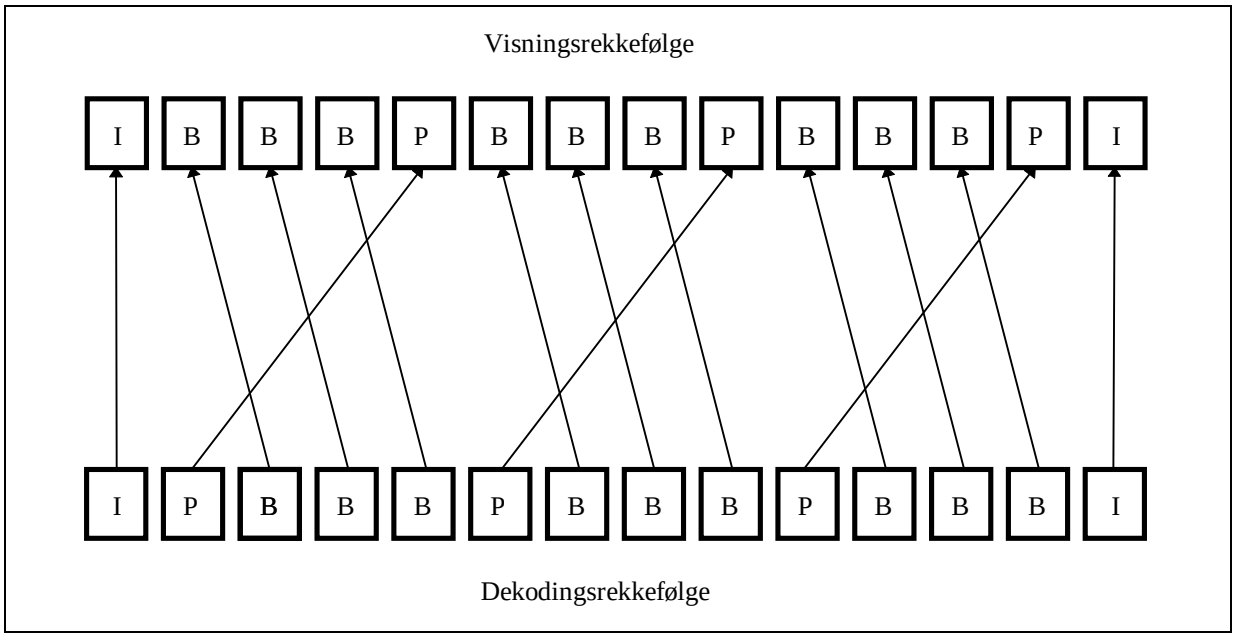
3.4.2.4 D-rammer

De tre overstående rammetypene er de som er mest vanlig å bruke i MPEG-kodede videoklipp. I tillegg finnes det en fjerde type. Denne kalles D-ramme (DC-ramme). D-rammer kodes på samme måte som en I-ramme, men ved DCT lagres kun DC-verdien fra transformasjonen. Dette gjør at vi får et bilde der hver blokk har kun en farge. Dette gir et grover bilde enn det som er ønskelig ved vanlig avspilling, men kan benyttes ved hurtigspoling. D-rammer kan ikke benyttes sammen med de tre andre typene og man må derfor ha en egen spolefil for bruk av D-rammer.

For mer informasjon om dekoding av MPEG-1 video se vedlegg 1.

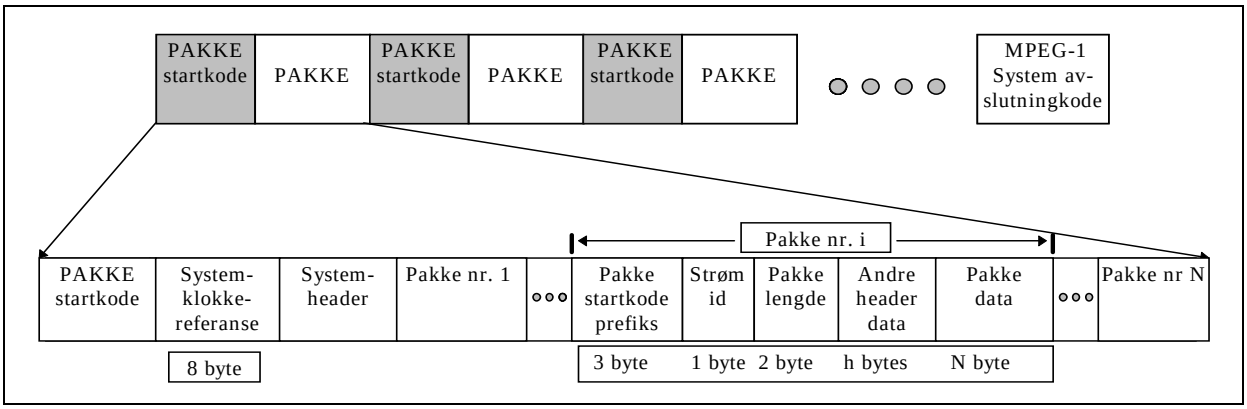
3.4.2.5 Bilderekkefølge og visningsrekkefølge

For å gjøre dekoding av MPEG-1 video enklere har man valgt å endre på bilderekkefølgen ved koding i forhold til den normale visningsrekkefølgen. Det er gjort fordi B-rammer kan kodes mot en etterfølgende ramme i visningsrekkefølgen. Det som gjøres er at den rammen som B-rammen er kodet mot, og som skal vises etter at B-rammen er vist, flyttes frem foran denne B-rammen. For at dekoderen skal klare å vise rammene i riktig rekkefølge har hver ramme i en bildegruppe et nummer, kalt temporal referanse, som forteller hvilket nummer i visningsrekkefølgen rammen har. Dette er vist i Figur 3.14



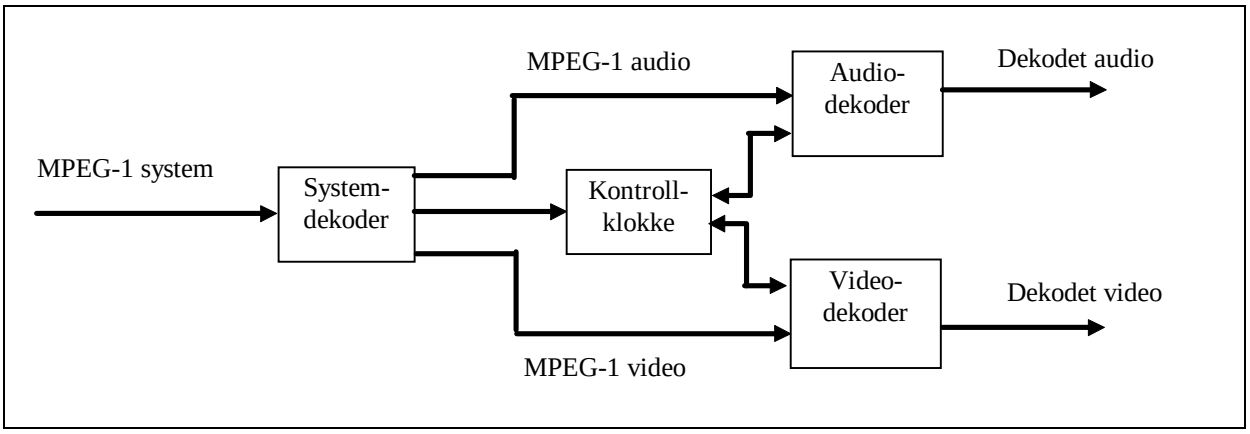
Figur 3.14 Overgang fra dekoding til visning.

3.4.2.6 MPEG-1 Systemlag



Figur 3.15 MPEG-1 systemstrøm.

Et MPEG-1 systemlag inneholder både MPEG-1 audio- og videodata. Dataene er splittet opp i store pakker og små pakker der en stor pakke inneholder flere små pakker (se Figur 3.15), mens hver liten pakke inneholder enten audio- eller videodata. Det finnes også muligheter for flere kanaler slik at man kan pakke flere forskjellige videoklipp i samme systemfilen. Dette er imidlertid lite brukt. Systemlagets viktigste funksjon er å synkronisere audio- og videodataene slik at de dekodes og vises til riktig tid. Til dette brukes en presentasjontidskode som er lagret i systemlaget.
Ved dekoding fjernes all informasjon i systemlaget før audio- og videodataene sendes til hver sin dekoder. Presentasjontidskoden sendes til en kontrollklokke som synkroniserer dekodingen.



Figur 3.16 Dekoder for MPEG-1 system

3.4.3 MPEG-2 [Liu-96]

MPEG-2 [MPEG-95] er en videreutvikling av MPEG-1. Den er utviklet for bruk ved mye høyere bitrater enn MPEG-1, dvs. bitrater i området mellom 2 og 15 Mbit/s. De største forskjellene fra MPEG-1 er at den støtter interlaced video og omskalering av bilder. En MPEG-2 dekoder kan dekode MPEG-1, men ikke vice versa.

3.4.4 H.261/H.263-standardene [H.261]

H.261 er standarden som er mest brukt til videokonferanser. Den er brukes i hastighets området 64 kbit/s til 1,920 kbit/s. Man har to typer rammer, en intrakodet rammetype og en interkodet rammetype, der man for den interkodede typen benytter seg av bevegelseskompensasjon med feilkoeffisienter. Standarden er ment brukt i forbindelse med sanntidskommunikasjon og krever derfor at både koding og dekoding kan foregå i sanntid.

H263 er en forbedret versjon av H261 og er ment å være arvtakeren etter H261. H263 er laget for båndbredder rundt 22 kbit/s [Tekalp-95].

3.4.5 AVI-standarden [Infoworld-96]

Avi står for Audio Video Interleave. Det er ikke en egen kodestandard, men et standard filformat for lagring av audio/video på PC. Avi-filer har en header der komprimeringsmetoden for lyd og video er angitt. For avspilling kan f.eks. MediaPlayer under Windows benyttes. Standarden er skapt av Microsoft og benytter seg av bla. Motion-JPEG for komprimering av video.

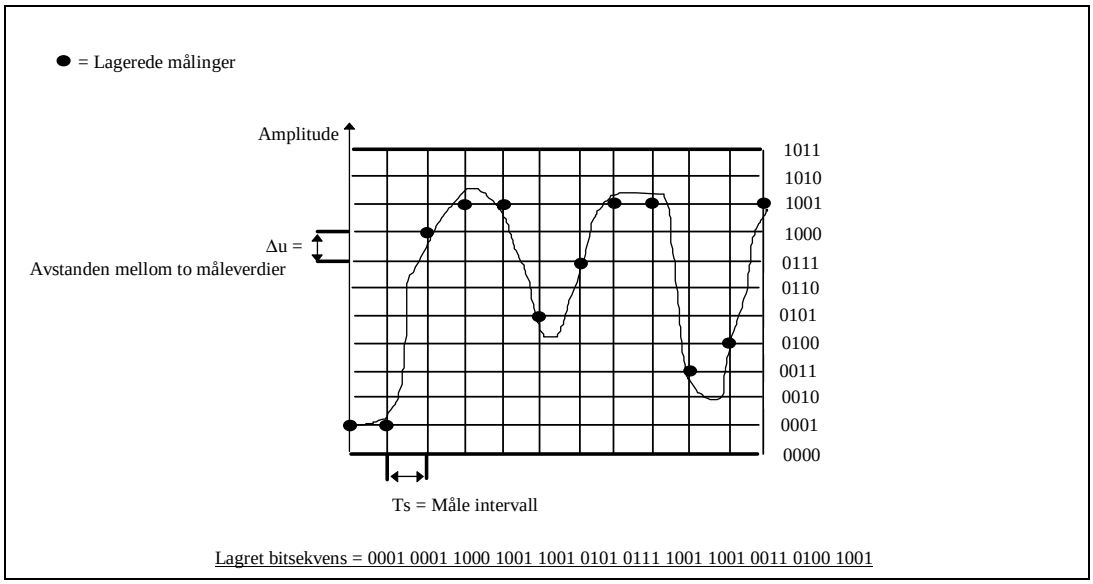
3.4.6 QuickTime [Apple-93]

QuickTime var opprinnelig en utvidelse av systemprogrammene til Macintosh, som muliggjør integrasjon av tidsbasert data i Macintosh applikasjoner. Eksempler på tids basert data er lyd, video og animasjon. For videokomprimering benyttes en egenutviklet komprimeringsmetode som er laget spesielt for hurtig dekoding. Komprimeringsfaktoren er fra 5 ganger til 25 ganger. Det benyttes både rom og temporal kompresjon. I de senere versjoner har man i tillegg til den egenutviklete metoden tatt i bruk MPEG-1 for video [Apple-96]. De senere versjoner av QuickTime er også tilgjengelig i programmer for Windows og UNIX.

3.4.7 ActiveX [activeX-96]

ActiveX er et sett teknologier som muliggjør interaktivt innehold for WWW. ActiveX innholder forskjellige så kalte kontroller som gir mulighet for interaktive objekter. Innenfor dette finnes det også muligheter for forskjellige videokomprimeringsfiltre. Disse brukes ved NORUT i en pakke som de kaller ActiveMovie. ActiveMovie har ingen egen komprimeringsstandard, men har filtre som inneholder flere av typene nevnt over.

3.5 Digital lyd



Figur 3.17 Forenklet eksempel på digitalisering av lyd.

For å digitalisere lyd må man på samme måte som med bilde dele lyden opp i biter. Størrelsen på disse bitene avgjør lydens kvalitet. Dette gjøres ved å ta sampler av den analoge lyden. Man måler lydens verdi ved bestemte tidsintervaller og lagrer disse målingene i målt rekkefølge. Nøyaktigheten på disse målingene (antall bit som benyttes) bestemmer av hvor nær måleverdien er den virkelige verdien til lyden som måles. Antall målinger som blir gjort i sekundet avgjør hvilke frekvenser som kan representeres i den digitale lyden.

Når man snakker om digitalisert lyd er CD-standarden den mest vanlige å ta utgangspunkt i [CD-94]. Den benytter seg av en målefrekvens på 44.1 kHz som tilsvarer et frekvensområde fra 0-22.5 kHz. Hvert sampel er på 16 bit som betyr at ca 65.000 ulike verdier kan representeres. Hvis vi ser på datamengden vil dette tilsi 176,4 kByte/s for tokanals (stereo) lyd i CD-kvalitet. For en 90 min langfilm vil dette tilsvare 952,6 MByte.

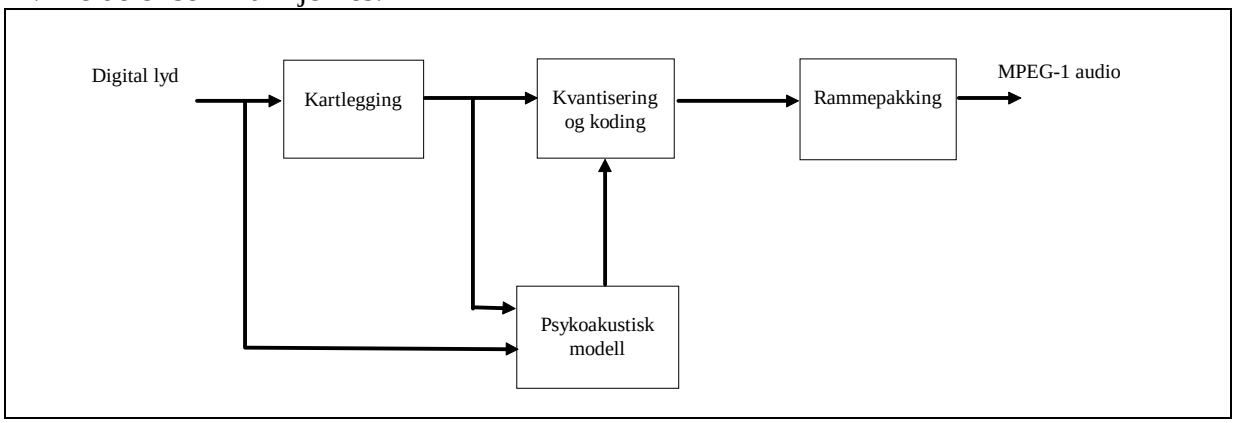
3.6 Komprimering av digital lyd

Selv om lydinformasjon ikke tar like stor plass som bildeinformasjon, er det likevel store datamengder det er snakk om for en langfilm som vist over. Hvis man i tillegg ønsker en

løsning med flere kanaler, som de nyeste standardene for flerkanals lyd der man typisk har mellom 4 og 6 kanalers lyd vil datamengden øke tilsvarende [MPEG-95].

3.6.1 MPEG-1 audio [MPEG-92]

MPEG-1 audio er den første internasjonale standard for komprimering av lyd. For komprimering utnytter MPEG-1 audio de perseptuelle begrensinger som finnes i de menneskelige høre-mekansmer. Man fjerner informasjon i lyden som man ikke ville oppfattet allikevel. For å få til dette brukes en psykoakustisk modell. Den analyserer lyden og finner ut hvilke deler som kan fjernes.



Figur 3.18 Koding av MPEG-1 audio [Pan-95].

MPEG-1 audio består av tre lag. Disse kalles MPEG-audio lag 1, lag 2 og lag 3. Lagets komprimeringskompleksitet, øker med nummeret. Dvs. lag 3 bruker den mest kompliserte komprimerings allgoritmen. Lagene følger den samme hierarkiske oppbygning. De er kompatible på den måten at en dekoder laget for lag 3 kan dekode lyd komprimert i lag 1 og 2.

Kompresjonsmetodene brukt for alle tre lag følger de samme prinsippene (se Figur 3.18): :

- Frekvensområdet deles opp i 32 subbånd.
- En fourier transformasjon benyttes for å representere signalet i frekvens domenet.
- En psykoakustisk modell benyttes på det transformerte signalet for å beregne det minste hørbare støynivået.
- Kvantisering utføres, hvor det bestemmes hvor mange bit som skal benyttes for å kode hvert enkelt sampel.
- Den kodede lyden blir delt inn i lyd rammer med informasjonkodingen.

En psykoakustisk modell fjerner lydinformasjon som man mener øret allikevel ikke vil oppfatte.

3.6.1.1 Lag I

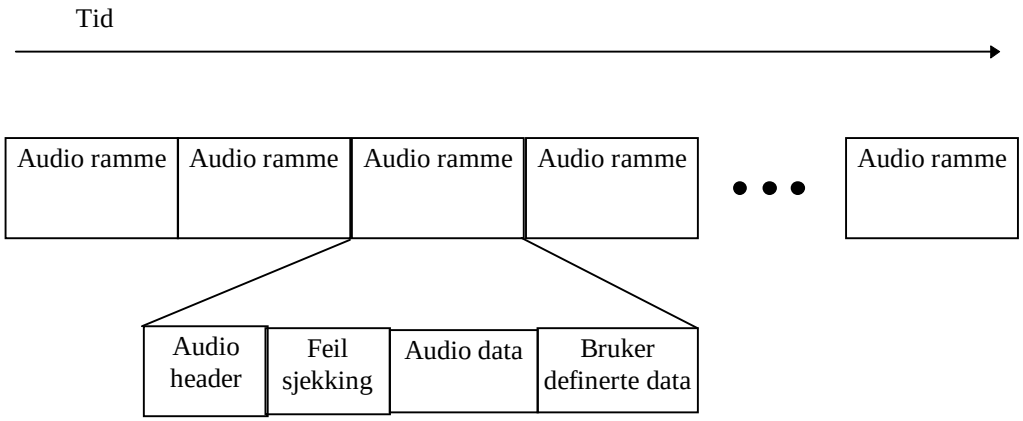
MPEG-Audio lag 1 er en enklere versjon av MUSICAM. Den gir mulighet for bruk av enkle kodere og dekodere. Den gir en middels lydkvalitet ved 192 eller 256 kbit/s. Hver ramme i lag 1 kan dekodes uavhengig av andre rammer. En ramme i består av 384 sampler. Lag 1 brukes i Philips DCC (Digital Compact Cassette). Hastigheten som brukes her er 128 kbit/s per kanal.

3.6.1.2 Lag 2

MPEG-Audio lag 2 er identisk med MUSICAM standarden. Den har samme kvalitet som en CD-plate. Bitraten ligger på 96 eller 128 kbit/s. Dette er den mest brukte laget fordi det gir en god kompresjon og er mye enklere å dekode enn lag 3. Hver ramme i lag 2 kan også dekodes uavhengig av andre rammer. En ramme inneholder 1152 sampler.

3.6.1.3 Lag 3

MPEG-Audio lag 3 gir best resultat. Den er en kombinasjon av MUSICAM og ASPEC (lydkompresjons-teknikk laget i Erlangen i Tyskland). Den gir nær CD kvalitet ved 64 kbit/s pr. kanal. Rammer i lag 3 er avhengig av informasjon tidligere sendt i bitstrømmen. Hver ramme inneholder informasjon for 1152 sampler.



Figur 3.19 MPEG-1 audio oppbygning.

3.7 Nettverkstandarder for overføring av video over nett

For å kunne utnytte en videoserver må klientene være koblet til et nettverk som kan håndtere dataratene det er behov for å ha ved overføring av video. Det viktigste kriteriet ved video overføring er at man trenger en konstant garantert datarate for å kunne levere videokvalitet.

3.7.1 ISDN [ISDN]

ISDN står for Integrated Services Digital Network [ISDN], og er en tjeneste de fleste telefonselskaper tilbyr. ISDN gjør det mulig å overføre data med hastigheter opp til 128 kbit/s. Dette er 4 ganger høyere enn den hastigheten som tilbys av modem for overføring over en telefonlinje. Hastigheten på ISDN er imidlertid ikke nok for sanntidsoverføring av video med brukbar kvalitet. Med brukbar kvalitet menes en kvalitet som kan sammenlignes med VHS. Man kan imidlertid sende levende bilder over nettet hvis man kan akseptere at man kun får ca. 10 bilder per sekund i relativt dårlig kvalitet, f.eks ved bruk av H.261.

3.7.2 Dataoverføring over kabelnett ved hjelp av kabelmodem [zdnet-96]

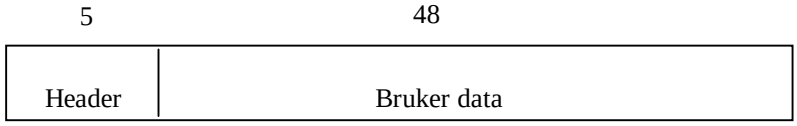
I den senere tiden har det vært mulig noen steder å sende og motta data fra internett over kabelnettet. Man benytter da et så kalt kabelmodem. Kabelmodemet har en Ethernet port som gjør det mulig å kommunisere med andre datamaskiner på kabelen eller via en hub til andre

nettverk. Hastigheten for henting av data for forskjellige kabelmodem varierer fra 500 kbit/s til 30 Mbit/s, mens sending av data varierer fra 96 kbit/s til 10 Mbit/s.

På grunn av at ethernet er brukt som protokoll er hastighetene beskrevet veldig avhengig av nettrafikken og det vil derfor aldri finnes en garanti for at du får den hastigheten du ønsker. Det er allikevel et skritt i riktig retning. Hvis kabeloperatører passer på at ikke for mange er koblet til den samme linjen kan man oppnå overføring av video med tilfredsstillende kvalitet på denne måten.

3.7.3 ATM (Asyncron Transfer Mode) [Tanenbaum-96]

Teknologien som i framtiden skal gi mulighet for å sende sanntids lyd og bildeinformasjon over store avstander er ATM. ATM benytter seg av små pakker kalt celler, med fast størrelse for overføring av data. Cellene er 53 bytes lange, der de 5 første bytene er en header og de resterende 48 bytene er brukerdata.



Figur 3.20 En ATM celle.

Fordelen med å bruke celle-svitsjing er at det gjør nettverket meget fleksibelt. Man kan både håndtere data som krever konstante datarater, f.eks lyd- og videoinformasjon og data som ikke krever konstante datarater. Celle-svitsjingen gir mulighet for meget høye overførings-hastigheter. Hastigheter oppe i Gigabit/s området er mulig.

Den tiltenkte hastighet for ATM-nett er 155 Mbit/s og 622 Mbit/s med mulighet for økning til Gigabit/s senere. Hastigheten 155 Mbit/s ble valgt fordi det er den hastigheten som trengs for overføring av HDTV.

ATM-nett er oppkoblingsorienterte det vil si at man først sender ut en forespørsel og får en tilbakemelding før data kan sendes.

En annen nyhet som ATM introduserer er muligheten for styring av overføringskvaliteten. Ikke bare sikring for at cellene kommer frem i riktig rekkefølge, men også mulighet for å sikre at cellene kommer frem til rett tid. Dette kalles Quality of Service. Her finnes det flere trafikktyper å velge imellom [ATM-96]:

- Konstant bitrate (CBR) kategorien garanterer en fast bitrate for overføringen. Dette betyr at man kan sende data med den angitte bitraten eller lavere. Denne kategorien egner seg godt til overføring av lyd og video.
- Sanntids variabel bitrate (rt-VBR) kategorien er ment for tidsfølsomme applikasjoner der man er avhengig av begrensede forsinkelser og forsinkelsesvariasjoner. Dette vil kunne pass for tale og videoapplikasjoner der overføringsraten varierer i tid.
- Ikke sanntids (nrt-VBR) kategorien er ment for applikasjoner som har variable overføringsrater.

- Tilgjengelig bitrate (ABR) kategorien er ment for applikasjoner som kan øke og senke overføringsrater etter som nettverket tillater det.
- Uspesifisert bitrate (UBR) kategorien er ment for ikke kritiske applikasjoner som tåler større forsinkelser i overføringen.

3.8 Andre løsninger

I dette kapitlet vil vi presentere noen av de videotjener-løsninger som eksisterer i dag. Noen er kommersielle, andre er laget i forbindelse med forskning på universiteter o.l.

3.8.1 Elvira (Eksperimentell videotjener for ATM) [Langørgen-94]

Dette er en videotjener utviklet av Stein Langørgen for TF (Televerkets Forskningsinstitutt) i sammarbeid med NTH (Norges Tekniske Høyskole) fra høsten 1994. Den leverer Motion-JPEG-video med VCR-funksjonalitet sammen med PCM-kodet audio. Audio-dataene blir lagret som rammer og varigheten av en audio-ramme tilsvarer varigheten av en video-ramme. Elvira har senere blitt utvidet til å kunne levere MPEG-1 systemstrøm uten VCR-funksjonalitet.

Levering av video skjer over et ATM-nettverk. TCP benyttes for kommando-overføring, MPEG-audio/video-overføring og UDP for audio/video-overføring for MJPEG.
Den benyttede bildehastigheten var 20 rammer pr. sekund og oppløsningen var 400x300 piksler per ramme for Motion-JPEG. Det ga en datastørrelse på ca. 2.0 Mbit/s.

Elvira benytter asynkron levering av video. Videoleveransen foregår ved at klienten sender forespørsler om bestemte rammer i videoen. Disse rammene blir så overført av videotjeneren. Hver forespørsel har en leveringstidsfrist som må overholdes for å oppnå kontinuitet i framvisningen av video hos klienten. Rammer blir buffret opp hos klienten for slik at videotjeneren kan levere rammer før den angitte tidfristen.

Elvira klarte å lever seks 2 Mbit/s videostrømmer per node og totalt 24 videostrømmer fra fire noder. Ved striping klarte man å levere totalt 16 videostrømmer fra de fire nodene. Man fant ut at det var henting av data fra disk som var den store flaskehalsen, og det foreslåes derfor å tilføre flere disker til hver node.

3.8.2 The Tiger Video Fileserver

Tiger er en videotjener utviklet av Microsoft Research. Tjeneren har noder som er vanlige PC-er med Windows NT som operativsystem og et ATM-nettverk for overføring av video-data. I følge [Tiger-96] er det en distribuert, feiltolerant sanntids filtjener, som kan levere datastrømmer med en konstant garantert hastighet til et stort antall klienter. Den er utviklet som en «video on demand» filtjener.

Den kan har mulighet for VCR-funksjonalitet. Klienten er ansvarlig for valg av rammer som skal brukes ved f.eks. spoling.

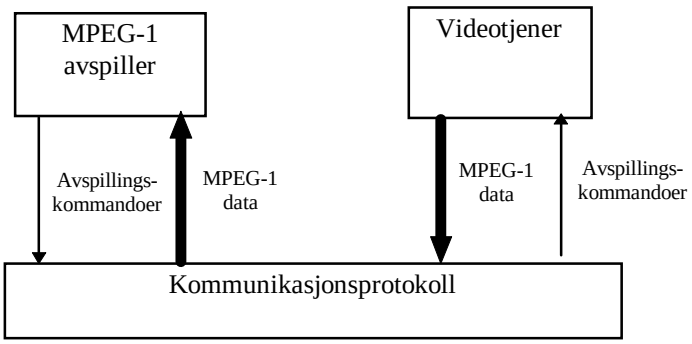
Man benytter UDP for overføring av videodata. Det kreves at alle strømmer fra en tjener har samme bitrate. Video som leveres er uavhengig av komprimeringsstandard. Den kan derfor levere MPEG-1, MPEG-2, komprimert AVI, ikke komprimert AVI eller andre formater.

Det benyttes striping for å utnytte diskene best mulig. For å hindre at flere klienter vil lese data fra samme disken samtidig er diskaksessen delt opp i slots. Maksimalt en klient kan lese data i hver slot. Det gjøres ved at man ved oppstart av en video venter til man finner en ledig slot. Man hevder å ha en teoretisk kapasitet på 60.000 videostrømmer ved bruk av 7.000 disker.

I et testoppsett har man bygget system med 5 PC-er med Pentium 133 MHz prosessorer, 48 MB RAM, PCI-buss, tre Seagate harddisker og et FORE OC-3 ATM-adapter. Ti 486/66 maskiner koblet til ATM-nettet fungerte som kilenter. Man oppnådde da en levering av 68 videostrømmer med bitrate på 6 Mbit/s. Nettverkskortene viste seg å være flaskehalsen i systemet tett fulgt av diskene som kunne levere 70 videostrømmer.

4. Kravspesifikasjon

Vi vil i det dette kapittelet beskrive hvilke krav som stilles til modulene som utgjør det totale systemet. Vi vil gå inn på hver enkelt del av systemet for å vise systemets muligheter og virkemåten som er tenkt for hver enkelt modul. For beskrivelsen tas det utgangspunkt i den nye Lava-avspilleren fra Norsk Regnesentral/NORUT og den nye kommunikasjonsprotokollen tilhørende Lava-prosjektet.



Figur 4.1 Overordnet oppbygning.

Systemet vil bestå av de komponenter som er vist i Figur 4.1. En MPEG-1 avspiller for mottak av og dekoding/visning av MPEG-1 kodet video, et kommunikasjonssystem for overføring av dataene over nettverk, og en videotjener for henting av data fra disk, manipulering av disse.

4.1 MPEG avspiller

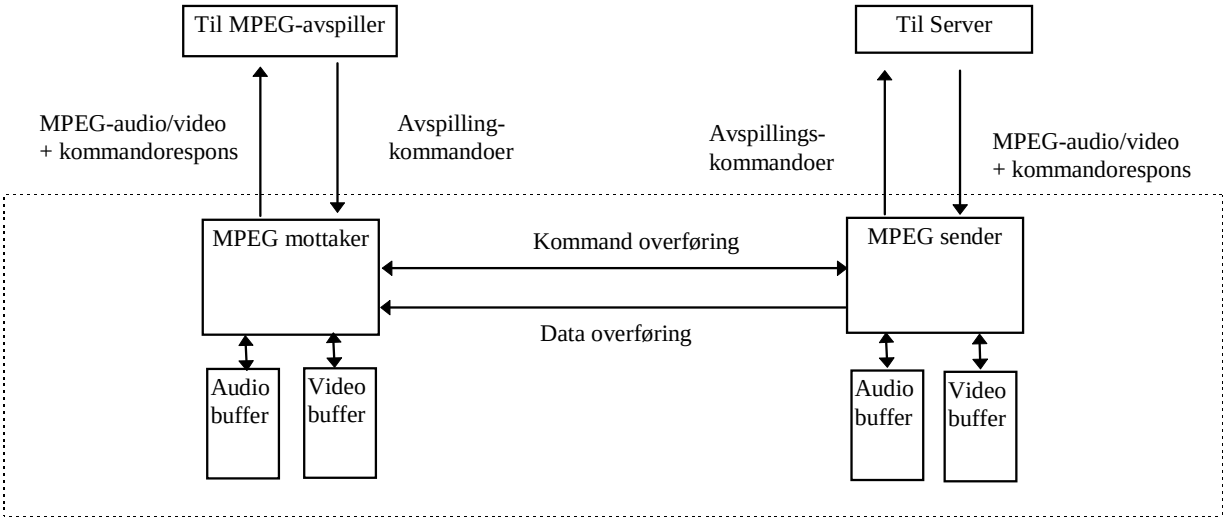
Følgende krav stilles til MPEG avspilleren :

1. Den skal kunne ta imot og spille av kodede MPEG-1 video- og audiostrømmer.
2. Den må ved avspilling kunne synkronisere en videostrøm og en audiostrøm ved hjelp av innebygde synkroniseringsverdier i strømmene.
3. Den må kunne skifte mellom avspilling av audio/video og kun video eller kun audio.
4. Avspilleren må kunne sende kommandoer som:
    - GetMove (Gjør server klar til sende film)
    - Play (Vanlig avspilling av film)
    - Ffwd (Spoling forrover, i et bestemt antall hastigheter gitt av kodingsformatet på video.)
    - Rewind (Spoling bakover)
    - Pause (Fryse bilde)
    - Step (Bilde for bilde)
    - Goto (Hopp til en bestemt ramme)

5. Avspilleren må kunne motta og tolke meldinger fra videoserveren. Dette kan være meldinger som :
    - Film not found
    - Audio not found
    - Video not found
    - End of film

Ved kommunikasjonsforbindelser med lav båndbredde kan det legges inn mulighet for å kutte ut videoen og kun levere audio. Dette må styres fra MPEG-avspilleren.

4.2 MPEG-1 overføringsprotokoll



Figur 4.2 Kommunikasjonssystemet.

Følgende krav stilles til overføringsprotokollen :

På sendersiden :

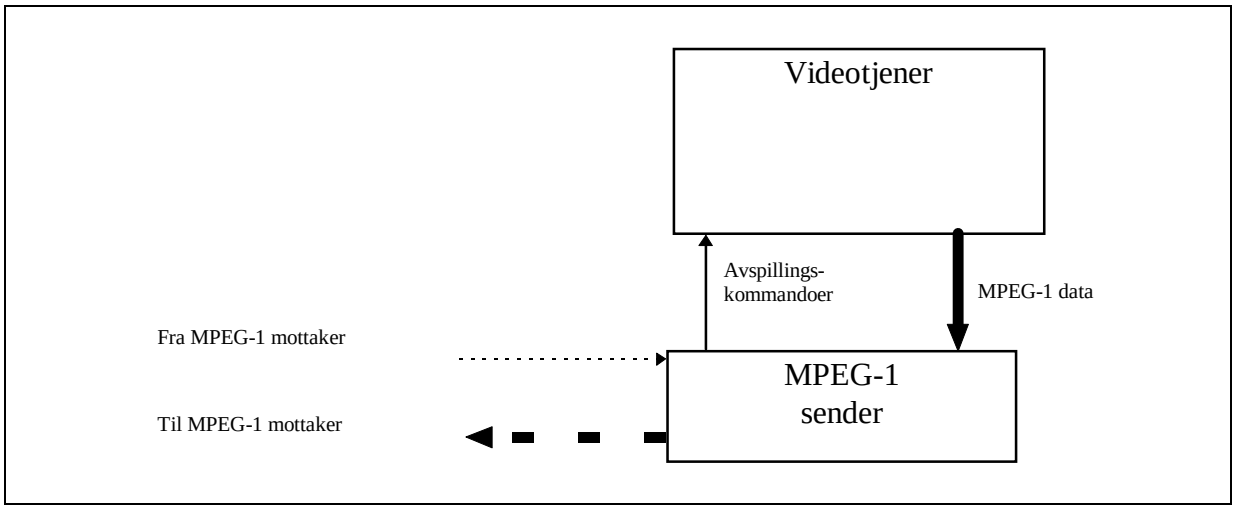
- Senderen skal kunne motta MPEG audio/video og sende dette videre til mottakeren.
- Senderen må kunne motta kommandoer fra MPEG-avspilleren (via mottakeren) og sende disse videre til serveren.
- Den skal kunne tømme audio/video buffrene sine ved hjelp av en kommando.
- Den skal kunne gi beskjed om hvor stor overføringskapasitet den aktuelle nettverksforbindelsen har.

På mottakersiden :

- Mottakeren skal kunne motta MPEG-audio/video og levere disse dataene videre til MPEG-avspilleren.

- Mottakeren må kunne motta kommandorespons fra serveren (via senderen) og sende disse videre til MPEG-avspilleren.
- Mottakeren må kunne buffre MPEG-1 audio/video. Dette for å kunne gi en så jevn nettverksbelastnig som mulig og for å jevne ut jitter i videostrømmen. Buffrene må kunne inneholde minst en bildegruppe.

4.3 Videotjener



Figur 4.3 Kommunikasjon med videotjener

Videotjeneren kommuniserer med MPEG-avspilleren ved hjelp av overføringsprotokollen beskrevet over. Den skal levere MPEG-audio/video i henhold til følgende krav :

- Videotjeneren skal motta kommandoer fra MPEG-avspillere.
- Den skal avgjøre om et ønske om avspilling av en gitt video skal godkjennes eller ikke. Det gjøres bla. ved hjelp av en katalog med oversikt over tilgjengelige filmer og tilgjengelige ressurser i videotjeneren. Hvis ønsket om video godkjennes skal videotjeneren garantere en feilfri leveranse.
- Ved leveranse av MPEG-audio/video skal videotjeneren levere dette i en så jevn strøm av video som gjennomførbart.
- Levering av audio sammen med video skjer kun ved avspilling i normal hastighet. I andre avspillingsmodus f.eks. spoling levers kun video.
- Ved ønske fra avspiller leveres kun audio. Bevegelse i denne audioen kan kun skje ved tilfeldig tilgang. Det finnes ingen spolingsmuligheter for audio.
- Videotjeneren skal kunne levere video i normal hastighet og et gitt sett av hastigheter forover og bakover.
- Den skal kunne «hoppe» til en gitt posisjon i filmen.

4.4 Katalogtjener

Katalogtjeneren er en del av videotjeneren. Den tar imot et ønske om å hente en film med en gitt filmidentifikator. Hvis filmen finnes returneres følgende data :

- Filens fulle navn
- Hvor filen er lagret.
- Minimum overføringskapasitet for nettverk
- Størrelse på bildegrupper
- Om bildegruppene er lukkede eller åpne.
- Hvilken MPEG-lyd komprimeringsteknikk som er benyttet. (Lag I, II eller III).
- Filmens lengde i antall rammer.

Hvis filmen ikke blir funnet, returnerer tjeneren en feilmelding. Denne delen av videotjeneren er ferdig laget i forbindelse med Elvira-prosjektet [Langørgen-94].

4.4.1 VCR-funksjonalitet

For spoling i MPEG-filen vil det gis muligheter for ulike spolingshastigheter. De mest aktuelle spolingshastigheter er fremover 1/5x, 1/2x ,3x, 10-15x og bakover 1x, 3x, 10-15x. Spolingshastighetene vil variere litt utifra oppbygningen av MPEG-filen (avstanden mellom I-rammer og P-rammer). Dvs. ber man om en spolingshastighet på f.eks. 3x kan hastigheten variere fra 2x til 5x alt etter kodingen av den aktuelle filmen.

MPEG-avspilleren må kunne lese MPEG-1 audio/video «ramme» for «ramme». Dette er egentlig ikke mulig med MPEG-kodet video. Muligheten ligger i å kunne sende hver I-ramme for seg, og deretter sende neste P-ramme og etterfølgende B-rammer så sendes neste P-ramme og etterfølgende B-rammer osv. Dette vil gi dekoderen mulighet til å dekode det den til en hver tid mottar.

5. Mulige løsninger

Med VCR-funksjonalitet menes de funksjoner som er mest vanlig på en vanlig VHS-videoavspiller. Her vil vi derfor beskrive hvordan disse funksjonene kan oppnåes på MPEG-1 kodet video. Det blir først gitt en generell beskrivelse av hvordan man oppnår dette på digital video. Deretter viser vi hvorfor dette ikke fungerer for MPEG-1-komprimert video. Til slutt presenterer vi metoder som kan benyttes for MPEG-1. En del av løsningene som blir presentert her er hentet fra [Koteng-96].

5.1 Innledning

Mulighet for spoling i et lagret videoklipp er blitt et naturlig ønske for de fleste som har brukt en videospiller. Det gir mulighet til å se ting på nytt, bevege seg forbi uinteressante sekvenser eller se igjennom et klipp på en hurtigere måte. Det er også ønskelig å kunne bevege seg saktere gjennom videoklippet, såkalt «slow motion» for å få med seg detaljer i videoen på en bedre måte.

For å oppnå disse funksjonene i et digitalt lagret videoklipp kan man ved spoling øke antall bilder som blir vist til det dobbelte ved 2 ganger spoling og det 10 dobbelte ved 10 ganger spoling. Dette er samme metode som benyttes for en vanlig videospiller. Ved «slow motion» vises et mindre antall bilder per sekund enn det som vises ved vanlig avspilling. Ved spoling med høy hastighet vil dette bety en stor økning i datamengde som skal hentes fra disk. Hvis det i tillegg skal leveres over et nettverk vil dette kreve stor kapasitet både hos leverandøren av videoklippet og av avspilleren som skal vise dette på skjermen. Hvis den digital lagrede videoen er komprimert for å ta mindre plass på lagringsmediet, vil det medfører at avspilleren i tillegg til å vise bildene må dekode bildene, noe som kan være meget resurskrevende. Det er derfor vanlig å redusere antall bilder som vises per sekund ved spoling, til det samme antall per sekund som vises ved vanlig avspilling. Man hopper da over et antall bilder for hvert bilde som vises. For 10 ganger spoling vil man da kun vise hvert 10. bilde i videoklippet.

5.2 Leveranse over nettverk

Siden ATM-nett er det eneste reelle alternativet som er funnet for sanntidsleveranse av video over lengre strekninger vil vi her ta utgangspunkt i muligheter som eksisterer i ATM-standarden. Man vil i et ATM-basert system kunne bestille en hvis fast overføringskapasitet (Quality Of Service) og denne bør ikke overstiges. Når vurdering av de forskjellige metodene for spoling skal gjøres vil det måtte legges mest vekt på er at man ikke øker nettverksbelastningen ved spoling og «slow motion». Dette vil også være fordelaktig ved henting av data fra disk, noe som også vil være en potensiell flaskehals i et slikt system.

5.3 Hva er problemet?

Siden vi ved spoling velger å ikke spille lyd vil vi kun diskutere spoling med hensyn på MPEG- video. Hva gjør så spoling i MPEG-video så vanskelig? Kan man ikke bare vise annenhvert bilde og oppnå 2 ganger spoling eller hvert 10. bilde og oppnå 10 ganger spoling? Dessverre, på grunn av MPEG-standardens viktigste komprimeringsmetode kan man ikke det.

I MPEG-video er enkelte biler kodet for seg (I-rammer), mens andre bilder er avhengig av at andre bilder er dekodet og tilgjengelige (P-rammer og B-rammer). Disse avhengighetene gjør det vanskelig å få til spoling i MPEG-video. I tillegg har man størrelsen på hver ramme å ta hensyn til. Det er mulig å kode en MPEG-video med kun I-rammer. Da vil man kunne bevege seg fritt i videoen på samme måte som i en ikke kodet digital video uten å tenke på avhengigheter mellom bilder. Dette ville dessverre øke bitraten på videoen med ca. 3 ganger, noe som absolutt ikke er ønskelig. Spesielt er den lave bitraten viktig når kapasiteter i nettverk og båndbredde fra harddisk er de viktigste ressursene for en videotjener.

Et av målene for denne oppgaven er å kunne levere data til MPEG-avspillere i henhold til MPEG-1-standarden. Dette for at det skal kunne benyttes en helt standard MPEG-1-dekoder/avspiller for å oppnå denne VCR-funksjonaliteten.

5.4 Hvordan oppnå spoling i digital video

Her presenteres to metoder for bevegelse i digitalt lagret video uten avhengighet mellom rammer. Vi vil også se på hvordan man kan lage en oversikt over den digitale videoen slik at man lett finner fram til ønskede posisjoner i videoen.

5.4.1 Indeksfil

I en digital video oppnår man ofte spoling ved å hoppe over rammer. Hvis dette skal gjøres er det nødvendig at man vet hvor hver ramme starter og hvilket rammenummer den aktuelle rammen har. Dette kan man oppnå ved å lage en indeksfil. Indeksfilen må inneholde informasjon om både lyd og bilde. Indeksfilen genereres ved å parse den digitale videoen. Parseren har som input den digitale videofilen og finner eventuelle startkoder for hver enkelt ramme. Posisjon for denne rammen og hvilket rammenummer rammen har blir lagret. I tillegg kan lyden deles opp på tilsvarende måte slik at en bilderamme tilsvarer en lydramme. Dette skrives ut i en indeksfil som lagres sammen med den digitale videofilen. Ved å benytte indeksfilen kan man nå hente ut hvilken som helst ramme i videofilen.

5.4.2 Spolefil

Hvis man ønsker en bestemt spolehastighet kan man lage egne filer for dette. Man benytter også da et slags parse-program der man tar inn videofilen og plukker f.eks. ut hver 10. ramme for 10 ganger spoling, for så å legge disse rammene ut i en egen fil. Hvis man skal kunne bruke dette må man ha en overgangstabell mellom avspillingsfilen og spolefilen. Denne tabellen forteller hvor i spolefilen man skal hoppe ved spoling fra et tilfeldig sted i avspillingsfilen og hvor i avspillingsfilen man skal hoppe fra et tilfeldig sted i spolefilen.

5.5 VCR-funksjonalitet i MPEG-video

For spoling i MPEG-video kan metodene over benyttes med visse modifikasjoner. Vi vil her presentere alternativene som er funnet for bevegelse i MPEG-video. Disse deles opp i grupper utfra hvilke hastigheter de benyttes i. For all bevegelse utenom avspilling kreves en indeksfil for å fortelle hvilke rammer som ligger hvor i filen. Det forutsettes her at en slik fil finnes tilgjengelig.

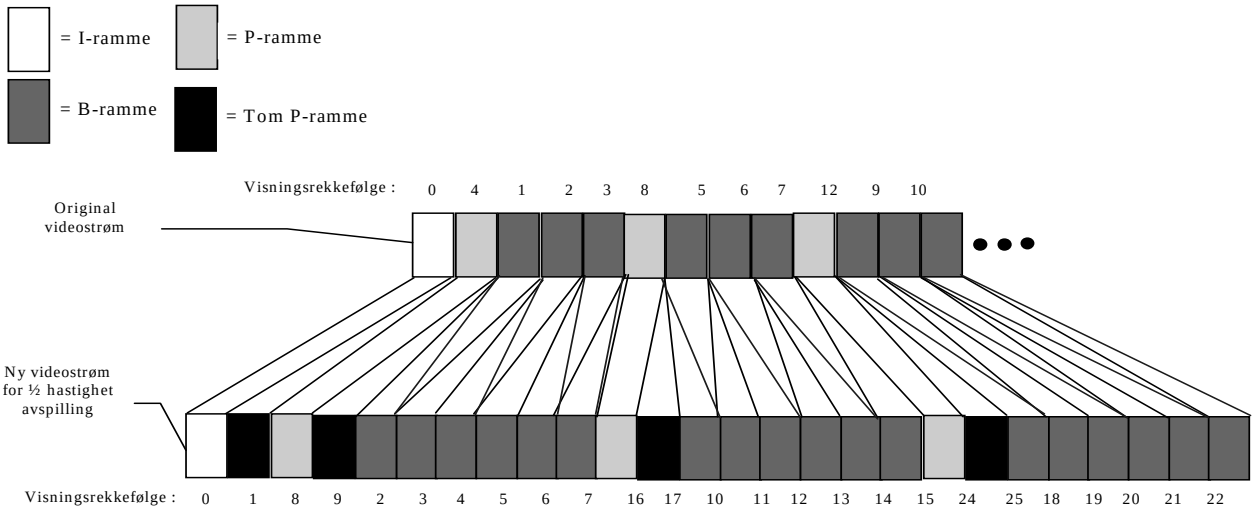
Siden antall I-, P- og B-rammer i en MPEG-kodet fil er noe varierende, vil vi her ta utgangspunkt i et videoklipp komprimert etter eksemplet gitt i MPEG-standarden. Den viser en koding av mpeg-bildegrupper som følger:

```
I  P  B  B  P  B  B  P  B  B  P  B  B     I .....
1  4  2  3  7  5  6  10 8  9  13 11 12    1
```

I dette eksempelet er størrelsen henholdsvis 19.000 Byte, 10.000 Byte og 2.900 for I-,P- og B-rammer. Vi forutsetter bildegrupper som er kodet uavhengig av andre bildegrupper.

5.5.1 Slow motion

Slow motion eller sakte film på norsk kan oppnås på to måter. Man kan legge inn fiktive rammer i MPEG-videoen slik at flere like bilder vises etter hverandre, eller man kan manipulere presentasjonstiden for hver enkelt ramme og dermed oppnår man at hvert bilde vises lengre enn normalt. Innleggelse av fiktive rammer kan enten skje inne i videotjeneren, eller er gjort tidligere slik at man har en egen fil for dette. Måten rammene legges inn er vist i Figur 5.1



Figur 5.1 Innleggelse av fiktive rammer i MPEG-video.

Vi ser her at annenhver ramme i den nye videostrømmen er tilsynelatende innført uten å bli hentet fra den originale strømmen. De tomme P-rammene er satt inn uten hensyn til bildemateriale i videostrømmen. Det eneste man må vite er bildestørrelsen til bildene. De ekstra B-rammene som er satt inn er derimot hentet fra videostrømmen. Den er en kopi av B-rammen foran i strømmen. Man må huske på å endre bildenummereringen i videostrømmen for å være sikker på at videoavspilleren tolker strømmen korrekt. Med denne metoden kan man oppnå alle hastigheter som kan beskrives ved formelen 1/x der x er et heltall, men ikke null. Den egner seg best for sakte film fremover fordi man ved sakte film bakover må dekode hele bildegruppen for å kunne vise bildene. Dette vil kreve stor lagringskapasitet i mottakers hukommelse samtidig som alle rammene i bildegruppen må leveres over nettet før bevegelsen i filmen skjer. Det vil være lite aktuelt å lagre en slik fil på forhånd da størrelsen på den vil overskride størrelsen på den originale filen og da mister man noe av fordelene med å bruke MPEG-1, nemlig størrelsen på de komprimerte filene.

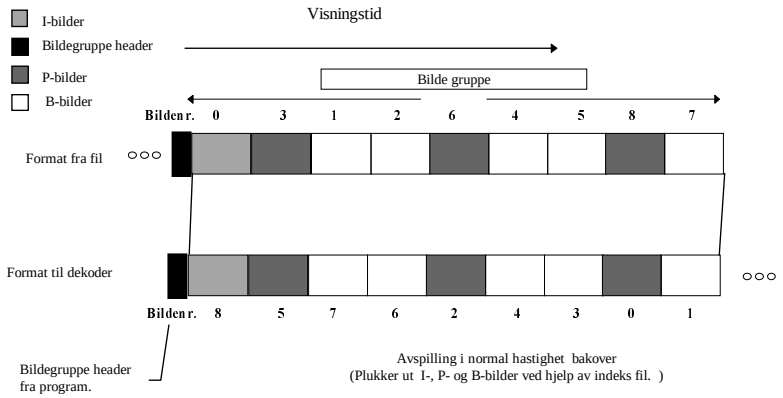
Ved manipulering av presentasjonstiden må rammene legges inn i systemstrøms-pakker eller man må på en annen måte overføre presentasjonstider for bildene. Disse kan f.eks. sendes separat i en slags rammeheader for sendingen. Ved å øke avstanden mellom presentasjonstidene vil man oppnå at hvert bilde vises i lengre tid og man oppnår sakte film. Hastigheter som kan oppnås er bare begrenset av presentasjonstidens oppløsning. Dette gir hvilken hastighet under 1 man kan ønske seg. Denne funksjonaliteten kan også overføres til klienten slik at omregning av presentasjonstider gjøres her

En tredje, men uaktuell mulighet, er å dekode bildene i videotjeneren på forhånd for så å sende flere av samme bilde til videospilleren.

5.5.2 Normal hastighet bakover

Normal hastighet bakover kan oppnås ved å benytte følgende metoder. Om nummerering av bilder i hver bildegruppe og ved endring av presentasjonstider for rammer benyttes.

Omnummerering av temporal referanse for en bildegruppe er vist i Figur 5.2. Her blir hver ramme omnummerert slik at de vises i omvendt rekkefølge. Man leverer da hver bilde gruppe i omvendt rekkefølge, mens bildene innenfor hver av gruppene leveres i riktig rekkefølge.



Figur 5.2 Omnummerering av rammer.

Ved endring av presentasjontider må man bytte om på presentasjonstiden for hver ramme i bildegruppene, slik at rammen med den laveste presentasjonstiden får den høyest osv. Her må man også lever bildegruppene i omvendt rekkefølge og bildene innenfor hver gruppe i riktig rekkefølge.

En forutsetning for at dette skal fungere perfekt er at bildegruppene er lukket. Hvis bildegruppene er åpne vil de siste B-rammene i hver bildegruppe bli dekodet feil. Hvilken metode som bør benyttes er avhengig av om man har tilgang til en presentasjonstid eller ikke. Hvis man har det kan metodene likestilles. Det vil også her kreves en del lagringskapasitet hos videoavspiller da hele bildegrupper må dekodes og lagres før bildene kan vises.

5.5.3 Spoling (hastigheter høyere enn normal avspilling)

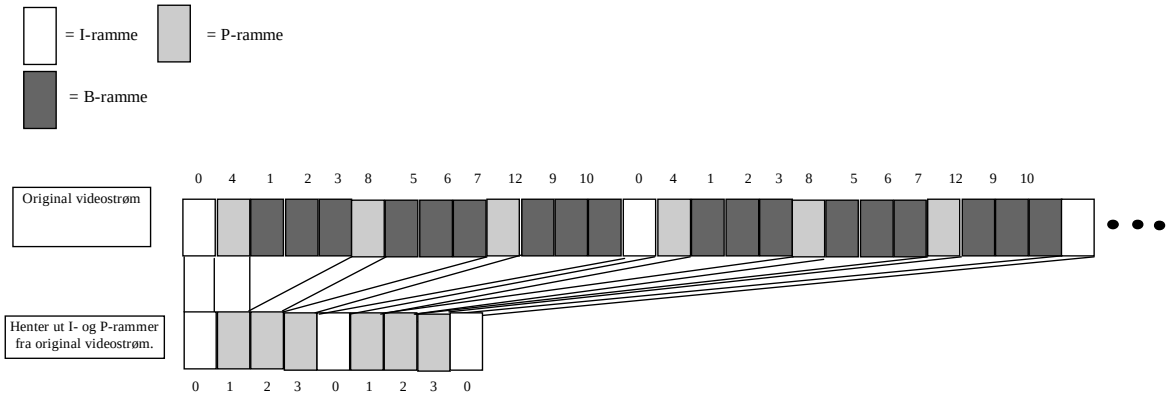
Spoling i MPEG-1 video kan oppnås på mange ulike måter. De metodene som er funnet presenteres i dette delkapitelet. Til slutt i delkapitelet vises et eksempel der metodene er brukt på data basert på en MPEG-1 videostrøm fra standarden [MPEG-92]

5.5.3.1 Spoling med I-rammer

Figur 5.3 Spoling med I-rammer.

Den enkleste metoden for å få til spoling i MPEG-1 video, er å vise kun I-rammer. Disse rammene er uavhengig av de andre rammene i videoen og vil derfor kunne plukkes ut og settes sammen til en ny MPEG-1 strøm, uten at bildene forandrer seg. Vi vil da få en spoling der ca hvert 15 bilde (avhengig av hvordan videoen er kodet) i den originale videoen vises. Ved samme rammerate som ved vanlig avspilling vil det gi en spolehastighet på ca 15 ganger.

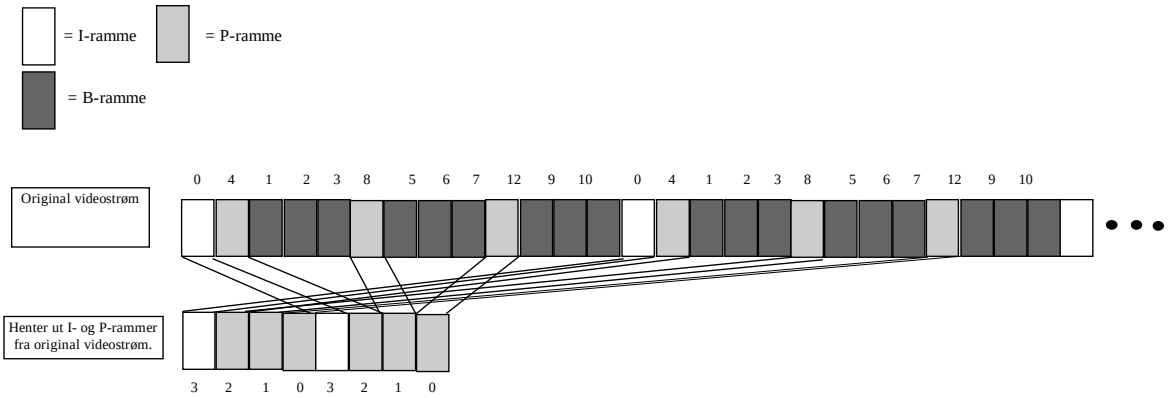
5.5.3.2 Spoling med I- og P-rammer



Figur 5.4 Spoling fremover med I- og P-rammer.

Ved spoling med I- og P-rammer hopper man over visning av alle B-rammer. Dette kan gjøres siden P-rammene kun er avhengig av nærmeste I- eller P-ramme. Nå vises ikke lengre hvert bilde, men ca. hvert 3. bilde. Ved samme rammerate som vanlig avspilling, vil vi ha en spolehastighet på ca 3 ganger.

Denne metoden må modifiseres for spoling bakover. Det er sende rammene til dekoderen i omvendt rekkefølge, fordi det gir dekoderen feil avhengigheter mellom rammer. Man løser dette på følgende måte (se Figur 5.5):



Figur 5.5 Spoling bakover med I- og P-rammer.

Hver ramme innenfor en bildegruppe sendes til dekoderen i riktig rekkefølge. Den eneste forandringen som gjøres i bildegruppen er at hver rammene nummereres om, slik at siste ramme i bildegruppen vises først. Hver bildegruppe vises i så i omvendt rekkefølge.

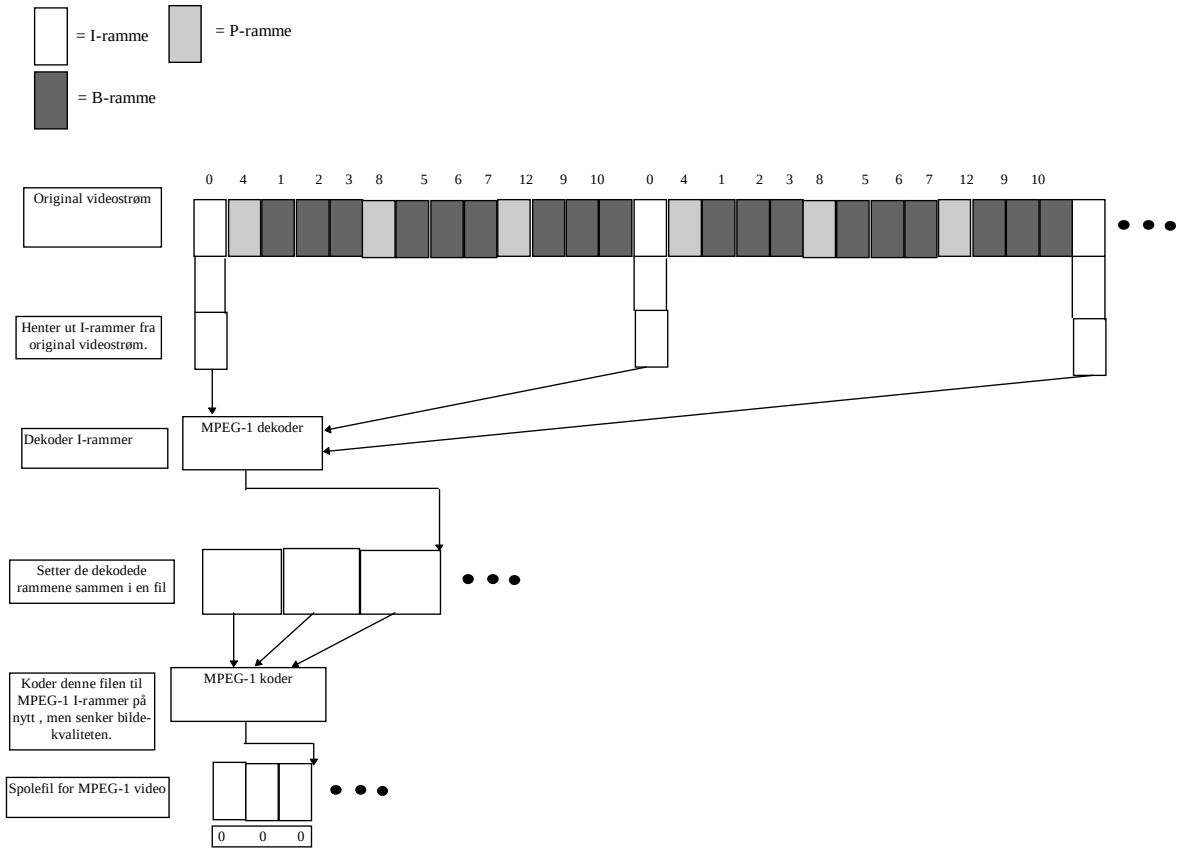
5.5.3.3 Spoling med I-, P- og B-rammer

For å få til denne typen spoling, kreves det at man har muligheter til å styre videodekoderen med presentasjonstidskoder. Det vil si, man har mulighet for å fortelle dekoderen når en ramme skal vises på skjermen. Hvis man har mulighet for det kan oppnå spoling ved å minske avstanden mellom presentasjontidskodene. Dette vil gjøre at hver ramme vises i en kortere periode slik at rammeraten økes.

For spoling bakover brukes samme metoden som vist for spoling med I- og P-rammer.

5.5.3.4 Spoling ved hjelp av spolefil

Hvis man i stedet lager en egen spolefil for spoling der man plukker ut alle I-rammene i originalfilen og koder disse på nytt enten som en vanlig MPEG-fil med I-, P- og B-rammer, eller som en MPEG-fil med kun I-rammer. Man kan på denne måten oppnå lavere bitrater enn ved de andre metodene. Fordelen med å bruke kun I-rammer i spolefilen er man kan hoppe inn i filen på hvilken som helst ramme uten å ta hensyn til rammeavhengigheter. Ulempen er at med å bruke kun I-rammer øker filens størrelse. Filstørrelsen kan reduseres ved å redusere bildekvaliteten i spolefilen.

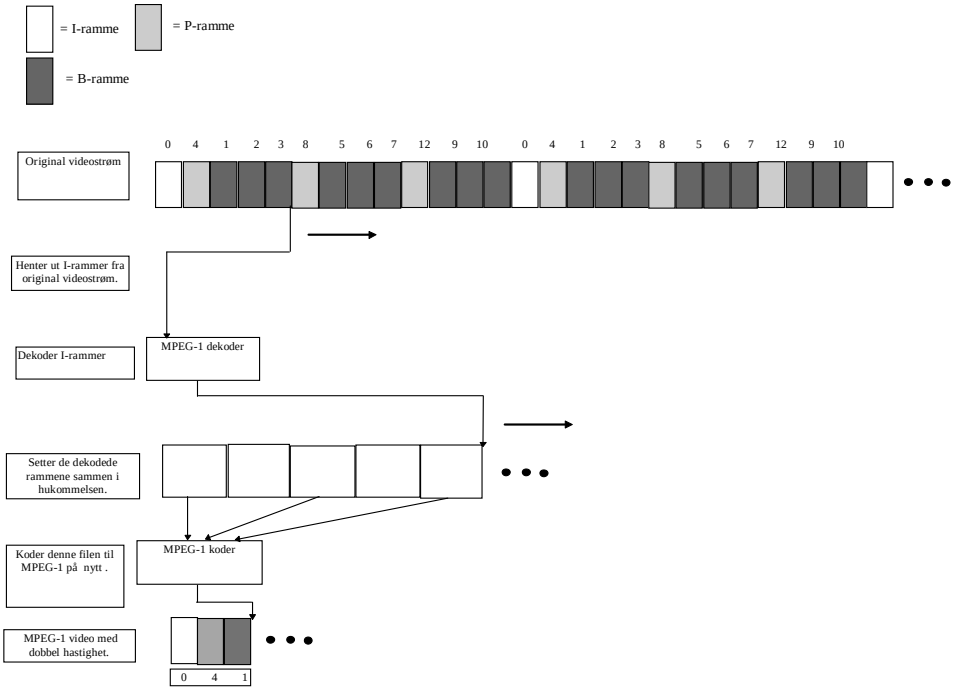


Figur 5.6 konstruksjon av en spolefil.

Ved å plukke hver I-ramme fra videofilen, dekode disse og kode disse på nytt kan vi lage en spolefil. Vi kan kode den nye filen som på samme måte som originalfilen, f.eks med kode sekvensen IPBBBPBBBPBBB, eller vi kan kode den med en dårligere bildekvalitet og kun benytte I-ramer. Hensikten er å gi spolefilen samme datarate som originalfilen eller lavere. Fordelen med å benytte kun I-rammer er at man kan hoppe inn på tilfeldige rammer i spolefilen og begynne å dekode direkte. Det er også lettere å oppnå spoling bakover hvis alle rammene er uavhengig av hverandre. Ulempen er at bildekvaliteten senkes. Siden ulempen med lavere bildekvalitet er såpass små, vil en løsning med kun I-rammer være den mest fleksible løsningen. Dette er en tilfredsstillende løsning, men må kunne håndteres av programvaren i en eventuell videotjeneren. Denne løsningen vil kunne gi de samme muligheten som over hvis man ønsker en spolehastighet ulik den som representeres av spolefilene. Man kan dobble spolehastigheten ved å kun sende over annen hver ramme fra spolefilen eller man kan senke spole hastigheten ved å legge inn en tom P-rammer etter hver ramme, eller sende samme ramme flere ganger.

Lavere videokvalitet kan vi få ved å minke bildestørrelsen, senke rammehastigheten eller endre kvantiseringsmatrisene til makroblokkene. Man kan da tenke seg at man kun benytter I-rammer og B-rammer i filen. Det gir mulighet for flere aksesspunkter i filen. Dette vil øke bitraten som kreves i forhold til å bruke I-,P- og B-rammer, men ved å bruke en lavere bildekvalitet kan man holde seg innenfor grensene satt av for avspillingsfilens bitrate. En annen stor fordel med spolefil er at man ved spoling uten spolefil må plukke ut data fra disk med store mellomrom noe som vil ta lengre tid enn å plukke fram en spolefil. Det gir derfor også mulighet for flere klienter på samme server node.

5.5.3.5 Spoling ved hjelp av sanntids kodere/dekodere

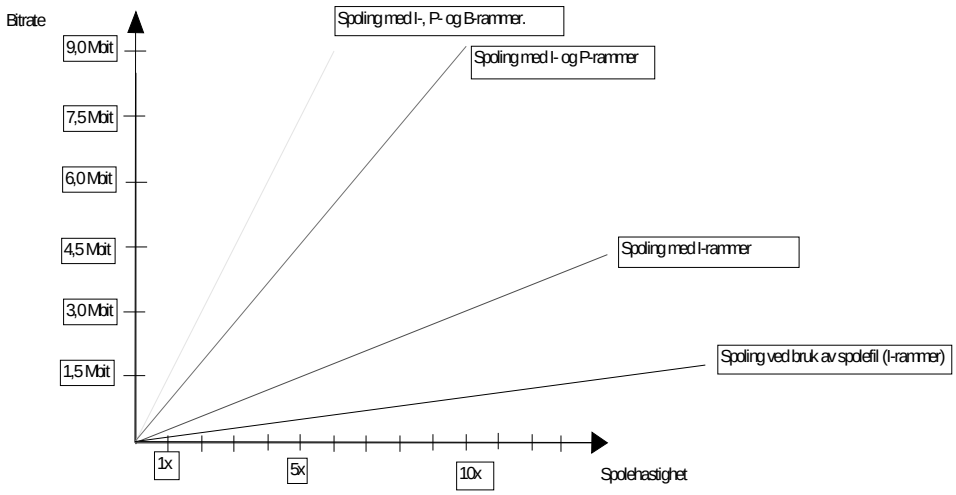


Figur 5.7 Spoling med 2 ganger normal hastighet.

Man kan også tenke seg en løsning der man tar utgangspunkt i en sanntids maskinvarebasert dekoder/koder med ekstrem kapasitet. Man kunne da tenke seg at man ved spoling dekodet videoklippet fra der spolingen starter for så å plukke ut bilder med ønsket mellomrom for den ønskede spolehastigheten. Disse settes sammen til et nytt klipp og koder disse til en egen MPEG-fil og sende disse over nettet. Dette er en løsning som er attraktiv for klienten, men som krever enorme ressurser hos tjeneren. Det vil bety at man må ha en koder for hver klient, eller i hvertfal så mange kodere som vi vil tillate samtidige klienter å spole. Denne løsningen vil være mer aktuell ettersom prisene på kodere faller og hastigheten på datamaskiner øker.

5.5.3.6 Beregninger for noen av de presenterte metodene for spoling

Vi vil her vise hva som skjer med bitratene til en MPEG-strøm når man forsøker å bruke de ulike metoder for spoling i MPEG-video. Dette presenteres ved hjelp av Figur 5.8.



Figur 5.8 Bitrater for spoling i MPEG-video.

Figur 5.8 er satt opp ved å bruke data fra et eksempel i [MPEG-92]. Den viser bitrate for spoling i forhold til spolehastighet for noen av spolemetodene. Ved å sammenligne Figur 5.8 med Figur 5.9 ser man hvilke rammerater som er grunnlag for de enkelte spolehastigheter. Normal avspilling har en bitrate på ca. 1,5 Mbit/s. Hvis man ønsker å oppnå spoling over et nettverk vil denne verdien være den magiske grensen man må holde seg under for å unngå høyre belastning ved spoling i forhold til normal avspilling.

Vi antar en ramme hastighet på 25 rammer pr. sekund. Det utgjør i følge eksempelet 2 I-rammer pr. sekund, 8 P-rammer pr. sekund og 15 B-rammer pr sekund. Vi får da:

2 rammer*19.000 + 8 rammer*10.000 + 15 rammer*2.900 = 161,5 kByte/s

Ved spoling med I-,P- og B-rammer er det umulig å få til noen spoling som ikke overgår bitratene til vanlig avspilling siden man sender alle rammene som vises ved vanlig avspilling. Dette er denne spolemetoden som vil belaste nettverk og harddisk mest i forhold til spolehastighet. Det er også den spolemetoden som gir den beste spolekvaliteten.

Ved spoling med kun I-rammer vises for dette eksempelet hvert 13. bilde. Dette vil gi en mer hakkete fremstilling av spolingen, særlig ved lave hastigheter. Fordelen med metoden er at bitraten senkes mye i forhold til noen av de andre metodene. Man ser likevel at bitraten til vanlig avspilling overgås ved ca. 5 ganger spolehastighet. Hvis man ønsker å vise kun I-rammer med samme rammehastighet som ved normal avspilling vil mengden data bli tre ganger større. Dette sees av dette regnestykket:

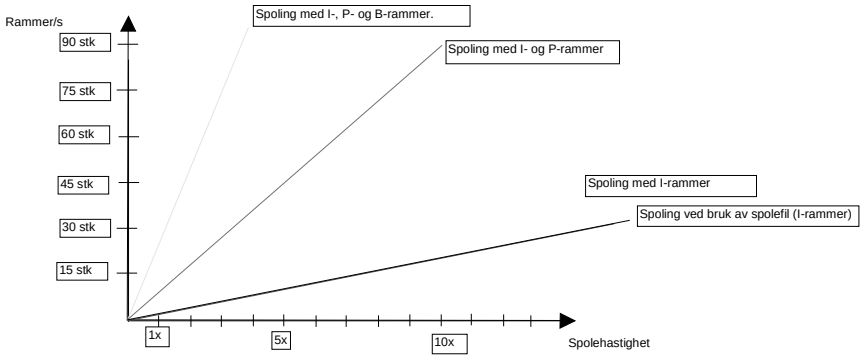
Spoling med I-rammer i samme rammehastighet som ved vanlig avspilling. Det vil si 25 I-rammer per sekund:

25 rammer * 19.0000 = 494 kBytes/s

Ved spoling med I- og P-rammer, vil vi få en bedre flyt i spoling. Man viser da ca hver tredje ramme, og får dermed en spolehastighet på 3 ganger ved en rammerate lik normal avspilling, bitraten vil da bli :

6 rammer*19.000 + 19 rammer*10.000 = 304 kByte/s

Dette utgjør ca en dobling av overføringshastigheten.



Figur 5.9 Rammerater for spoling i MPEG-video.

Hvordan kan man forhindre denne økningen i nettverksbelastning? En løsning kan være å prøve å sende færre bilder over nettet ved spoling. Siden nettet belastet 3 ganger mer ved spoling med I-rammer, kan man sende 1/3 så mange bilder. Man viser da 8 rammer per sekund isteden for 25. Dessverre tillater ikke MPEG-standarden at ramme-raten reduseres med 1/3. Den har kun faste rammerater på fra ca 24 rammer per sekund til 60 rammer per sekund. Man kan imidlertid "jukse" dette til ved å forandre på tidskodene for pakker i systemlaget.

En annen måte å løse dette på er å legge inn «tomme» P-rammer etter en I-ramme. Disse «tomme» P-rammene inneholder kun data som sier at foregående bilde skal vises på nytt. De blir allikevel behandlet som en vanlig ramme. Siden størrelsen på denne P-rammen er veldig liten, ca 1/600 av en I-ramme (størrelsen var på 32 Byte ved testing). Ved å vise hver tredje I-ramme og legge inn 2 slike «tomme» P-rammer mellom hver I-ramme oppnår vi en spolingseffekt på 13 ganger uten å øke belastningen på nettet. Kvaliteten på spolingen vil være redusert fordi mindre informasjon fra videoen blir vist. Men med et snitt på et bilde vist fra hvert 1,5 sekund kan man allikevel orientere seg i videoklippet.

5.6 Hvilken metode skal velges?

Hvilken løsning man velger avhenger mye av hvilke ressurser man har til rådighet.

Har man store krav til kvaliteten på bilde under spoling og ønsker spoling i alle tenkelige hastigheter kan man få dette ved hjelp av sanntids kodere/dekodere. Det er meget dyrt og krever da en koder/dekoder for hver klient som vil spole på et gitt tidspunkt.

Hvis man har plass på disk/tape og ønsker kun noen få spolingshastigheter og tåler en viss forringelse av bildekvaliteten kan man velge å benytte seg av egne spolefiler. Dette vil særlig forenkle bevegelse bakover i videoklippet.

Hvis man ønsker den enkleste løsningen å implementere, velger man spoling med I- og P-rammer. Denne vil fungere, men kvaliteten på spolingen vil være lavere enn de to foregående

løsningene. Man vil bruke mindre plass på disk/tape enn ved bruk av spolefiler. Man vil få en billigere løsning enn å bruke sanntidskoder/dekoder

Vi vil i dette prosjektet konstruere følgende løsninger. For spoling velges en løsning der man benytter seg av spolefil og en annen der spoling ved hjelp av I-rammer benyttes. For sakte film benyttes manipulering av presentasjonstider og ved avspilling bakover benyttes omnummerering av rammer.

5.7 Spoling i audio

Man kan vise annen hvert ramme i en video og få et godt inntrykk av hva som skjer allikevel. Man kan også spille videoen baklengs uten at man mister sammenhengen. Det er ikke like lett å få til dette for audio. Man kan hvis det er ønskelig få til en slags spoling av audio i fremoverrettning, som faktisk er forståelig ved lave spolehastigheter. Ved f.eks 2 ganger spoling kan man hoppe over annenhver ramme i audioen og likevell høre hva som sies. Grunnen til at dette går an er at hver ramme dekker et såpass lite tidsrom med lyd at man klarer å tyde lyden uten de manglende bitene. For lag 2 som er den mest brukte komprimeringsmetoden inneholder hver ramme 1152 sampler. Ved en samplingfrekvens på 44.1 kHz, vil da hver ramme inneholde audio som varer i ca. 2.6 ms med lyd.

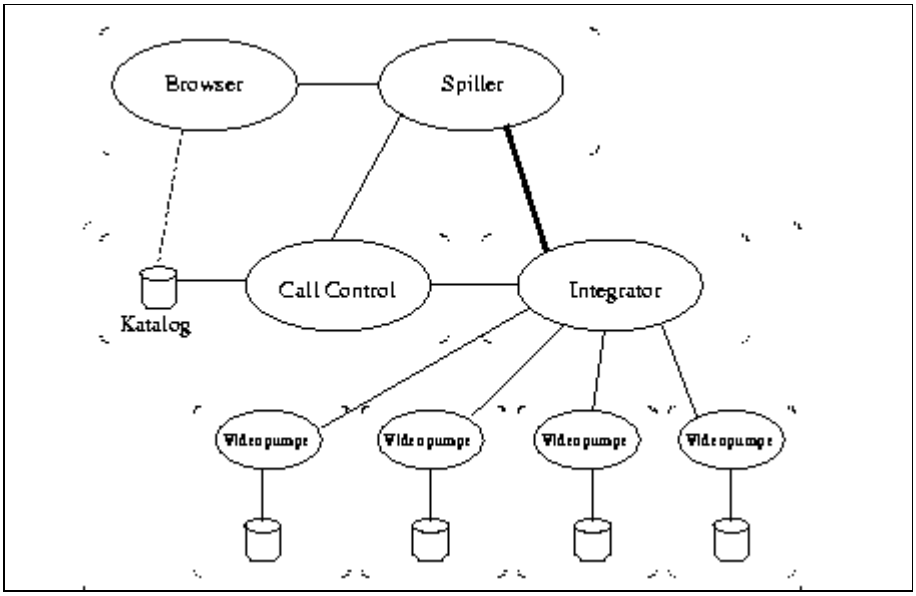
6. Konstruksjon

I dette kapittelet presenteres konstruksjonen av videotjeneren og klient. Vi vil først se på hvilke
erfaringer som kan hentes fra et tidligere hovedoppgave som ble utført av Stein Langørgen ved IDT's Gruppe for databaseteknikk. Prosjektet gikk under navnet ELVIRA (EksperimenteL Videotjener for ATM)[Langørgen-94]. Her ble det konstruert en videoserver for avspilling av Motion-JPEG video. Deretter vil en overordnet konstruksjon presenteres, for til slutt å vise konstruksjon av de enkelte moduler.

6.1 Analyse av videoserveren ELVIRA

Vi vil her gi en kort oversikt over ELVIRA's arkitektur og erfaringer som kan trekkes ut fra denne arkitekturen.

6.1.1 Arkitektur



Figur 6.1 Oppbygning av ELVIRA.

Figur 6.1 viser ELVIRA's arkitektur slik den var når diplomoppgaven til Stein Langørgen ble levert [Langørgen-94]. Den besto da av følgende komponenter:

Klient :

- En videospiller for avspilling av Motion-JPEG video med tilhørende PCM-kodet lyd.
- En browser som har oversikt over hvilke filmer som er tilgjengelig til enhver tid.

Tjener:

- En Call Control prosess, for behandling av ønsker om levering av video fra en klient, belastningsmålinger og styring av leveranse.

- En Integrator-prosess per leveranse for levering av video til klient i forskjellige formater.
- En videopumpe per leveranse per node for henting av video fra disk

Hver enkelt komponent er en uavhengig prosess og kommunikasjonen skjer derfor via sockets eller meldingskøer.

Videotjeneren er request-dreven dvs. det er videospilleren som ber om å få overlevert et vist antall rammer i løpet av et gitt tidsintervall, og første ramme må være levert innen en gitt tidsfrist. Det betyr at mye av videokontrollen ligger hos videospilleren. Det gir en del unødvendig buffering hvis antall rammer som oversendes er for stort.

ELVIRA har støtte for striping. Det er altså mulig å fordele en videofilm over flere disker og flere maskiner. Dette for å sikre lastbalansering og dermed bedre utnyttelse av den totale disk-kapasiteten i systemet.

ELVIRA har også støtte for virtuelle dokumenter. Et virtuelt dokument lages ved å sette sammen to eller flere filer til en helt ny film uten å lagre det nye dokumentet. Det er dermed videotjeneren sitt ansvar å sett disse filene sammen under avspilling slik at det for brukeren av tjeneren ser ut som en film.

6.1.2 Erfaringer og problemer med ELVIRA

ELVIRA må sies å være en vellykket konstruksjon. Den brukes i dag bla. ved IDT og Norsk Regnesentral. Den har imidlertid gått igjennom noen ansiktsløftninger etter hovedoppgavens innlevering.

6.1.2.1 Striping/integrator

Under eksperimenter med stripping fant Stein ut at videotjeneren kunne levere flere videostrømmer uten striping i forhold til med striping. Dette forutsatte at alle filmer var tilgjengelig på alle disker eller at klientene ønsket å se filmer slik at det ble en jevn belastning på hver node. Siden stripingen gikk mellom flere maskiner gikk nettverksbelastningen opp ved striping. Det er en utvikling som man ikke ønsker da nettverkskapasitet er en flaskehals i seg selv. I tillegg krever integratoren mye minne fordi den må lagre rammer som kommer fra pumpene inntil den kan sette de sammen i riktig rekkefølge. Det viste seg også at integratoren ble en meget tung prosess å drive på grunn av all nettverkstrafikken. Ved kjøring av flere enn en integrator på samme maskin fikk man problemer med nettbelastningen.

6.1.2.2 Synkronisering og belastning

ELVIRA benytter request-drevet videoleveranse. Dette gjør at belastningen på videoserveren er ujevn. Vi har en typisk arbeid/hvile fordeling på 1 sekund i arbeid og 1 sekund hvile. Dette kan slå uheldig ut hvis alle prosesser arbeider samtidig og dermed også hviler samtidig. Nettverket blir også unødvendig hard belastet ved et slikt design. ELVIRA benytter flere prosesser for videoleveranse. Dette gir synkroniseringsproblemer, spesielt ved striping da de forskjellige prosessene arbeider på forskjellige maskiner.

6.2 Endringer i design i forhold til ELVIRA

Her vil vi oppsummere hovedpunkter for forskjeller mellom den nye videotjeneren og ELVIRA.

6.2.1 MPEG-video med VCR-funksjonalitet

Hovedpunktene i endringer i forhold til ELVIRA er MPEG-video med VCR-funksjonalitet. Den første utgaven av ELVIRA hadde kun støtte for for Motion-JPEG video. ELVIRA ble senere utvidet med primitiv støtte for MPEG, dvs. kun sekvensiell avspilling. I den nye videotjeneren vil vi ha muligheter for fleksibel avspilling av MPEG. Dette innebærer at man har mulighet for spoling av video i begge retninger med ulike hastigheter. Man har muligheter for avspilling i sakte film i begge retninger. I tillegg kommer mulighet for stillbilde og hopp i videoen.

6.2.2 Synkronisering av video-leveranse

Leveranse av video er styrt av videotjeneren. Klienten gir kun kommandoer som play, fast forward, rewind osv., resten tar tjeneren seg av. Tjeneren leverer video med en ramme av gangen for å gi en mest mulig jevn belastning på nett og på CPU.

6.2.3 Striping/integrator

Vi ønsker å ikke ha med mulighet for distribuert striping og behøver dermed ikke lenger en integrator. Grunnen til at vi ikke ønsker dette er at det i forrige versjon av ELVIRA fungerte det dårlig i praksis, og ble derfor lite brukt. Det ville også komplisere avspillingen av MPEG-video. En mulighet som kan utprøves er lokal striping over flere disker på hver maskin.

6.2.4 Bruk av tråder

Forrige versjon av ELVIRA var basert på at hver videoleveranse ble utført av en eller flere samarbeidende prosesser. Prosess-skifter er ressurskrevende. For å minske belastningen på maskin/CPU vil vi eksperimentere med bruk av tråder i stedet for prosesser.

6.2.5 Bruk av Lava-prosjektets nye kommunikasjonssystem

Man har ved Norsk Regnesentral konstruert et nytt kommunikasjonssystem for levering av video over nett. Dette er egentlig utviklet for et bankovervåkningssystem som Norsk Regnesentral har laget, men skal også ha andre anvendelsesområder. Dette brukes for å gi samme grensesnittet til nettverk på alt som lages under LAVA-prosjektet.

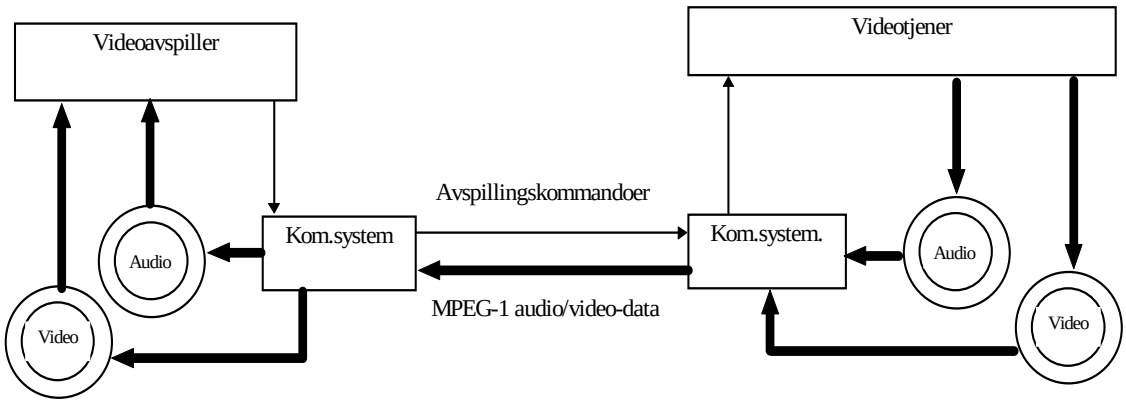
6.2.6 Gjennbrukte komponenter

Store deler av diskscheduleren og noen objekter rundt denne vil bli gjennbrukt. Dette fordi oppbygningen virker effektiv, stabil og funksjonell. Bortsett fra dette vil mesteparten bli laget på nytt.

6.3 Konstruksjon av videotjener

I denne delen av rapporten vil løsningen av leveranse av video til videospilleren forklares. Vi vil først gå innpå den overordnede funksjonaliteten for senere å gå inn på spesifikke detaljer i konstruksjonen.

6.3.1 Overordnet beskrivelse



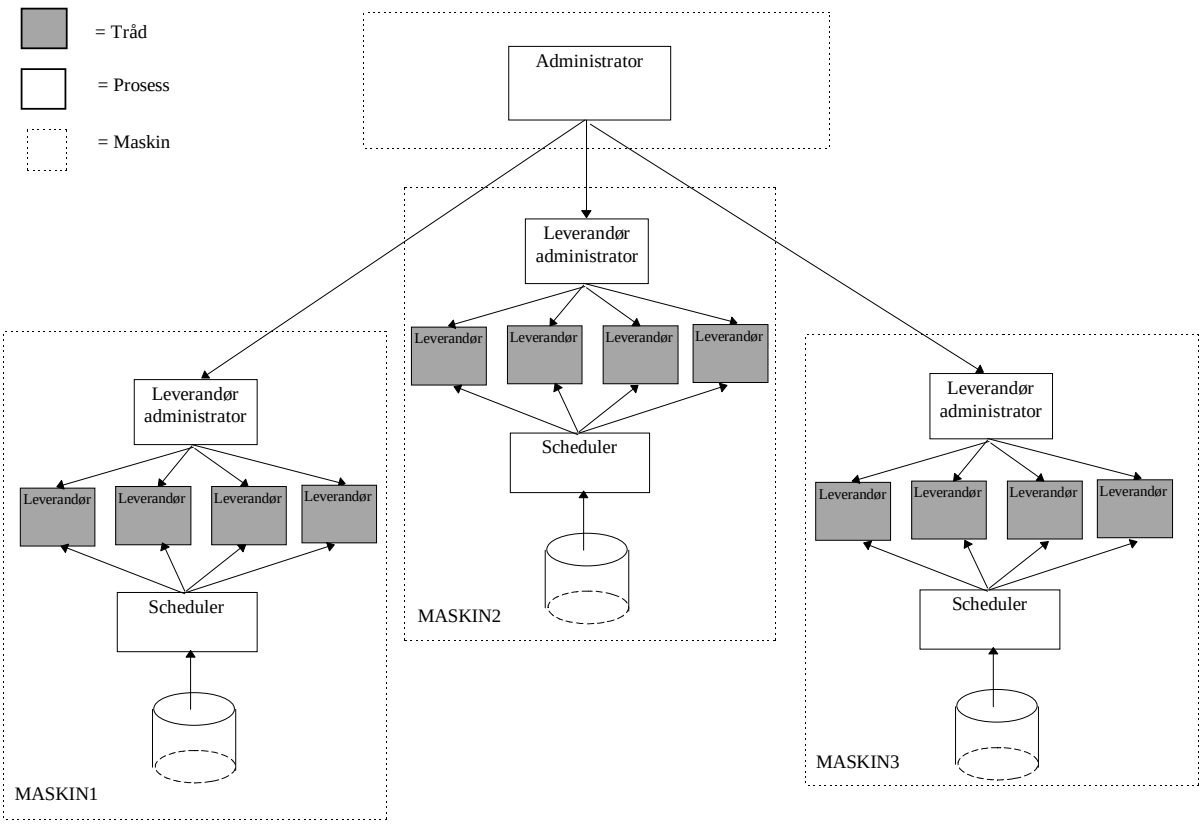
Figur 6.2 viser overordnet konstruksjon av en videoleveranse.

Systemet består av tre hovedkomponenter (se Figur 6.2):

- Videotjener, for henting av videodata fra disk og levering av disse til kommunikajonssystemet
- Videoavspiller, for mottak av videodata, dekoding og visning av disse.
- Kommunikasjonssystem, for overføring av dataene over nettverk.

Hovedfokus i denne oppgaven er videotjeneren. For å gi et komplett bilde av det fullstendig systemet, er det også tatt med en oversikt over kommunikasjonssystemet og klienten.

6.3.2 Videotjener



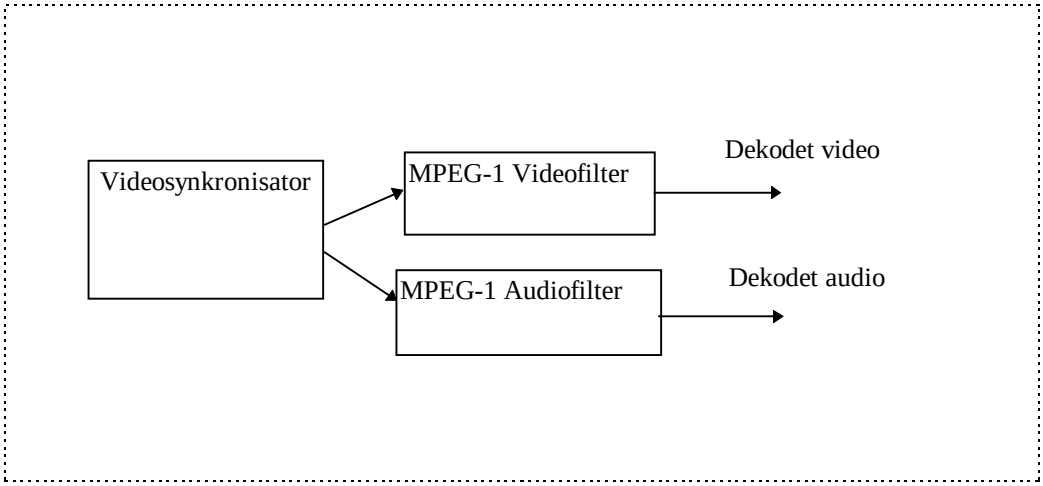
Figur 6.3 Oppbygning av videotjener.

Figur 6.3 viser oppbygningen av videotjeneren. Antall maskiner med leverandørprosesser er avhengig av antall maskiner tilgjengelig. At tegningen viser tre maskiner er tilfeldig.

Videotjeneren er bygget opp på følgende måte:

- Vi har en administrator-prosess. Den skal ta imot ønsker fra klienter om levering av video. Ønsker om levering vurderes etter belastninger på maskinen der filmen ligger. Hvis belastningen ikke er for stor sendes en melding om levering av video til en leverandør-administrator.
- På hver maskin i systemet ligger en leverandøradministratorprosess og lytter på meldinger fra administratorprosessen. Ved mottak av en melding med instruksjoner om levering av video, starter leverandør administratoren opp en leverandørtråd.
- Leverandørtråden er kjernen i systemet. Det er denne som styrer henting av data fra disk, formaterer dataene og sender disse til klienten. Levetiden til en tråd er en videoleveranse.
- Scheduleren tar imot lesemeldinger fra leverandørtråder, henter data fra disk og gir de tilbake til tråden.

6.3.3 Videoavspiller

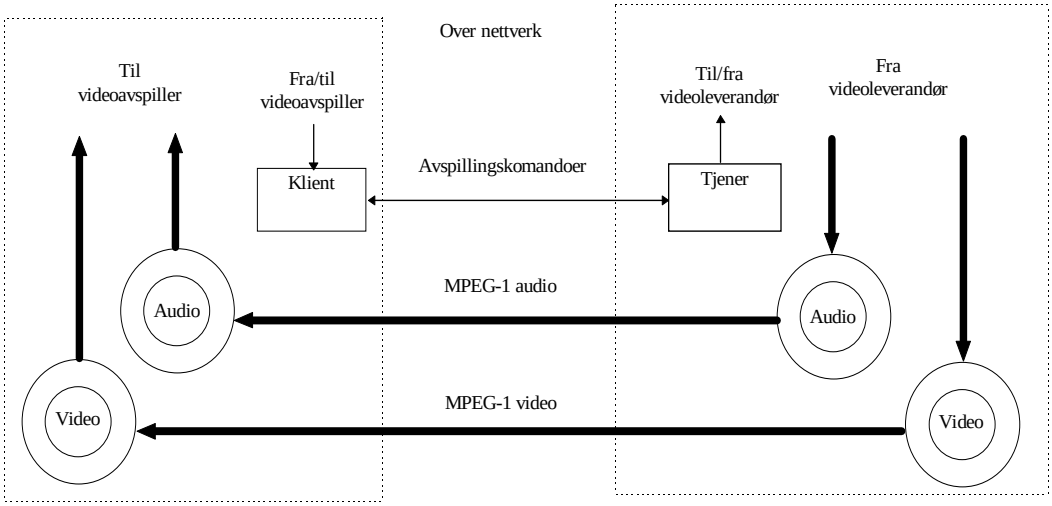


Figur 6.4 Oppbygning av videoavspiller

Videoavspilleren består av følgende komponenter:

- En videosynkronisator som sender avspillingskommandoer til en videoleverandør, tar imot MPEG-1 data fra en videoleverandør og synkroniserer audio- og videofiltrene for behandling av MPEG-1 dataene.
- Et MPEG-1 videofilter som dekoder MPEG-1 video i henhold til MPEG-1 videostandarden.
- Et MPEG-1 audiofilter som dekoder MPEG-1 audio i henhold til MPEG-1 audiostandarden.

6.3.4 Kommunikasjonssystemet



Figur 6.5 Oppbygning av kommunikasjonssystemet

Oppbygningen av kommunikasjonssystemet er som følger:

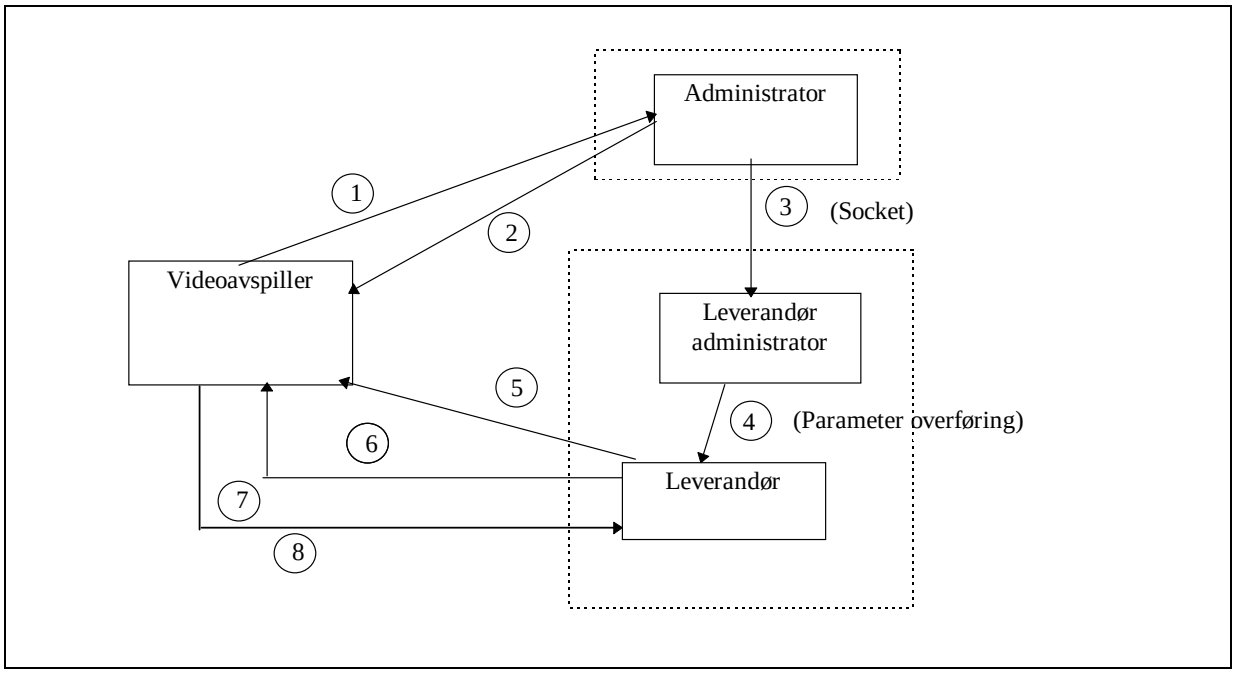
- Vi har fire forskjellige prosesser/tråder.
- En klientprosess som mottar avspillingskommandoer (play,fast forward og rewind osv.) fra videoavspilleren og sender disse til tjenerprosessen via et nettverket.

- En tjenerprosess som mottar avspillingskommandoer fra klienten og lever disse til videoleverandøren i videotjeneren.
- Et ringbuffer for sending av MPEG-1 audiodata fra videoleverandøren og et ringbuffer for mottak av MPEG-1 audiodata til videoavspilleren.
- Et ringbuffer for sending av MPEG-1 videodata fra videoleverandøren og et ringbuffer for mottak av MPEG-1 videodata til videoavspilleren.

Vi vil i de videre beskrivelser regne kommunikasjonssystemet som en del av henholdhvis klienten og tjeneren for å forenkle figurer og beskrivelser.

6.4 Virkemåte for videotjener

Vi gir her en kort beskrivelse av den tiltenkte virkemåten til videosystemet.



Figur 6.6 Oppkobling av forbindelse mellom videospiller og videotjener.

Ved hjelp av Figur 6.6 vil oppkobling av en videospiller beskrives :

1. Administratoren lytter på en fast port som er kjent av videospilleren. Videospilleren tar kontakt med videotjeneren via administrator-prosessen. Den ber om levering av en film, og gir fra seg leveringsadresse for filmen.

2. Administratoren finner ut om ønsket film er tilgjengelig og om det er kapasitet til å levere filmen. Hvis svaret er positivt, sendes en bekreftelse på dette til videospilleren. Forbindelsen tas så ned. Videospilleren vil etter dette vente på svar på den oppgitte portadressen.

3. Administratoren tar kontakt med leverandør-administratoren med beskjed om hvilken fil(m) som skal leveres og hvilken/hvilke porter den skal levere til på hvilken maskin.

4. Leverandør-administratoren starter så opp en videoleverandør-prosess/tråd. Fil(m)navn, maskin og portnummer overføres til prossesen/tråden

5. Videoleverandøren tar kontakt med videospilleren på den oppgitte porten. Forbindelsene settes opp.

6. Levering av video starter i henhold til kommandoer fra videospilleren.

7. Ved å gi ulike avspillingskommandoer kan videoavspilleren styre videoleverandøren til å levere video i ønsket hastighet og retning.

8. Ved kommandoen «avslutt leveranse» fra videoavspiller tas kommunikasjonssystemet ned og leverandørtråden avsluttes. Beskjed om avsluttet leveranse sendes videre til administratoren (via leverandøradministratoren) for oppdatering av belastningsoversikt.

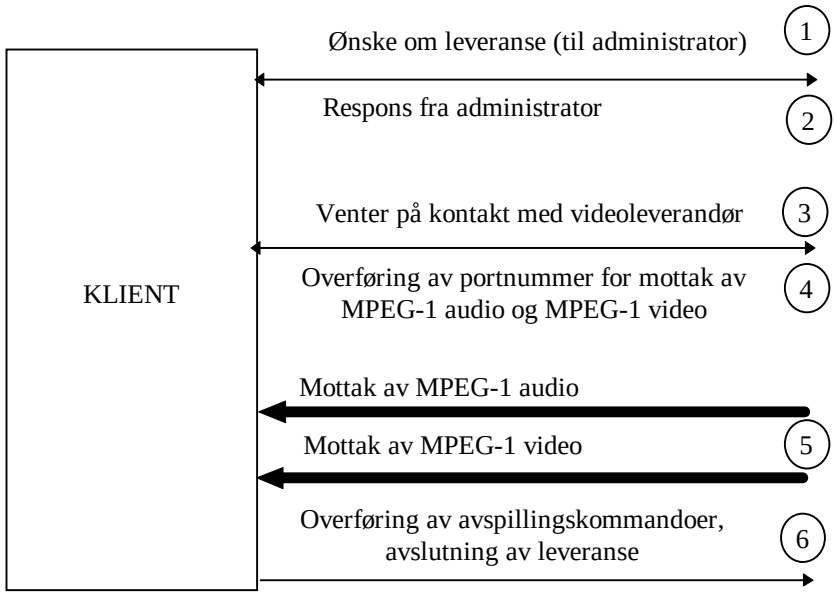
Kommunikasjonssystemet benyttes i stegene 1, 2, 5, 6, 7 og 8.

6.5 Detaljert beskrivelse

Vi vil her gå igjennom oppgavene og virkemåten til hver enkelt objekt i konstruksjonen.

6.5.1 Klienten

Klienten består av en MPEG-1 videoavspiller tilknyttet kommunikasjonssystemet. Detaljene rundt den er imidlertid litt uklare siden den ennå ikke er ferdig. Den skal etterhvert komme fra NORUT, men ingen tidsfrist for levering er satt. Man har ytret ønske om å beholde de originale tidskodene fra systemstrømmen ved overføring. Dette innebærer at eventuelle økninger og senkninger i hastighet ved hjelp av disse fra videoleverandøren er umulig. Denne funksjonaliteten må da eventuelt innebygges i klienten.



Figur 6.7 Klient.

Klienten fungerer på følgende måte :

- Fra kommandolinjen gis informasjon om hvilken maskin som skal kontaktes, hvilken port den skal kontaktes på og hvilken mpeg-film man ønsker å motta data fra.
- Den prøver så og opprette kontakt med administratoren i videotjeneren.
- Hvis kontakt opprettes, sendes navnet på filmen som ønskes avspilles sammen med en ny portadresse. Den portadressen benytter videoleverandøren for leveranse av video.
- Den venter så på svar fra administratoren om filmen er tilgjengelig og det finnes kapasitet til å utføre leveransen.
- Ved positiv respons vil klienten vent på kontakt fra en videoleverandør på den tidligere oversendte portadressen.
- Mottak av video starter
- Under leveranse av video kan den sende ulike kommandoer til videoleverandøren. Disse er oppsummert i delkapittelet om kommunikasjonssystemet.

6.5.1.1 Innvendig oppbygning

Her beskrives klientens oppbygning i detalj.

Ved oppstart av klienten gis følgende parametre:

- Maskin adresse for datamaskin der administratorprosessen befinner seg.
- Portnummer for kontakt med administrator prosessen.
- Navn på ønsket film.

Etter oppstart kontaktes administratoren på den angitte adressen. Følgende data overføres:

- Maskin adresse for datamaskin der klienten befinner seg.
- Navn på film som ønskes levert.
- Portnummer for kontakt mellom klient og videoleverandør.

Hvis ønsket om leveranse godtas av administratoren, vil klienten vente på kontakt fra videoleverandøren på oppgitt port. Når videoleverandøren tar kontakt opprettes nye forbindelser mellom disse for MPEG-1 audio- og videodata. Deretter opprettes ringbufferne for video og audio. Klienten kan nå sende ulike kommandoer til videotjeneren. Kommandoer som er tilgjengelig er:

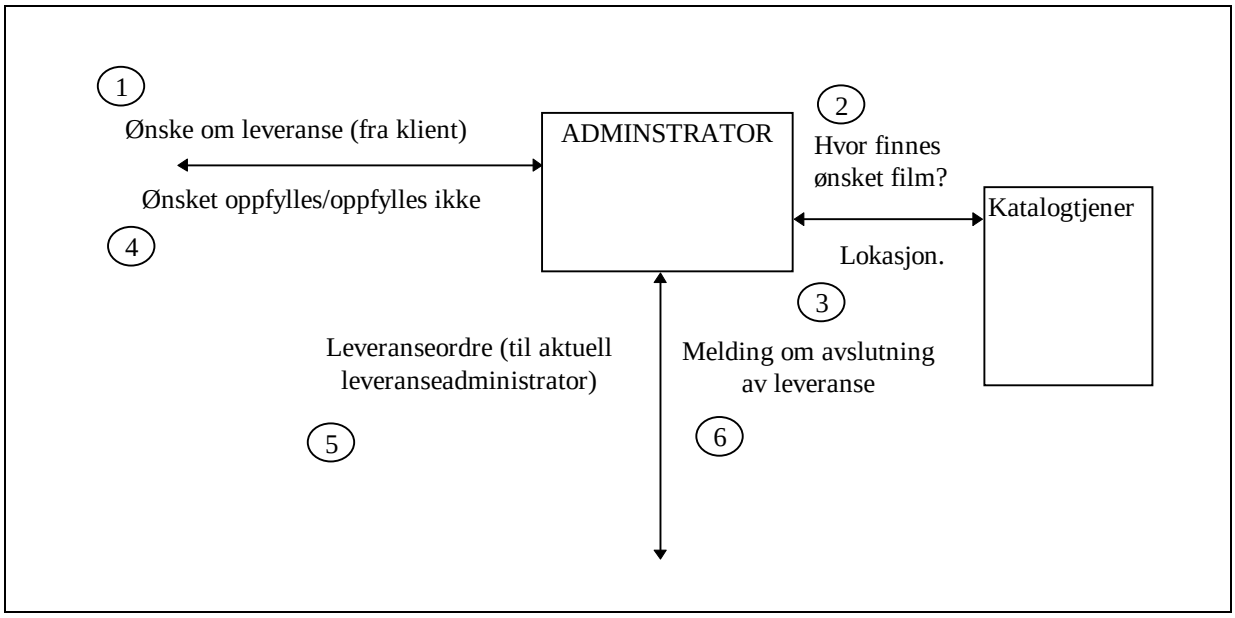
- Play(): Normal leveranse av video.
- Ffw(): Spoling fremover.
- Frw(): Spoling bakover.
- Goto(): Hopp i video til ønsket ramme.
- Stop(): Stilbilde.
- OpenaudioStream(): Åpning av forbindelse for MPEG-1 audio.
- OpenvideoStream(): Åpning av forbindelse for MPEG-1 video.
- CloseaudioStream(): Lukking av forbindelse for MPEG-1 audio.
- ClosevideoStream(): Lukking av forbindelse for MPEG-1 video.
- Close(): Avslutning av leveranse fra klient.

Motatt video hentes fra ringbufferne og leveres til audio og video filtrene. Formatet for lesing av audio/video-data fra kommunikasjonssystemet er:

- Peker inn i ringbufferet
- Lengde på data fra pekerposisjon
- Tidskode for levering til audio/video buffer

6.5.2 Administratoren

Administratoren har til oppgave å behandle ønsker om filmavspilling fra klienter samt å ha oversikt over belastningsfordeling på de enkelte leverandøradministratorene.



Figur 6.8 Administratorprosess.

Administratoren lytter på en fast port for kommunikasjon med klienter. Den har i tillegg fast oppsatte porter for kommunikasjon med de enkelte leverandøradministratorene.

Ved forespørsel om leveranse fra en klient vil følgende hendelser forekomme:

1. Et ønske om leveranse ankommer fra en klient. Dette ønske inneholder filmnavnet på ønsket film, klientens lokale maskinadresse samt portnummer for kontakt fra en videoleverandør.

2. Administratoren går inn i katalogtjeneren og finner ut om filmen eksisterer, hvilken/hvilke maskiner og disker filmen ligger på og hvilken bitrate filmen er kodet for.

3. Ved hjelp av en belastningsoversikt og opplysningene om hvor filmen befinner seg, finner administratoren ut om ønsket kan oppfylles eller ikke.

4. Klienten gis beskjed om resultat.

5. Hvis ønsket skal oppfylles, sendes klientens maskinadressen, filnavn for film og portnummer til leverandøradministratoren. På utvalgt maskin og belastningsoversikten blir oppdatert.

6. Ved avslutning av en leveranse gir leverandøradministratoren beskjed om dette til administratoren, slik at belastningsoversikten kan oppdateres.

6.5.2.1 Innvendig oppbygning

Her beskrives administrator-prosessens innvendige oppbygning i detalj.

Etter oppstart opprettes et server objekt fra kommunikasjonssystemet. Dette settes til å lytte på den angitte porten. Ved mottak av ønske om leveranse fra en klient, slås ønsket film opp i

katalogtjeneren, som er en database over tilgjengelige filmer i videotjeneren. Databasen kan inneholde opplysninger som :

- Film tittel
- Format
- Tid (lengde)
- Bildestørrelse
- Rammerate
- Bitrate
- Størrelse
- Filstørrelse
- Plassering

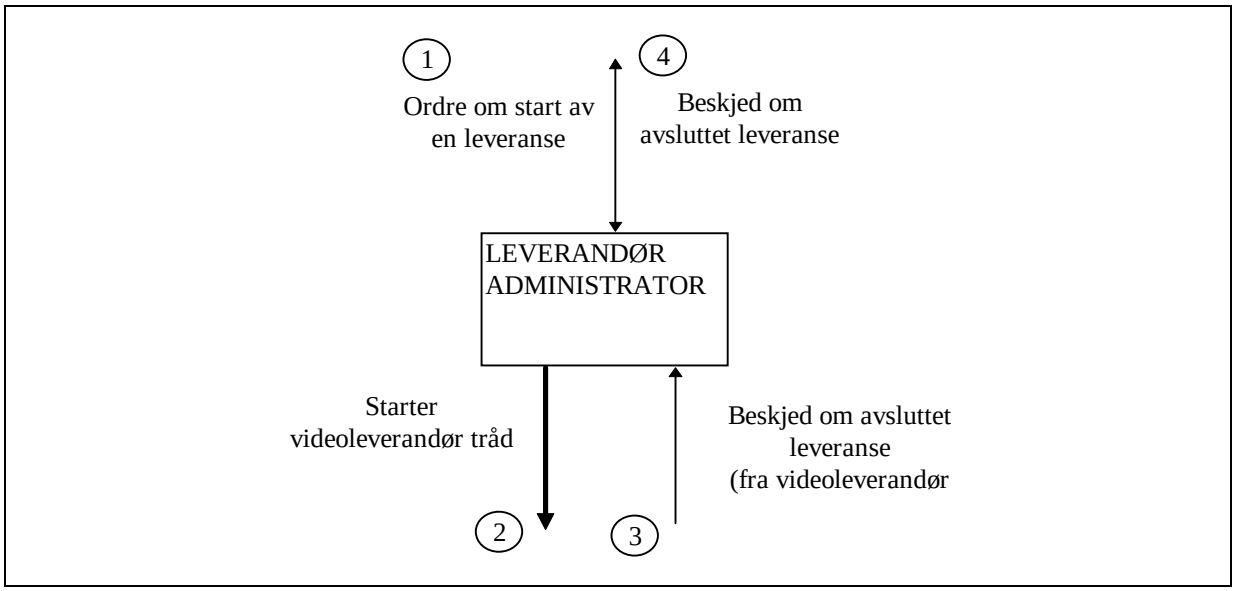
Ved å se på filmens bitrate kan en belastningsfaktor for levering av filmen beregnes. For hver maskin med en leverandøradministrator er det lagret en maksimal belastningsfaktor. Denne sammenlignes med belastningsfaktoren for den nye leveransen pluss belastningsfaktor fra før. Hvis maksimal belastning ikke overstiges godtas leveransen. Belastningsfaktoren for aktuell maskin oppdateres og respons til klienten sendes. Det opprettes så kontakt med den aktuelle leverandøradministratoren via en socket. Beskjed om leveranse av film til aktuell klient sendes leverandøradministratoren der filmen er lagret.
Beskjeden har følgende format:

Klientadresse
portnummer
filmnavn

Så sjekker administratoren om den har mottatt noen beskjeder om avsluttede leveranser. Hvis dette er mottatt oppdateres belastningsfaktoren for den aktuelle maskinen. Deretter venter administratoren på nye ønsker om leveranse fra klienter.

6.5.3 Leverandøradministratoren

Leverandøradministratoren lytter på forbindelsen til administratorprosessen og tar imot ordre om videoleveranser. Ved mottak av ordre startes en videoleverandør. Ved avsluttning av en videoleveranse sender den beskjed om dette til administratoren



Figur 6.9 Leverandøradministrator.

Ved ordre om oppstart av en videoleverandør skjer følgende:

1. Ordre fra administratoren mottas. Denne inneholder klient maskinadresse, navn på ønsket film og portnummer for kontakt med klient.
2. Leverandøradministratoren starter en tråd-prosess der opplysningene fra punkt 1 sendes med.
3. Ved avsluttning av en leveranse gis det beskjed fra videoleverandøren til leverandøradministratoren.
4. Beskjeden om avsluttet leveranse sendes videre til administratoren.

6.5.3.1 Innvendig oppbygning

Her beskrives den innvendige oppbygningen til leverandøradministratoren i detalj.

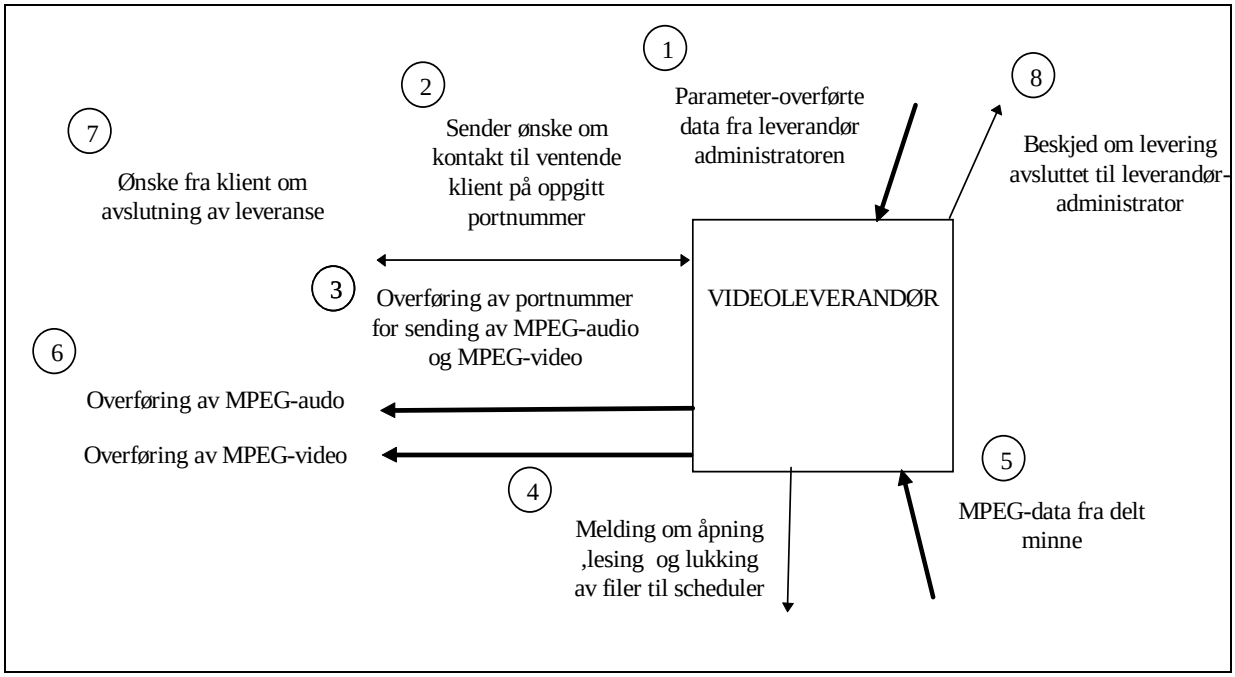
Etter oppstart opprettes en socketforbindelse med administratoren. Leverandøradministratoren oppretter en global tabell for å holde en oversikt over hvor hvilke videoleverandører som er aktive. Den stiller seg så til å vente på neste beskjed fra administratoren. Når en beskjed fra administratoren mottas finner leverandøradministratoren en ubrukt plass i tabellen og starter en leverandørtråd. Ved opprettelse av tråden følger følgende parametere med :

Klientadresse
portnummer
filmnavn
tabellplass

Tabellplassen markeres så som opptatt ved å sette den aktuelle plassen i tabellen til opptatt. Så sjekkes tabellen for avsluttede leveranser. Hvis en videoleverandør har avsluttet, sendes beskjed til administratoren. Til slutt stiller leverandørprosessen seg til å vente på ny beskjed fra administratoren.

6.5.4 Videoleverandør

Videoleverandøren startes fra leverandør administratoren som en egen tråd. Dens oppgave er å hente data fra disk via scheduleren, for så å sende disse til klienten etter klientens ønske.



Figur 6.10 Videoleverandør.

Ved start av en videoleverandør-tråd skjer følgende:

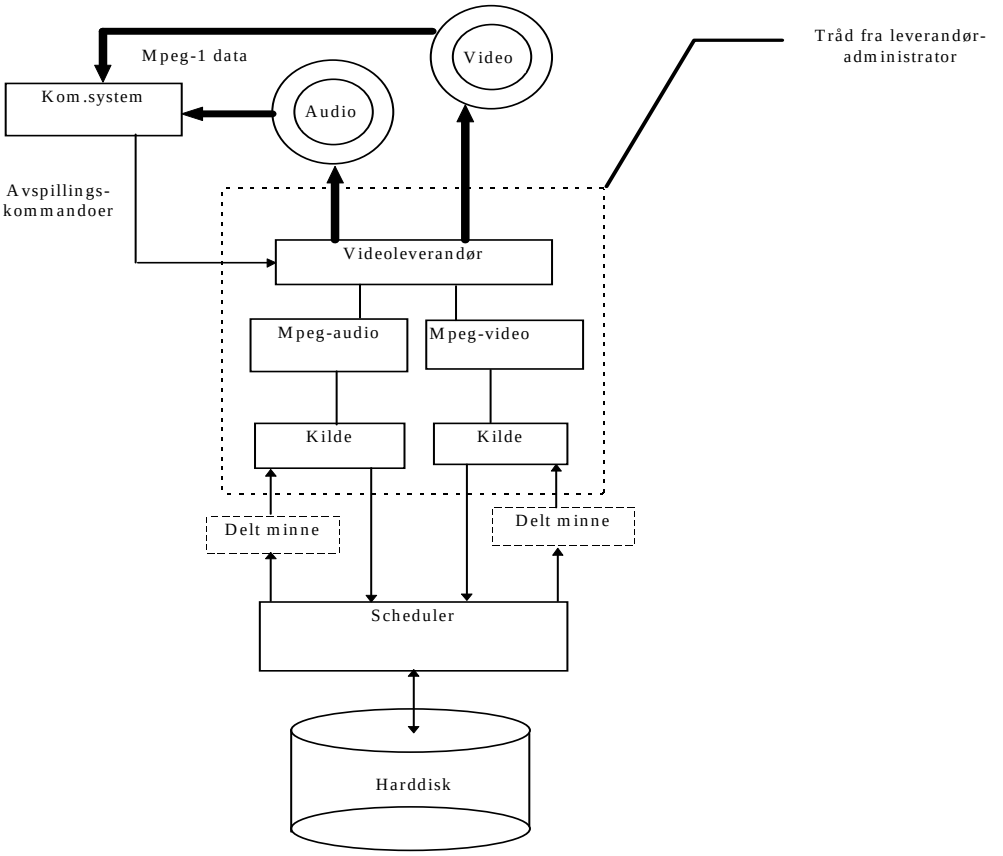
1. Parametere fra leverandøradministratoren mottas.
2. Leverandøren tar kontakt med den ventende klienten på angitt maskinadresse og portnummer.
3. Klienten svarer med å sende to nye portnummer for overføring av henholdsvis MPEG-audio og MPEG-video.

Videoleverandøren åpner angitt film via scheduleren (4).

5. Videoleverandøren leser data fra denne via delt minne.
6. De leste dataene formateres og overføres til klienten.
7. Videoleverandøren for beskjed fra klienten om å avslutte leveringen.
8. Beskjed om avsluttet levering sendes til leverandør administratoren

6.5.4.1 Innvendig oppbygning

Her gir vi en detaljert beskrivelse av innvendig funksjonalitet i videoleverandøren.



Figur 6.11 Oppbygningen av en videoleverandør.

For å gi en fleksibelt konstruksjon består videoleverandøren av flere adskilte nivåer (Figur 6.11) der hvert nivå har spesifikke oppgaver. Siden hvert nivå er uavhengig av andre, vil det være mulig å bytte ut et nivå uten å bytte ut de andre nivåene. Dette er særlig aktuelt for nivået som er MPEG-1 spesifikt, fordi man her kan legge inn kontroll av video med andre formater uten at resten av systemet må forandres.

Videoleverandør

Øverste nivå i videoleverandøren er kalt videoleverandør. På dette nivået vil alt som har med synkronisering, timing og kommunikasjon med klient utføres. Stikkord er mottak av avspillingskommandoer, sending av videodata og regulering av sendehastighet.

Videoleverandøren opprettes fra leverandøradministratoren. Ved opprettelse mottas parametere fra leverandøradministratoren .

Etter opprettelsen opprettes et serverobjekt for kommunikasjonssystemet ved hjelp av de mottatte parametrene (Klientmaskin og portnummer). Så opprettes ringbuffere for audio og video for kommunikasjonssystemet. Deretter kontaktes klienten. Kommunikasjonssystemet tar seg så av opprettelsen av kommunikasjonslinjer for audio og video.

Det mottatte filnavnet leveres til et MPEG-1 videoobjekt og et MPEG-1-audioobjekt. Disse finner rett filnavn for filene som skal benyttes for avspilling av den aktuelle filmen.

Før vi går inn i leveringsløkken, hentes startpekeren for ringbufferne fra MPEG-1-objektene.

Deretter venter leverandøren på en avspillingskommando fra klienten. Når denne er mottatt hentes en tidsreferanse fra operativsystemet og man starter levering av video. Hastigheten på leveringen avhenger av avspillingskommandoen og settes når denne mottas.

Ved levering går leverandøren i en leveringsløkke. Her hentes pekere til rammer i MPEG-1 audio og video til ringbufferne, og det sjekkes om en ny avspillingskommando er mottatt. Deretter måles antall rammer som er levert i forhold til hvor mange som burde vært levert. Ved avvik endres hviletiden mellom hver ramme, for å kompensere for dette avviket. Hviletiden beregnes så ut ifra målt forbrukt tid og avvik i leveringen. Den beregnede hviletiden benyttes så i en hvilekommando.

Kommunikasjonssystemets ringbuffere

For å spare minne- og prosessorkapasitet er ringbufferne til kommunikasjonssystemet og delt minne i videoleverandøren det samme minneområdet. Dette gjør at man slipper å bruke tunge minnekopierings-kommandoer hver gang en ramme skal sendes. Man lever bare opplysninger om hvor rammene ligger i delt minne til kommunikasjonssystemet, så ordner kommunikasjonssystemet resten. Et problem som oppstår ved å gjøre dette, er at man ikke lenger har full kontroll på inneholdet i delt minne. Det vil si, man må forhøre seg med kommunikasjonssystemet om data er send før en skriver over data i delt minne.

MPEG-1 video

På nivået under befinner det MPEG-1 spesifikke nivået. Her styres henting og behandling av data slik at man oppnår de ønskede VCR-funksjonaliteter på MPEG-1 data. Dette nivået er delt i to fordi MPEG-1 videoformatet er mer komplisert en MPEG-1 audio. Det er også hensiktsmessig siden lyd bare leveres når vi har normal avspilling.

Følgende metoder i MPEG-1 video er tilgjengelig for videoleverandøren:

- **GetFrameRate(),** for avlesning av rammerate for filmen for å vite leveringstakt for leveringløkke.
- **BufStart()**, for henting av peker til første dataposisjon i delt minne.
- **SetSpeed(NewSpeed)**, for endring av spolehastighet for levering til videoleverandør.
- **GotoFrame(FrameNo)**, for hopping til bestemte rammer i filmen.
- **NextFrame(FreeArea)**, for henting av neste ramme for levering, og eventuelt innlesning av nye data til delt minne (se delkapittel om bruk av delt minne).

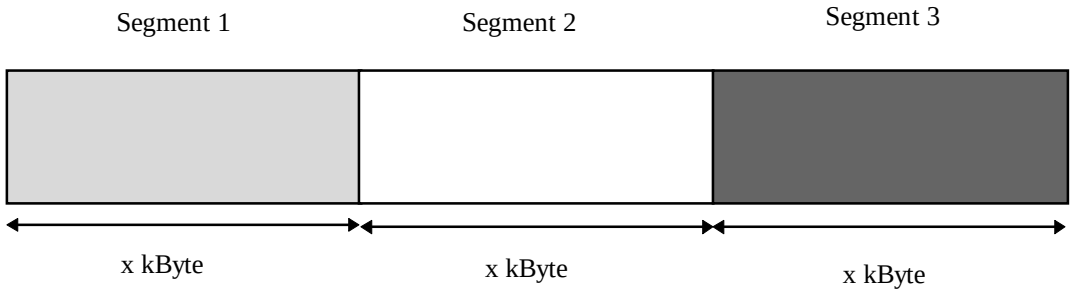
Ved opprettelse av MPEG-1 video skjer følgende:

- Videofilen åpnes via kilde-objektet.
- Eventuell spolefil åpnes via kilde-objektet.
- Indeksfiler for videofil og eventuelt spolefil leses inn i minnet.
- Tidsfrist for levering av data fra scheduler beregnes.
- Første segment av film leses inn i delt minne.
- Rammeraten til filmen hentes fra dette segmentet.

MPEG-1 audio

MPEG-1 audioobjektet er en enkel versjon av MPEG-1 videoobjektet. Den har de samme funksjonene, unntatt de som har med spoling å gjøre. For hopp i lyd benyttes funksjonen GotoFrame. Denne tar inn en presentasjonstidskode, levert fra MPEG-1 video via videotjeneren, for lokalisering av riktig ramme.

Bruk av delt minne



Figur 6.12 Oppbygning av delt minne for video

Delt minne for video er oppbygget av flere innlesningssegmenter som vist i Figur 6.12. Størrelsen på segmentene kan varieres etter ønske. Størrelsen bør velges med hensyn til segmentering på disk slik at man alltid leser hele segmenter fra disk. Disker har en fast segmentstørrelse mellom 1 og 8 kByte.

Data leses inn i minnebufferet fortløpende slik at segment 1 fylles med første innleste dataområde fra disk, mens segment 2 fylles med andre innleste dataområde fra disk osv. Dette sikrer at det alltid blir lest hele segmenter fra disk. Oppdelingen i flere segmenter gjør det mulig å lese data fra disk, samtidig som man behandler tidligere innleste data, man sparer derfor en del ventetid.

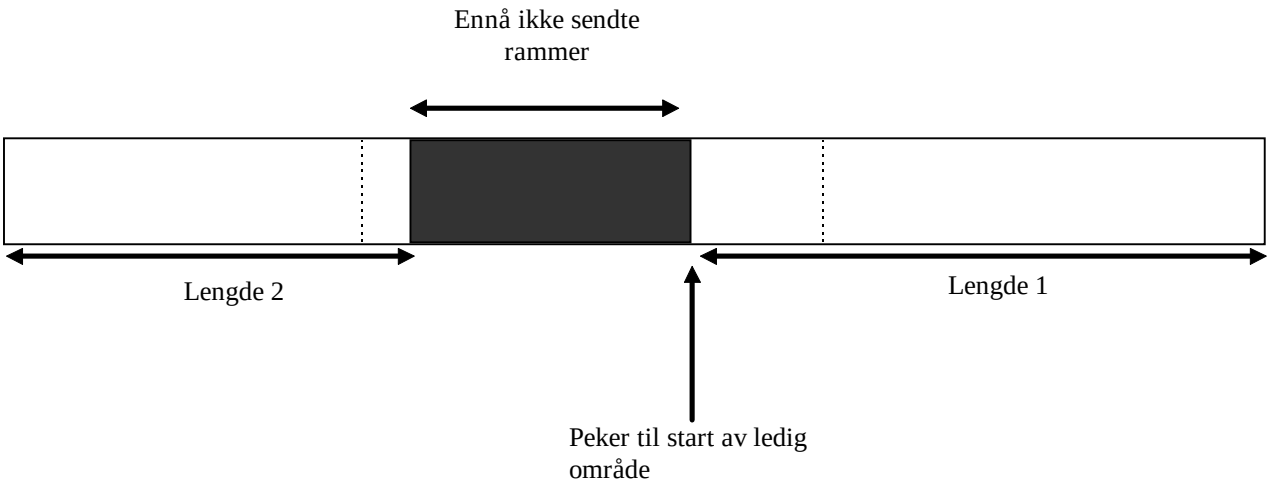
Utskiftning av data i segmenter skjer når dataene som ligger der er sendt til klienten, og man befinner seg på segmentet foran. Det vil si utskiftning av data i segment 1 kan bare skje når aktivt segment er segment 3. Utskiftning av segment 2 kan bare skje når aktivt segment er segment 1 osv. Med aktivt segment menes det segment man i øyeblikket leverer data fra. For å holde orden på hvilket segment som er aktuelt for utskiftning, deklareres det en egen variabel for dette.
For utskiftning hentes opplysninger om rammer er sendt eller ikke hentes fra kommunikasjonssystemets ringbuffer via øverste nivå i videoleverandøren. Opplysninger som hentes har følgende format (se Figur 6.13):

- Peker til start av ledig område
- Lengde på ledig område (Lengde 1 + lengde 2).

Kriterer som må oppfylles ved utskiftning:

- Pekeren til start av ledig område skal ligge innenfor segmentet foran segment som skal utskiftes.
- Avstanden mellom starten av aktivt segment og peker + ledig lengde skal overstige størrelsen til to segmenter.

Når disse kriterer er oppfylt leses data inn i segmentet og variabelen for segment som skal utskiftes oppdateres.



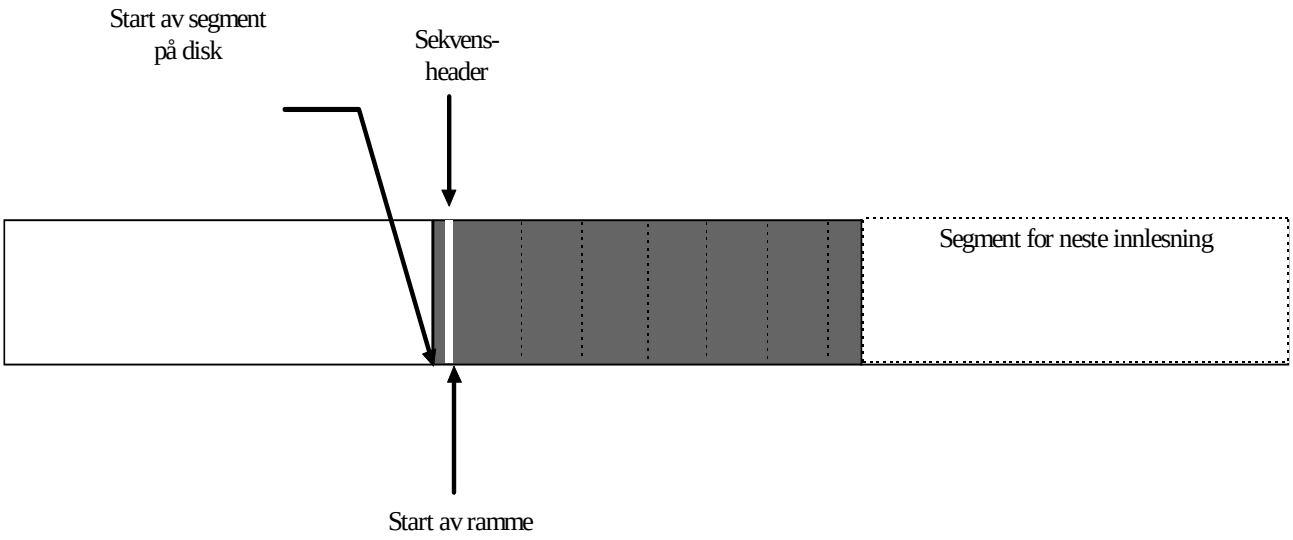
Figur 6.13 Opplysninger fra ringbuffer.

Skifte fra normal avspilling til spolefil for spoling fremover

Ved skifte av avspillingskommando til hurtig spoling fremover, skiftes det til lesing av spolefil.
Utskiftningen gjøres på følgende måte:

- Man finner nærmeste bildegruppe start i fremover retning ved hjelp av indeksfil for normal avspilling.
- Man bruker så indeksfil for spolefil for å finne posisjon for start av bildegruppe i spolefil.
- Så beregnes nærmeste hele disksegment i forhold til denne posisjonen.
- Filposisjon for neste avlesning settes.
- Aktuelt minnesegment for neste innlesning settes.
- Peker til den nye rammen i minnet beregnes.

På grunn av omkoding av spolefil må sekvensheaderen for filen legges inn foran første ramme som skal leveres. Dette i tilfelle man har endret default kvantiseringsmatrise ved omkoding. Når dette gjøres flyttes pekeren for den nye rammen til start av sekvensheader slik at denne sendes med rammen.



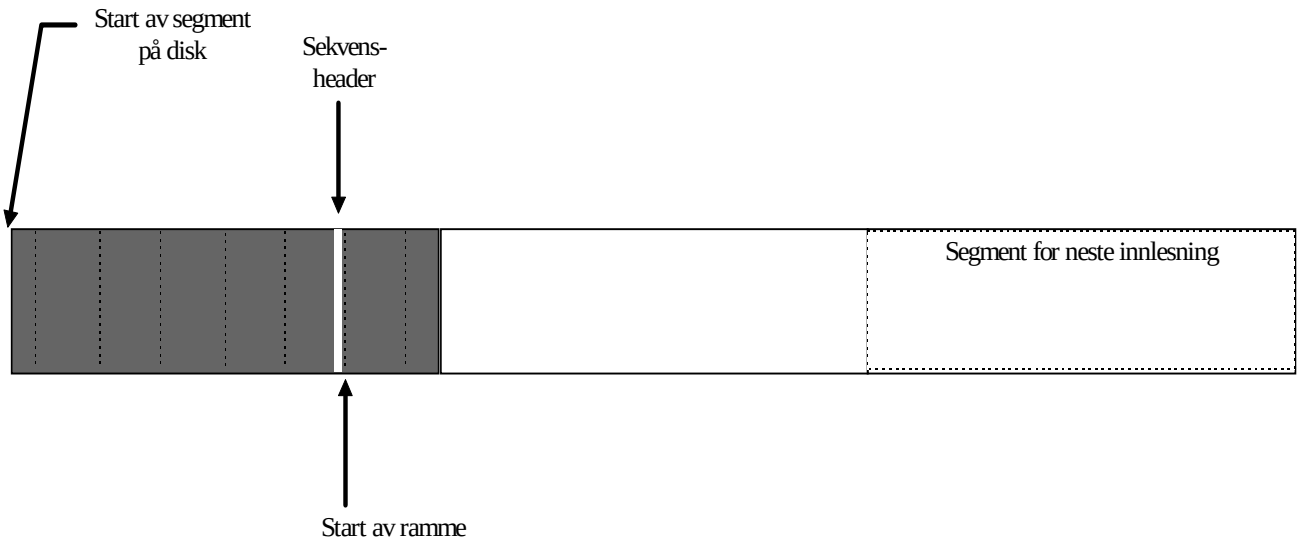
Figur 6.14 Start av spoling med spolefil fremover.

Skifte fra normal avspilling til spolefil for spoling bakover

Ved skifte av avspillingskommando til hurtig spoling bakover. Skiftes det til lesing av spolefil.
Utskiftningen gjøres på følgende måte:

- Man finner nærmeste bildegruppe start i bakover retning.
- Man bruker så indeksfil for spolefil for å finne posisjon for start av bildegruppe i spolefil.
- Så beregnes nærmeste hele disksegment i forhold til posisjonen til slutten av rammen.
- Filposisjon for neste avlesning settes.
- Aktuelt minnesegment for neste innlesning settes.
- Peker til den nye rammen i minnet beregnes.

Også her legges sekvensheaderen for filen inn foran første ramme som skal leveres.Dette betyr at vi skriver over data for neste ramme som skal leveres. Dette løses ved at man henter ut dataene som ligger der sekvenheaderen blir plassert. Man sender så rammen med sekvensheaderen. Så legges de utkopierte dataene tilbake.

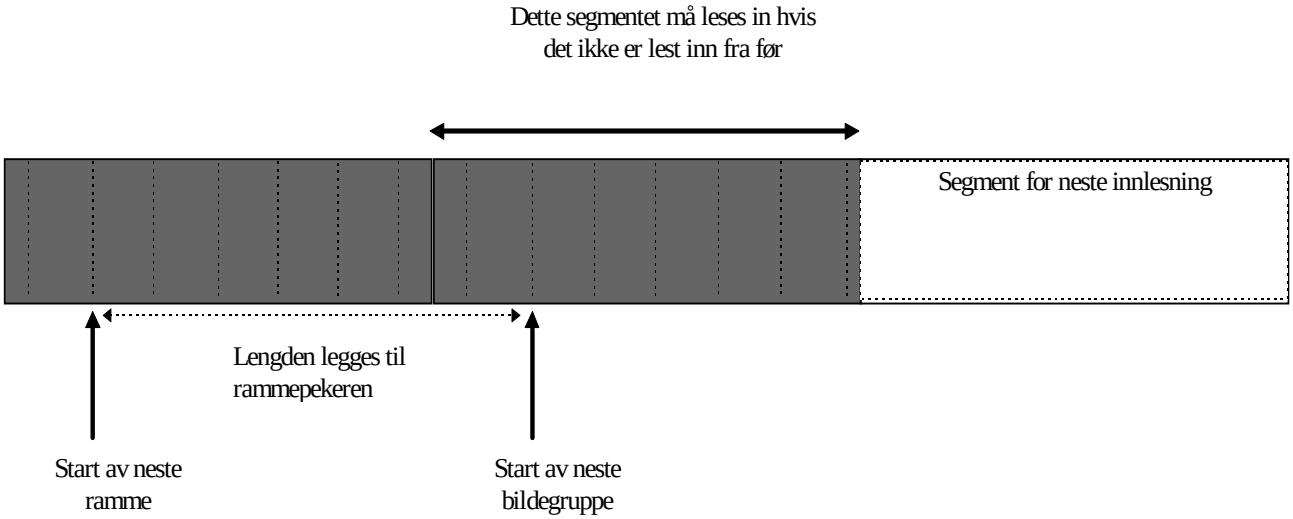


Figur 6.15 Start av spoling med spolefil bakover.

Skifte fra normal avspilling til spoling fremover uten spolefil

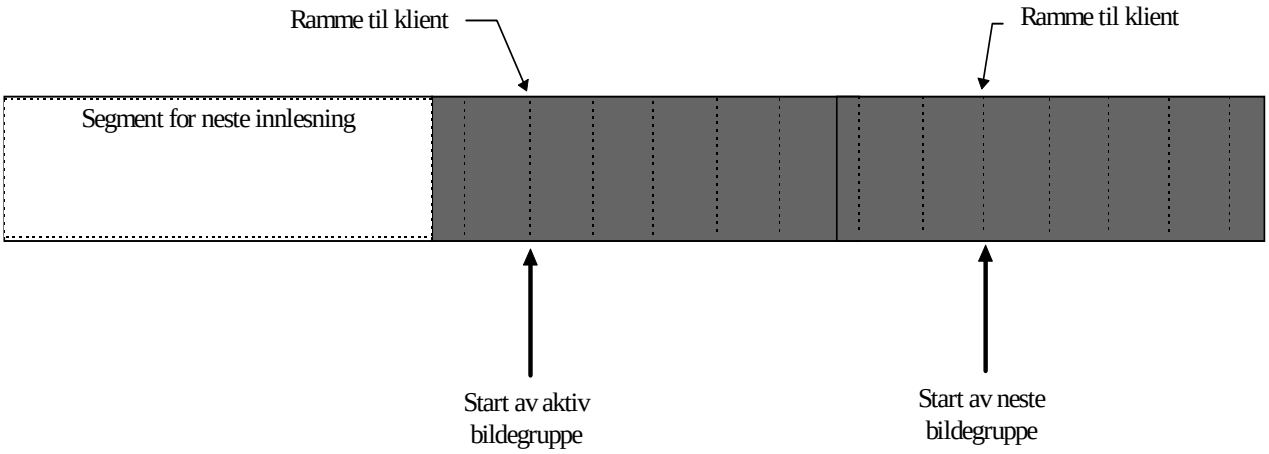
Skiftet til spoling fremover uten spolefil gjøres på følgende måte:

- Man finner nærmeste bildegruppe start i fremover retning.
- Man finner avstanden mellom rammen som rammepekeren i øyeblikket peker på i og
  posisjonen til I-rammen i bildegruppen.
- Man sjekker så om man har krysset over i et nytt segment og om dette segmentet allerede er lest inn.
- Leser inn segmentet vi har havnet på hvis det ikke er innlest fra før.
- Filposisjon for neste avlesning settes.
- Peker til den nye rammen i delt minne settes.
- Aktuelt minnesegment for neste innlesning settes.



Figur 6.16 Start av spoling fremover uten spolefil.

Bevegelse i delt minne ved spoling uten spolefil (I-rammer)

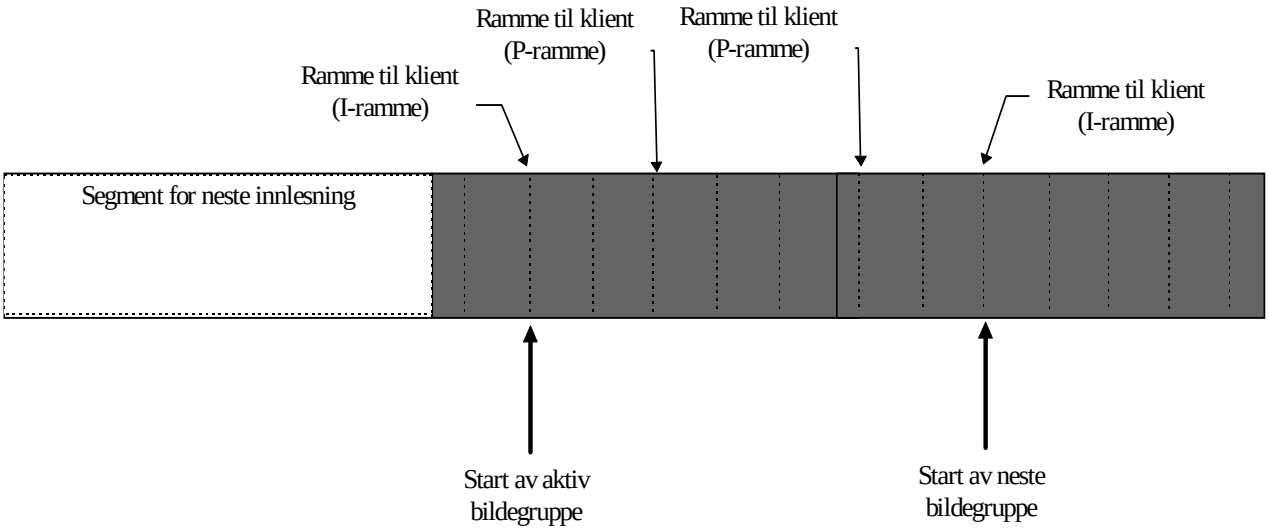


Figur 6.17 Bevegelse i delt minne ved spoling uten spolefil (I-rammer).

For denne bevegelsen benyttes bildegruppetabellen i indeksfilen. Den inneholder de parameterene vi trenger, startposisjon for bildegruppe og rammenummer for startramme i bildegruppen.

Vi vil her oppleve at man må lese inn data i delt minne med hyppigere frekvens enn tidligere. Det skjer fordi mye av de dataene som leses inn, blir hoppet over og aldri brukt (se Figur 6.17). Hvis størrelsen på bildegruppene nærmer seg størrelsen på et segment vil det lønne seg å lese inn hver I-ramme i stedet for alle data. En slik endring vil ikke kreve en ny skiftealgoritme, men det vil kreve endring av måten man leser inn data videre.

Bevegelse delt minne ved spoling uten spolefil (I- og P-rammer)



Figur 6.18 Bevegelse i delt minne ved spoling uten spolefil (I- og P-rammer).

For bevegelsen i delt minne brukes rammetabellen i indeksfilen. Parametere som benyttes er:

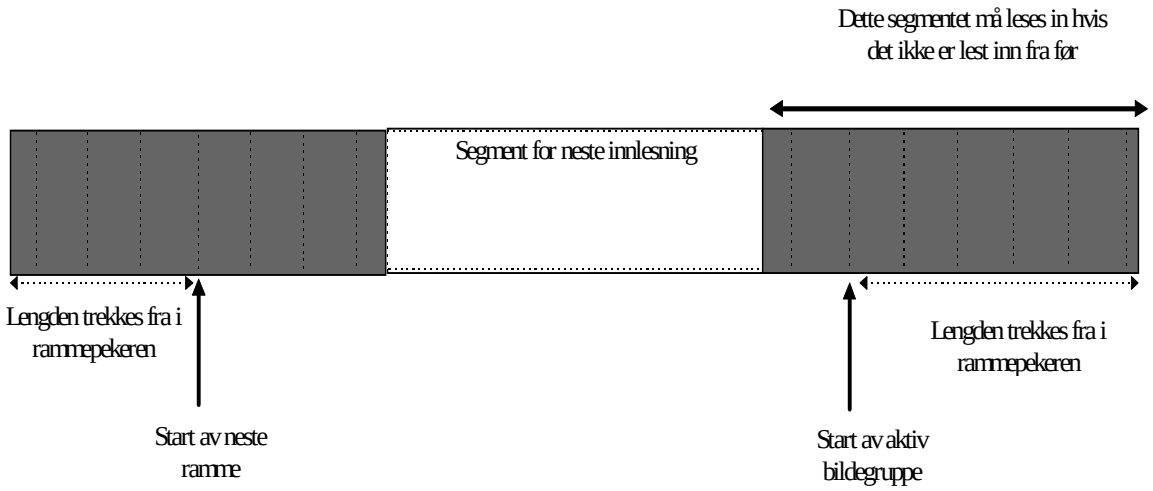
- Rammeposisjon.
- Rammetype.

Vi vil også her oppleve at man må lese inn data i delt minne med hyppigere frekvens enn tidligere. Det er imidlertid ikke like markert som ved spoling bare med i rammer, da det oftest finnes 3-5 P-rammer mellom hver I-ramme. Man bør også poengtere at spoling med I- og P-rammer er ment brukt ved lavere spolehastigheter enn spoling kun med I-rammer.

Skifte fra normal avspilling til spoling bakover uten spolefil

Skiftet til spoling bakover uten spolefil gjøres på følgende måte:

- Man finner nærmeste bildegruppe start i bakover retning.
- Man finner avstanden mellom rammen som rammepekeren i øyeblikket peker på i og
  posisjonen til I-rammen i bildegruppen.
- Man sjekker så om man har krysset over i et nytt segment og om dette segmentet
- allerede er lest inn.
  Leser inn segmentet vi har havnet på hvis det ikke er innlest fra før.
- Filposisjon for neste avlesning settes.
- Peker til den nye rammen i delt minne settes.
- Aktuelt minnesegment for neste innlesning settes.



Figur 6.19 Start av spoling bakover uten spolefil.

Vi vil også her oppleve at man må lese inn data i delt minne med hyppigere frekvens enn tidligere, på samme måte som i spoling fremover uten spolefil.

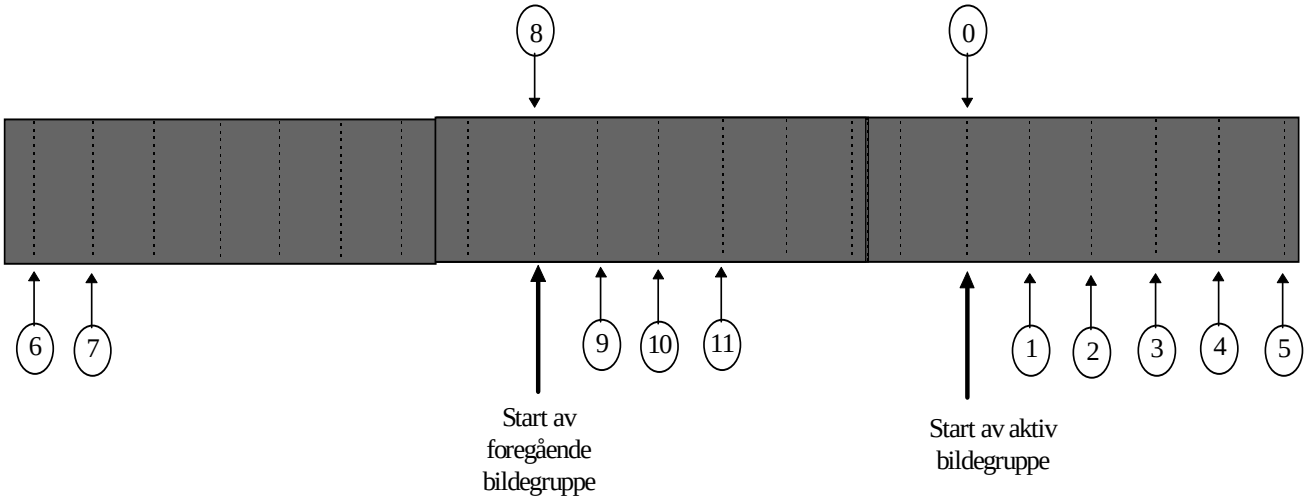
Skifte fra normal avspilling til av revers.

Ved skifte til revers med alle rammer, dvs. bevegelse bakover i hastigheter mellom 0 og 1 der alle bilder vises, må man nummerere om rammene i hver bildegruppe som vist i løsningskapittelet. Jeg velger her å vise hele bildegruppen man i øyeblikket er inne i. Dette vil gjøre at tilbakebevegelsen starter på en annen ramme enn den man avsluttet normal avspilling på. Dette gjøres fordi det ville bli for komplisert å starte tilbakebevegelsen på akkurat denne rammen, og fordi det ikke gir noen nevneverdige ulemper.

Skifte fra normal avspilling til revers er identisk med skifte til spoling bakover uten spolefil med I-rammer, bortsett fra at den temporale referansen til startrammen i bildegruppen må endres slik at visningsrekkefølgen blir endret. Utregningen av ny temporal referanse er gjør slik:

> Maksimal temporal referanse i bilede gruppen (Antall bilder i bildegruppen minus 1) - nåværende temporal referanse = ny temporal referanse.

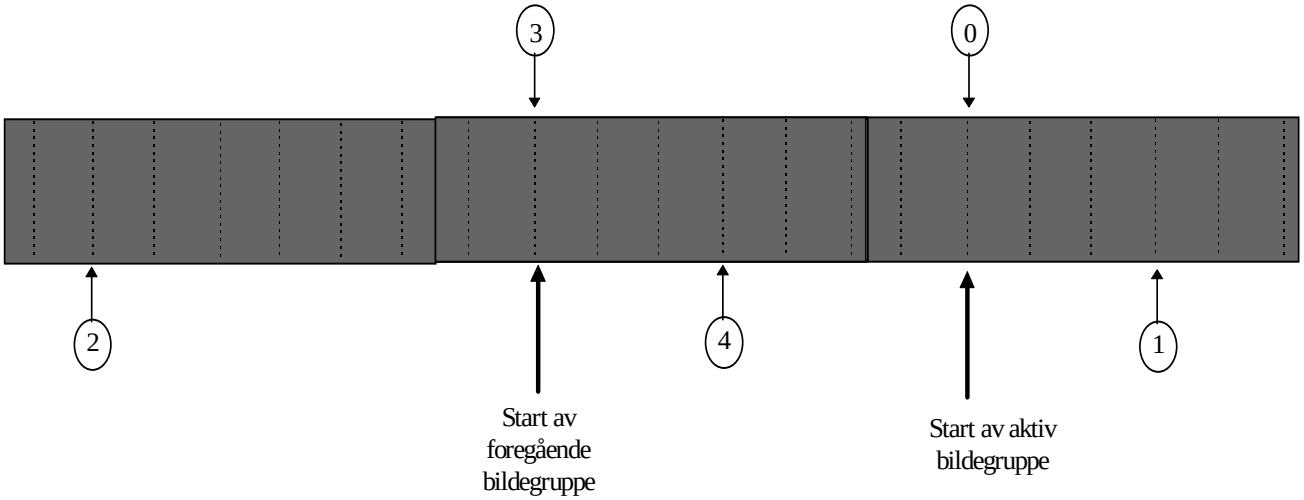
Bevegelse i delt minne ved revers



Figur 6.20 Bevegelse i delt minne ved reverse.

Vi ser av Figur 6.20 hvordan rammer leses ved reverse. Man beveger seg fremover i delt minne ved lesing av hver ramme inne i en bilde gruppe. Hver ramme blir i tillegg omnummerert som beskrevet i skifte til revers. Når man bytter bildegruppe må man bevege seg bakover. Dette hoppet fra siste ramme i en bildegruppe til første ramme i den som ligger bak, kan vise seg å være meget langt. Det lønner seg derfor å tilpasse størrelsen på minnesegmentene etter hvor stort sprang dette kan bli på de filmene som det er aktuelt å levere.

Bevegelse i delt minne ved spoling bakover (I- og P-rammer)



Figur 6.21 Bevegelse i delt minne ved spoling bakover med I- og P-rammer.

Vi ser av Figur 6.20 hvordan rammer leses ved spoling bakover uten spolefil med I- og P-rammer. Man beveger seg på samme måte som ved revers, bortsett fra at man hopper over alle B-rammer. Siden alle B-rammer hoppes over blir det færre bilder per bildegruppe som blir levert. Det gjør at om nummereringen av bildene som leveres må utføres annerledes. Ny formel for utregning av temporal referanse for rammer er:

> Antall I- og P-rammer i bildegruppen -1 for første ramme.
> For de neste rammene reduseres dette tallet med en for hver gang en ny ramme skal leveres.

Kildeobjektet

Kildeobjektets primære oppgave er å abstrahere kommunikasjonen mellom videoleverandøren og scheduleren. Den tar seg av alt som har med opprettelse av kommunikasjonsforbindelser, åpning og lukking av filer, og lesing av data fra filer.

Ved opprettelse av kildeobjektet mottas navn på filer som skal åpnes og antall segmenter delt minne skal bestå av, fra det MPEG-1 spesifikke nivået. Det åpnes en meldingskø mellom kilde og scheduler. Denne benyttes for kommunikasjon fra videoleverandøren til scheduleren. Deretter blir delt minne blir tatt ut, med det antall segmenter som er mottatt. Størrelsen på hvert segment bestemmes av en tidligere definert konstant. Nøkkel til delt minne sendes til scheduleren slik at denne kan koble seg til det.

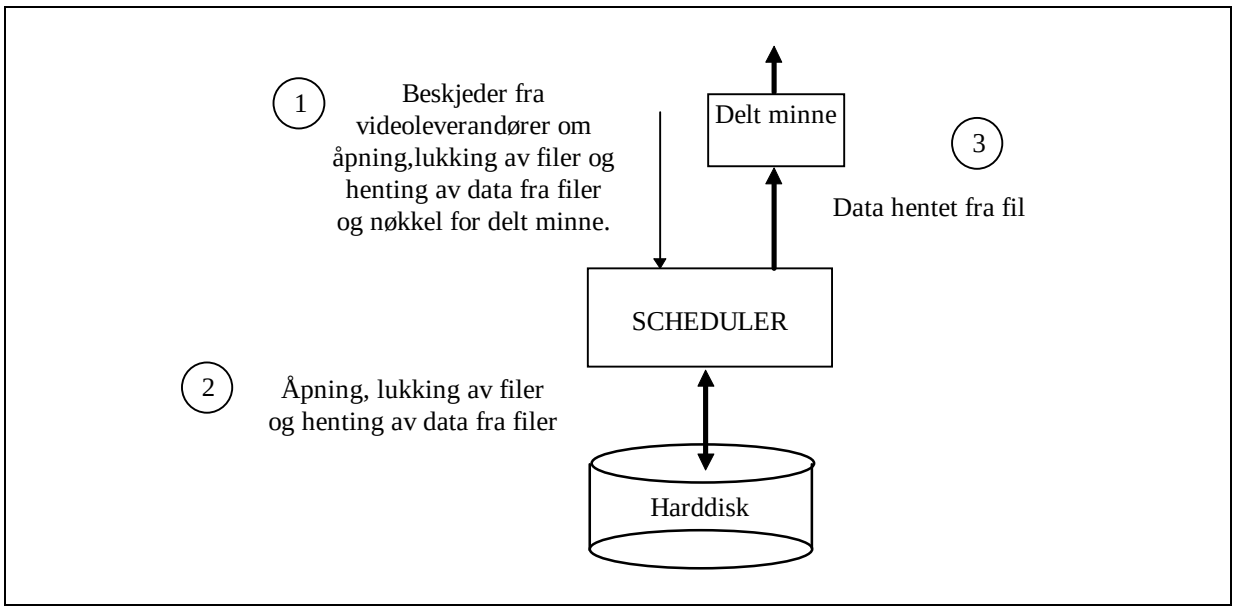
Kilde objekter har følgende funksjoner:

> **OpenFile(filename)**, For åpning av filer.
> **CloseFile()**, For lukking av filer
> **Read(FilePos,Size,Shardmemory_pos, TimeLimit)**, for lesing av data fra fil.
> **ReadWind(FilePos,Size,Shardmemory_pos, TimeLimit)**, for lesing av data fra spolefil.

Når en fil åpnes mottas en filidentifikator for denne filen fra scheduleren. Denne filidentifikatoren benyttes senere ved lesing og lukking av filen. Read og ReadWind har fire parametere. FilePos er leseposisjon for fil, Size er antall bytes som skal leses, Sharedmemory_pos er posisjon for plassering av de innleste data i delt minne og TimeLimit er en tidsfrist for når dataene må ligge i delt minne.

Meldingene som sendes fra de ulike funksjonen er forklart i delkapittelet om scheduleren.

6.5.5 Scheduleren



Figur 6.22 Scheduler.

Kommunikasjon med scheduleren skjer på følgende måte:

1. Beskjed om åpning av fil, luking av fil, eller henting av data fra fil mottas fra videoleverandør. .
2. Åpning av fil, luking av fil eller henting av data fra fil gjøres.
3. Beskjed om åpning av fil, lukking av fil eller henting av data leveres til delt minne.

6.5.5.1 Innvendig oppbygning

Etter oppstart av scheduler opprettes en meldingskø. Denne benyttes for å sende meldinger til scheduleren.

For kommunikasjon via delt minne benyttes første byten i delt minne. Før sending av melding til scheduleren settes denne byten til FALSE fra videoleverandøren. Når data er ferdig innlagt i delt minne fra scheduleren setter den denne byten til TRUE fra scheduleren.

Scheduleren tolker 3 forskjellige meldinger fra videoleverandøren :

1. OPEN filnavn deltminnenøkkel

Denne melding brukes ved åpning av nye filer. Filnavn er navnet på filen som skal åpnes. Delt-minnenøkkel er en unik nøkkel verdi som benyttes ved opprettelse av delt minne. Dette er nøkkelen til det delte minnet som data fra den oppgitte filen skal leveres til. Scheduleren oppretter derfor kontakt med delt minne med den oppgitte nøkkelen. I retur til kommandoen legger scheduleren inn en lokal filident for filen.

2. GET filid filposisjon «antall byte» «delt minne-posisjon» tidsfrist prioritet

Denne meldingen benyttes ved henting av data. Filid er identifikator til filen i scheduleren. Filposisjon er posisjon i byte for hvor dataene skal leses fra i filen. Antall byte er antall byte

som skal leses fra filen. Delt minne posisjon forteller hvor dataene som leses skal plasseres i delt minne. Tidsfristen forteller schedulleren hvor lang tid den har på å hente dataene. Ved behandling av flere «GET» meldinger vil meldingen med kortest tidsfrist behandles først. Prioritet benyttes for å skille mellom henting av data for spoling og henting av data for vanlig avspilling. Henting for vanlig avspilling skal prioriteres foran spolling. Dette betyr at alle «GET» meldinger for avspilling må være ferdigbehandlet før meldinger for spoling behandles. Det gjøres fordi man hos klienten vil mer merke til forsinkelser ved vanlig avspilling.

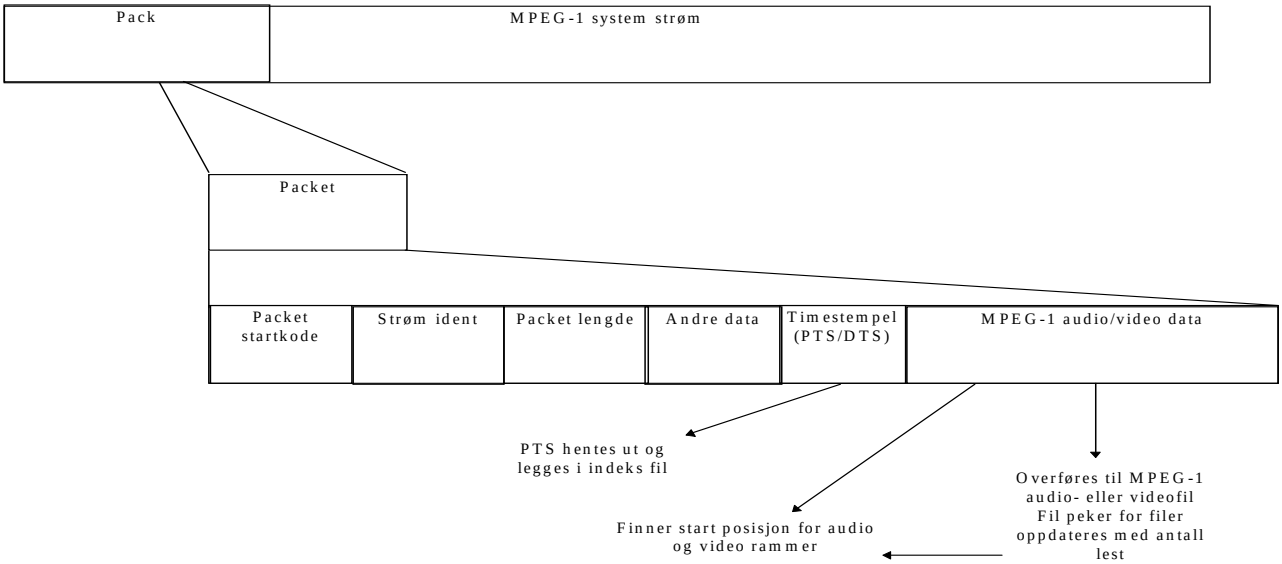
3. CLOSE filid

Benyttes ved avslutning av en leveranse fra videoleverandøren. Filen med filident lik filid lukkes.

Scheduleren er stort sett lik scheduleren som ble brukt i ELVIRA [Langørgen-94].

6.6 Indeksfiler

For å kunne manøvrere i audio og videofilene lages indeksfiler. Disse inneholder opplysninger som gjør det mulig å finne fram til ting som f.eks rammeposisjoner og rammelengder i filene. Filene opprettes ved separering av MPEG-1 systemfil til MPEG-1 audio- og videofil.



Figur 6.23 Henting av data fra systemfil

Separering av data skjer på følgende måte:

- Man finner Start kode for packeter
- Strømidenten forteller om dataene er audio eller video data.
- Man leser packet lengden.
- Man leser en eventuell presentasjonstid.
- Så klippes dataene ut av packeten og lagres i en .mpg fil for video og en .mp2 fil for audio.

Følgende data legges i indeksfilen for video:

Header inneholder følgende data :
- Størrelsen på video filen i byte.
- Antall rammer i filen.
- Antall bildegrupper i filen.

Group inneholder følgende data :
- Start posisjon for bildegruppen.
- Rammenummer for første ramme i bildegruppe

Frames inneholder følgende data:
- Presentasjonstid for ramme.
- Posisjon til ramme i fil.
- Temporal referanse for ramme.
- Ramme type (I, P, B eller D)

Ved å lagre Header fremst i indeksfilen vet man nøyaktig hvor mange rammer og bildegrupper filen inneholder, og det gjør det mulig å hente resten av dataene fra filen.

Indeksfil for audio inneholder følgende data:

Header inneholder følgende data:
- Størrelsen på audio filen i byte.
- Antall audio rammer i filen.

Frame inneholder følgende data:
- Presentasjonstid for ramme
- Startposisjon for ramme

Her legges header informasjonen først i indeksfilen for å vite hvor mange rammeindekser man må lese inn.

I tillegg til dette kommer en egen indeks fil for spolefil. Den vil inneholde følgende data:

- Antall I-rammer i spolefilen
- Start posisjon for rammene.

Også her lagres antall I-rammer i spolefilen først i indeksfilen, slik at man vet hvor mange startposisjoner man må hente ut. For å lage denne indeksfilen må man selvfølgelig lage spolefilen først. Dette gjøres på følgende måte:

1. Man plukker ut alle I-rammer i MPEG-1 videofilen og legger disse i en nye fil.
2. Man dekoder denne filen og koder den på nytt.

7. Implementasjon

Dette kapittelet beskriver hvordan videotjeneren er implementert og hva som er forskjellig/mangler i forhold til konstruksjonen. Det vil først bli gitt en oppsummering av hva som er implementert, deretter gis en oversikt over hvilke verktøy som er brukt for realiseringen, til slutt nevnes problemer med implementasjon og mangler i systemet.

7.1 Hva er implementert?

Med utgangspunkt i konstruksjonen i foregående kapittel er en prototype for videotjeneren implementert. All funksjonalitet som er beskrevet for videoleverandøren er implementert. Det er laget to forskjellige objekter for spoling, et objekt der spoling med spolefil benyttes og et der spoling ved hjelp av I-rammer og I- og P-rammer benyttes. Dette er gjort for å minske ressursbruk ved avgjørelse av hvilken metode som skal brukes. Revers er kun implementert i objektet med spolefil, fordi løsningen med spolefil er den som mest sannsynlig vil bli benyttet i et ferdig system. Administratoren er ikke implementert på samme måte som i konstruksjonen, da databasen for videofilmer ennå ikke er tatt i bruk. Som tidligere nevnt har konstruksjon og implementasjon av klient og kommunikasjonssystem blitt gjort av Norsk Regnesentral og NORUT. Det er opplysninger om disse som ligger til grunn for konstruksjonen jeg har laget for klient og kommunikasjonssystem.

7.2 Brukte verktøy

For implementering av videotjeneren er programmeringsspråket C++ valgt. Jeg hadde ingen tidligere erfaringer med dette språket, men ble anbefalt å bruke dette på grunn av videotjenerens naturlige oppdeling i objekter. For kompilering ble gcc benyttet fordi det er den mest brukte C++ kompilatoren for ved IDT.

For debugging benyttet jeg Purify, og det viste seg som et nyttig verktøy. Purify sjekker alle minneaksesser i programmet og gjør det derfor mulig å oppdage feil som ellers ville være vanskelig å finne. Et lite problem var at den fant flere mindre feil i ulike C++ biblioteker, feil som ikke har vært grove nok til at man har sett seg bryderiet verdt å rette dem. Det gjorde at fokus ble noe borte fra de feil som var i egne programmer. Med all testing av minneaksesser vil programmer bli merkbart tregere ved bruk av Purify. Det har allikevel vært et positivt bekjentskap, og Purify er et verktøy som kan anbefales.

For avspilling av MPEG-1 filmer er det brukt tre forskjellige avspillere:

- LAVA-spilleren, som er bygget på grunnlag av Berkeley spilleren (UNIX).
- mpeg_play, som er en MPEG-1-videospiller som kun spiller video (ikke audio) (UNIX).
- Mpegplay, som er en MPEG-1-videospiller under Windows NT.

Grunnen til flere spillere er brukt er at de har forskjellige styrker og svakheter som ble utnyttet. LAVA-spilleren var den eneste som klarte å spillle systemkodet MPEG-1. Den var derimot meget følsom for feil i MPEG-1-video og gikk derfor ofte ned uten å gi noen fornuftig

feilmelding. Den hadde også en ulempe ved at den leste inn hele bildegrupper før de ble vist. En annen feil ved denne var at den ikke var programmert til å skjønne sekvensheadere inne i MPEG-1-video ved f.eks. bytte fra vanlig fil til spolefil. I henhold til MPEG-1 standarden[MPEG-92] er dette lovlig.

Mpeg_play under UNIX ble brukt fordi den var raskere enn LAVA-spilleren under visning. I tillegg dekoder og viser den en og en ramme i stedet for hele bildegrupper.

Mpeg_play under Windows NT ble brukt fordi den var mer tolerant for feil i MPEG-1-video. Den fortsatte å dekode selv om det var feil på en ramme. Feilen dukket kun opp som forstyrrelser i det ene bildet. Dessverre var versjonen som ble brukt en shareware versjon, som ikke tillot bruk av MPEG-1-filer større enn en MByte.

For dekoding av MPEG-1-video til ppm-bilder ble «The MPEG Library» benyttet [Ward-95]. Dette er et bibliotek som inneholder ulike funksjoner for dekoding og visning av MPEG-1 video. Biblioteket er basert på kode utviklet ved Universitet i Berkeley.

For koding av MPEG-1-video ble «mpeg_encode» brukt. Dette er en software koder utviklet ved Universitet i Berkeley. Den tar inn ppm-filer eller YUV-filer, og setter disse sammen til MPEG-1 video. For å styre kodingen benyttes en tekstfil der parametere som bildekvalitet, kodingsmetode og bilderekkefølge bestemmes.

7.3 Programstruktur

Programmene følger samme struktur som i konstruksjonen. En liten endring er gjort i MPEG-1 objektene der henting av posisjon, lengde og tidskode for en ramme er lagt i en funksjon og beregning av ny posisjon og eventuell henting av data fra disk er lagt inn i en annen funksjon. Dette er gjort fordi jeg følte at det ga en mer naturlig struktur i programmet. Det har ingen praktisk betydning for funksjonaliteten.

7.4 Problemer under implementasjonen

Det største problemet under implementasjonen var kommunikasjonssystemet. Det hadde ennå ikke vært utprøvd på IDT's maskiner før det skulle bli tatt i bruk sammen med videotjeneren. Konstruksjonen av kommunikasjonssystemet var tilpasset et annet prosjekt Norsk Regnesentral holdt på med. Det omhandlet sanntidssending av Motion-JPEG-kodede bilder fra et overvåkningskamera i en bank, til f.eks. nærmeste politistasjon. Dette betyr at funksjoner for styring av kamera og for vanlig avspilling var de som var utprøvd. Problemer som at pakker ikke ble sendt, tråder i klient og server hang seg osv., gjorde det etterhvert umulig å benytte kommunikasjonssystemet, slik det var implementert av Norsk Regnesentral. Vi fikk allikevel testet oppkobling av systemet slik at vi så at det ville fungere med nye versjoner av kommunikasjonssystemet.

Et annet problem var at andre komponenter som skulle brukes ikke var ferdig implementert eller ferdig uttestet. Det gjaldt særlig videoavspillieren, men også databasen for lagring av filmopplysninger. Dette medførte at det ikke var mulig å se hvilke hensyn videotjeneren må ta til ting som skjer i klient ved dekoding og visning av video.

7.5 Uttesting

Programmet er implementert og uttestet under SunOS 5.5 (Solarris 2.5) på en Sparc LX maskin.

7.6 Modul beskrivelse

I dette delkapitelet beskrives modulene som programmet består av. De fleste modulene består av to filer, en headerfil og en kode fil. Beskrivelsen vil starte med klienten, så videre til administrator, leverandøradministrator, videoleverandør, MPEG-1 audio og video, kildeobjektet og til slutt scheduleren.
Programkode for moduler i videotjeneren finnes i vedlegg 2.

7.6.1 MpegElviraClient

Denne modulen skal representere klienten. Den benytter et klientobjekt fra kommunikasjonssystemet for kommunikasjon med administratorprosessen. Resten av programmet leser fra ringbufferne, og legger audio- og videodata på fil, med filnavn «audiofile.mp2» og «videofile.mpg». Den er implementert av Norsk Regnesentral og kun ment for uttesting av kommunikasjonssystemet. Man har ved IDT mottatt en klient med innebygget videoavspiller, men man har ennå ikke fått den til å virke.

Den startes opp på følgende måte:

elvira_client elvira://maskinadresse:portnummer/filmnavn

Maskinadresse er adressen til maskinen der videotjenerens administratorprosess er startet og portnummer er nummer på porten administratorenprosessen lytter på for mottak av leveranseønsker.

7.6.2 Administrator

Denne modulen representerer administratoren. Den benytter et tjenerobjekt fra kommunikasjonssystemet for mottak av leveranseønsker fra klienten. Ved mottak av leveranse-ønske sjekkes filmnavn mot en angitt filmkatalog. Hvis filmen finnes, sendes en beskjed til en leverandøradministratorprosess om start av leveranse.

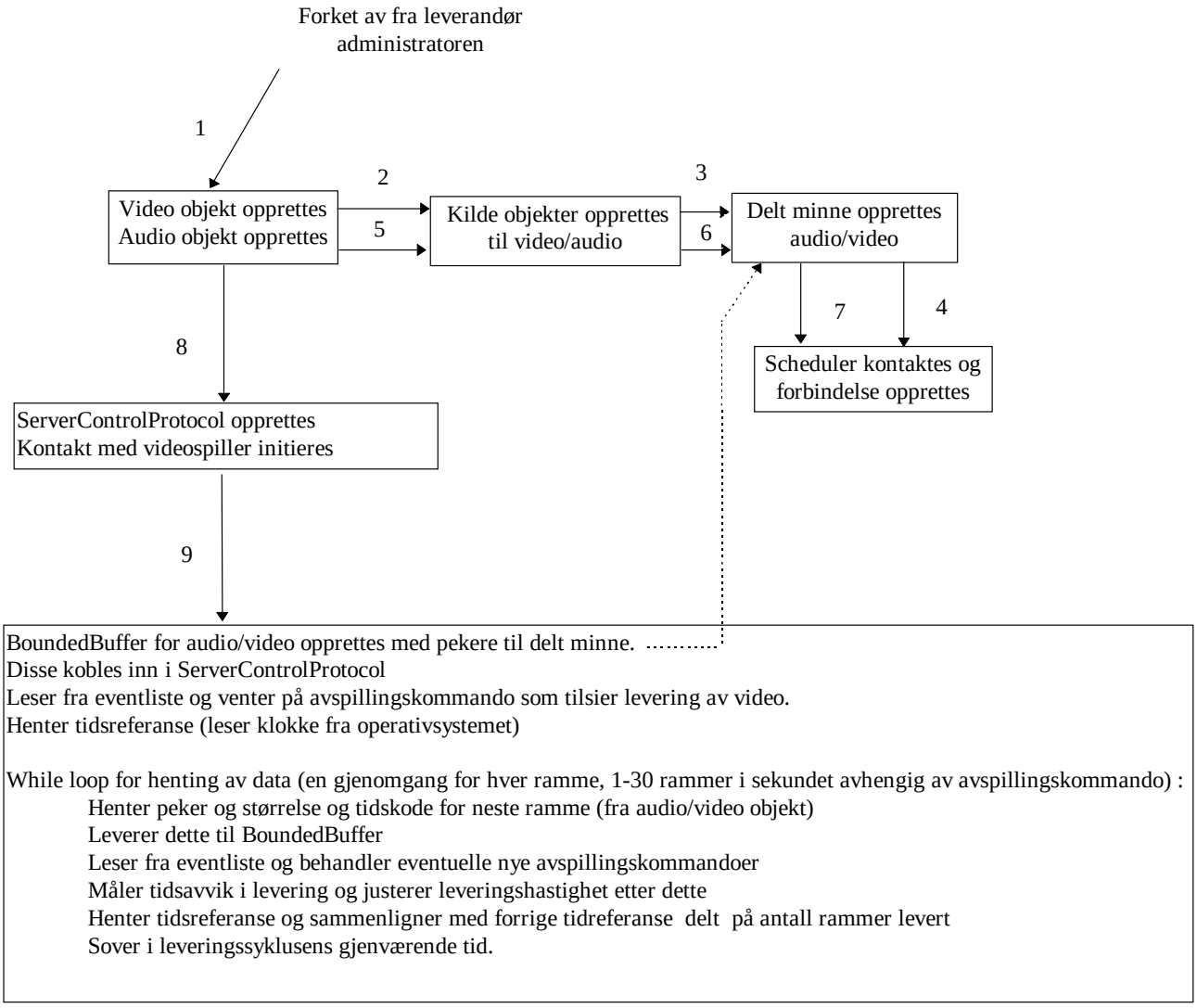
Administratoren startes på følgende måte:

administrator «katalog der filmene befinner seg» klientportnummer leverandørportnummer

Der «klientportnummer» er portnummer for mottak av leveranseønsker fra klienten og «leverandørportnummer» er portnummer for kommunikasjon med leverandøradministratoren.

7.6.3 leverandør

Leverandør modulen består både leverandøradministratoren og øverste nivå i videoleverandøren. Pseudokode for øverste nivå i videoleverandøren er vist i Figur 7.1. Ved start av en leveranse opprettes en tråd i leverandøren av leverandøradministratoren. Denne tråden representerer videoleverandøren. Trådene som brukes er av typen posixtråder.



Figur 7.1 Pseudokode for videoleverandør tråd.

Leverandøradministratoren startes på følgende måte:

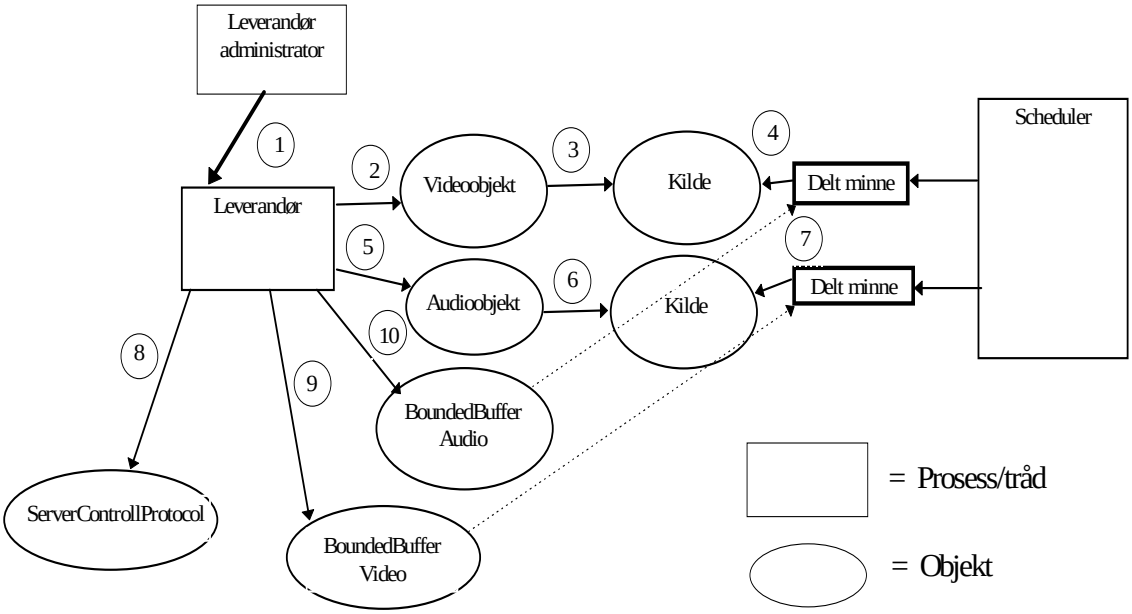
lever administratorportnummer

For kommunikasjon med klienten benyttes følgende objekter:

- ServerControlProtocol
- Boundedbuffer

ServerControlProtocol tar seg av all kommunikasjon over nett. Den mottar og sender events, som tydes hos mottaker i en switch-settning, der de ulike typene events mottakeren skjønner er representert. Boundedbuffer er et minnebufferet for overføring av audio eller video.

Figur 7.2 viser hvordan objekter blir opprettet i videoleverandøren, der punkt 1 til 7 er videoleverandørobjekter, 8 til 11 er objekter tilhørende kommunikasjonssystemet.



Figur 7.2 Objekter som opprettes i leverandørprosessen/tråden.

7.6.4 Mpeg-audio

Denne modulen henter en og en audioramme fra delt minne ved hjelp av indeksfil, og lever disse til videoleverandøren. Den består av følgende instanser:

- **LoadDir**(), for henting av data fra indeksfiler.
- **NextFrame**(), for henting av posisjon, lengde og tidskode av nest ramme som skal leveres.
- **GetNext**(DataFrame df_free), for beregning av posisjon for neste ramme og eventuell innlesning av nye data til delt minne.
- **BufStart**(), for overføring av startpeker til delt minne.
- **GotoFrame**(ulong timestamp), for resynkronisering av lyd og bilde.

7.6.5 Mpeg-video

Mpeg-video er bygget opp på samme måte som mpeg-audio. Den har i tillegg disse instanser:

- **SetSpeed**(short new_speed), for valg av spolehastighet.
- **GetFramRate**(), for henting av rammerate for den spesielle filmen.

Det er laget to forskjellig versjoner av mpeg-video en versjon for spoling med spolefil, og en versjon for spoling med I-rammer.

7.6.6 Kilde

Kilde er objektet som har til oppgave å kommuniserer med scheduleren. Den har følgende instanser:

- **OpenFile**(char *name), for sending av beskjed om åpning av fil til scheduleren.

- **Read**(uint size, ulong position, uint shm_pos, float lim), for sending av beskjed til scheduleren om henting av data fra avspillingsfil og plassering av disse i delt minne.
- **ReadWind**(uint size, ulong position, uint shm_pos, float lim), for sending av beskjed til scheduleren om henting av data fra spolefil og plassering av disse i delt minne.
- **CloseFile**(), for sending av beskjed om lukking av fil til scheduler

7.6.7 Scheduler

Scheduleren adminstrerer data fra disk til hver enkelt klient-prosess i serveren. Den henter data fra disk og legger disse for videre lesing i delt minne. Prioritering av henting skjer ved oppgivelse av tidsfrist for henting. Denne benytter scheduleren til å bestemme rekkefølgen henteønsker skal tilfredsstilles i. Kommunikasjonen med klient-prosessen foregår på to måter. Klient-prosessene kommuniserer med scheduleren v.h.a. messagepasing der man oppgir en nøkkel som forteller hvor meldingen skal hen. Denne nøkkelen er hardkodet i filen sglobal.h. Scheduleren leverer de ønskede data via shared memory som opprettes av klient-prosessen. Identen til sharedmemory overføres ved messagepasing.

Scheduleren er basert på scheduleren benyttet i ELVIRA [Langørgen-94].

7.7 Implementasjon av indeksfiler

For implementasjon av ble to forskjellige programmer benyttet. Kildekoden for disse programmene finnes i vedlegg 3

7.7.1 Separerer

Separerer benyttest til å splitte en MPEG-1-systemfil til en MPEG-1-audiofil, en MPEG-1 videofil, en indeksfil til audio og en indeksfil for video. Eksempel:

Inn:	olav.mps	(MPEG-1 systemfil)
Ut:	olav.mpg	(MPEG-1 videofil)
	olav.mp2	(MPEG-1 audiofil)
	olav.mpv	(Indeksfil for video)
	olav.mpa	(Indeksfil for video)

7.7.2 Mpwmake

Programmet mpwmake lager indexfil til spolefile. Den tar inn en fil av typen mpg og gir ut en indeksfil av typen mpw.

7.8 Implementasjon av spolefil

For å lage en spolefil med I-rammer fra videofilen, ble tre forskjellige programmer benyttet. Disse presenteres her. Kilde koden for programmene nevnt under ligger i vedlegg 4.

7.8.1 I_frame

I_frame er et program som tar inn MPEG-1-videodata og plukker ut alle I-rammene i denne og lagrer disse i en ny fil.

7.8.2 Mpegtoppm

For dekoding av MPEG-1-videoen ble mpegtoppm laget. Mpegtoppm er et program som tar inn en MPEG-1-videofil, dekoder bildene i den og legger de ut i separate filer som i ppm-format. Programmet bruker rutiner hentet fra biblioteket mpeg_lib.

7.8.3 encodeparam

For koding av bildene tilbake til MPEG-1-video, ble mpeg_encode benyttet. Den krever en tekstfil med kodeparametere for å fungere. Et eksempel på kodefilen som ble brukt fremstilles her.

Encodeparam.txt:

```
PATTERN I   #Ramme mønster for hver bildegruppe
# Katalog der den ferdige spolefilen plasseres
OUTPUT /home/stud/b/rogerko/arbeid/diplom/program/test/Spolefil.mpg
# Katalog der rammene som skal kodes ligger
INPUT_DIR /home/stud/b/rogerko/arbeid/diplom/program/test/convert/mpeg_lib/extras
# Her angis filnavn for rammer
INPUT
outfilm*.ppm [1001-1130+1]
END_INPUT
INPUT_CONVERT *          # Her kan man angi preprosseringskommandoer for
rammene
BASE_FILE_FORMAT PPM     # Filformat for rammer
GOP_SIZE 1               # Størrelsen på bildegrupper
SLICES_PER_FRAME 256     # Skiver i hver ramme
PIXEL FULL               # Nøyaktighet for makroblokk vektorer
RANGE 2                  # Maksimal lengde for vektor
PSEARCH_ALG EXHAUSTIVE   # Kodemetode for P-rammer
BSEARCH_ALG EXHAUSTIVE   # Kodemetode for I-rammer
IQSCALE 30               # Koprimeringsfaktor for I-rammer
PQSCALE 50               # Koprimeringsfaktor for P-rammer
BQSCALE 50               # Koprimeringsfaktor for B-rammer

REFERENCE_FRAME DECODED# Referanse ramme for koding skal være dekodet
```

8. Målinger og resultater

Hensikten med denne oppgaven var få til VCR-funksjonalitet for MPEG-1-video. Et delmål må da bli at det ikke koster for mye å få til dette, det være seg i forbruk av disk, prosessor, minne eller nettverksressurser. Det vil her presenteres en del måleresultater med henblikk på dette. Siden to forskjellige metoder for spoling er implementert er det naturlig å sammenligne disse ved hjelp av de foran stående kriteriene. Særlig forbruk av disk og nettverkskapasitet er kritisk i en videotjener. Det vil derfor bli lagt vekt på dette i målingene.

8.1 Utgangspunkt for målinger

Vi har valgt å foreta målinger på overføringen av to forskjellige filmer. Begge har lyd kodet i MPEG-1-audio lag 2. Den ene har en bitrate på 300 kbit/s, der 236 kbit/s er bilde mens 64 kbit/s er lyd. Den inneholder 1953 rammer og 130 bildegrupper. Rammeraten er på 25 rammer/s, følgelig er lengden av filmen ca 1 minutt og 20 sekunder. Filstørrelsen for bilder er på 2,3 MByte for video for avspilling, 418 kByte for spolefil og 625 kByte for lyd.

Den andre filmen har en bitrate på 1,36 Mbit/s, der 1,16 Mbit/s er bilde og 200 kbit/s er lyd. Den inneholder 6628 rammer og 415 bildegrupper. Rammeraten er på 25 rammer/s, følgelig er lengden av filmen ca 4 minutter og 20 sekunder. Filstørrelsen for bilder er på 38,4 MByte for video for avspilling, 2,5 MByte for spolefil og 6,4 MByte for lyd.

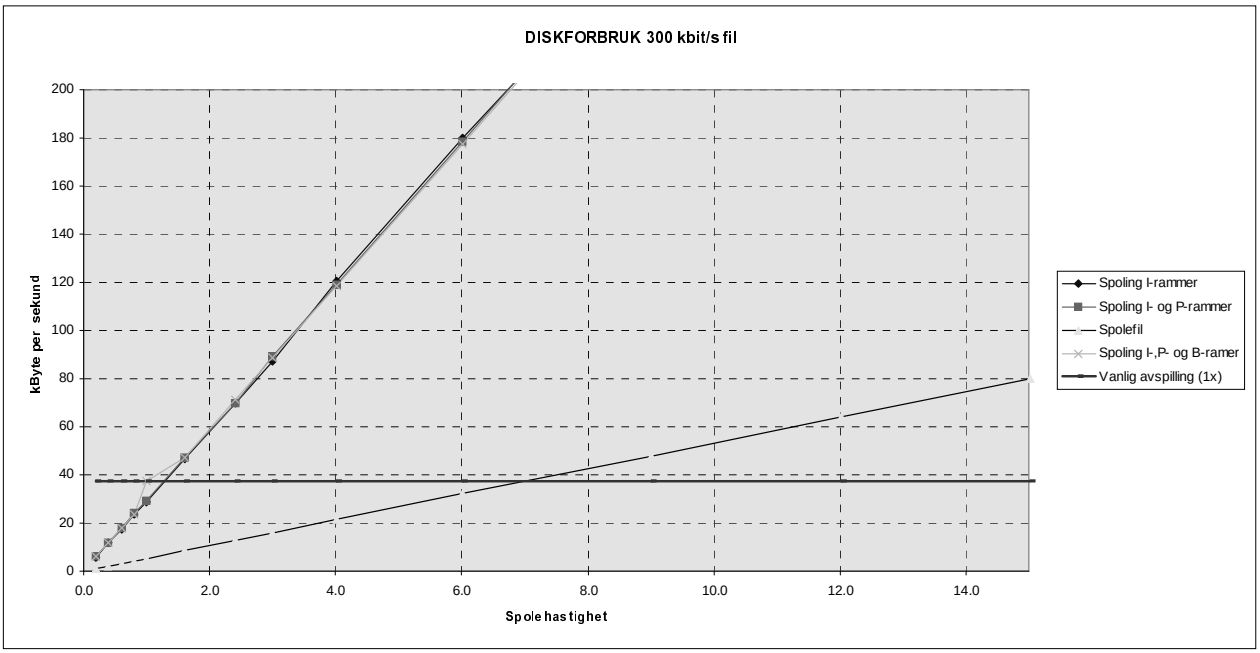
Det vil bli målt og sammenlignet antall byte hentet fra disk per sekund ved ulike hastigheter, og antall byte sendt ut på nettet per sekund. Dette sammenlignes med antall byte hentet fra disk per sekund og antall byte levert ut på nettet per sekund ved vanlig avspilling. Målet er å ikke overstige de ratene som måles for normal avspilling.

Det vil i tillegg bli gjort en kvalitetsvurdering av det som blir levert ut ifra rammehastighet og bildekvalitet i forhold til vanlig avspilling, dette for å vise at man taper kvalitet ved å knipe inn på diskbelastning og nettverksforbruk.

8.2 Spoling i fil kodet med bitrate 300 kbit/s

Her presenteres måleresultater for lesing fra disk og sending av data over nett, for 300 kbit/s fil. Dataene presenteres i byte, noe som gir en byterate på 37,5 kByte/s for en bitrate på 300 kbit/s.

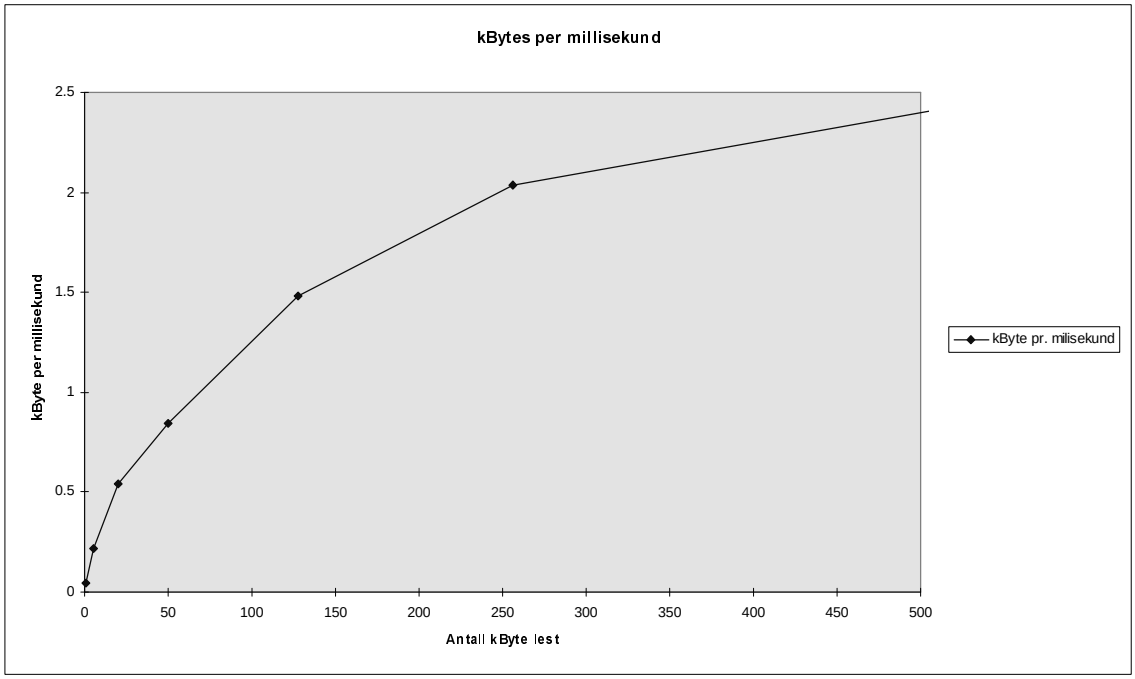
8.2.1 Diskforbruk



Figur 8.1 Diskforbruk ved forskjellige spolemetoder 300 kbit/s fil.

Figur 8.1 viser hvordan antall byte lest per sekund øker med økende spolehastighet. For tre av spolemetodene er lese mengden nøyaktig den samme uansett spolehastighet. Det er bare spoling med I-, P- og B-rammer som øker litt ekstra ved vanlig avspillingshastighet. Det er fordi lesing av audio er lagt inn i tillegg her. Det er bare spoling ved hjelp av spolefil som skiller seg positivt ut. Spolefilen er ikke så liten som ønsket på grunn av begrensninger nedad i bildekvalitet ved MPEG-1 komprimering. Størrelsen på rammene i spolefilen er ca 2 ganger større enn ønskelig. Selv med denne doblingen av rammestørrelsen, vil vi spare en masse lesninger fra disk ved bruk av spolefil. Hvis vi sammenligner verdiene ved 6 ganger spoling, ser vi at antall byte som leses ved bruk av spolefil er 32 kByte/s, mens man ved de andre metodene leser 180 kbyte/s. Dette er ca 5.6 ganger så mange lesninger i snitt. Dette betyr følgende, hvis det er disken som begrenser kapasiteten til videotjeneren, så kan man levere 5.6 ganger flere videostrømmer til klienter. Dette viser klart fordelen med å bruke en spolefil. For å oppnå målet med å ikke overstige belastningen på disk mer enn under vanlig avspilling, vil vi få en maksimal spolehastighet på 7 ganger ved spolefil, mens vi vil ikke få til spoling i det hele tatt ved de andre metoder.

Vi kunne argumenter med at ved f.eks. spoling med I-rammer trenger man bare å lese inn I-rammene og ikke de andre rammene slik det blir gjort nå. Da ville antall byte lest falle med 3 til 4 ganger. Dette ville allikevel ikke lønne seg siden tiden en disk bruker på å lese et vist antall byte ikke er proposjonal med antall bytes som skal leses. Vi har gjort noen målinger for å illustrere dette. I figuren under ble tidsforbruket for lesing av ulike data mengder fra disk målt. Hver måling er et gjennomsnitt av 50 måleverdier slik at tilfeldigheter i diskposisjon ikke skulle være så avgjørende for resultatet som de ellers ville bil. Det er også tatt hensyn til buffering i diskcache og minne, på den måten at cachen/minnet ble tømt/overskrevet etter hver måling. Målingen er gjort på en Sparc LX maskin med en Seagate Barracuda harddisk.

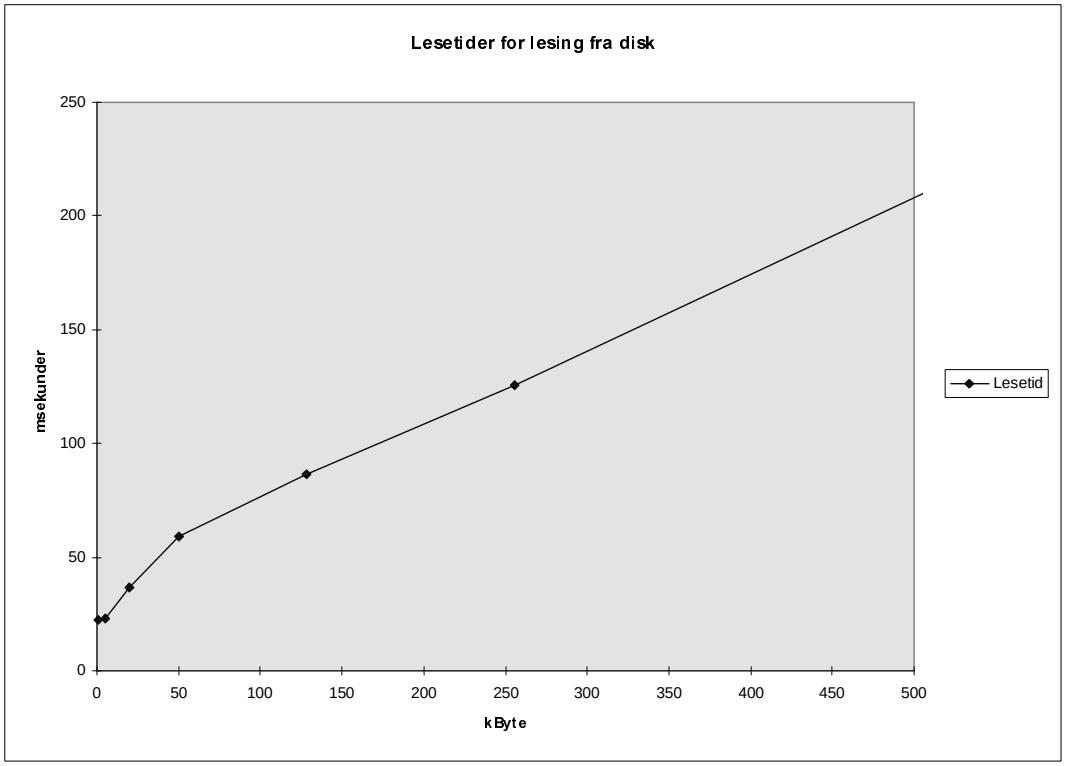


Figur 8.2 Antall kBytes/s lest fra disk ved forskjellige lesemengder.

Målingen (Figur 8.2)viser at ved innlesning av 128 kByte pr lesning, slik som er gjort i implementasjonen, tar hver lesning i snitt 86 ms, mens man ved 5 kByte, som er gjennomsnittsstørrelsen på en I-ramme for denne filen, bruker 23 ms for en lesning. Bildegruppestørrelsen i filen er i gjennomsnitt ca 20 kByte. Det betyr at man leser inn 6.4 I-rammer per lesning i snitt. For lesing av 6.4 enkeltstående I-rammer trenger man dermed:

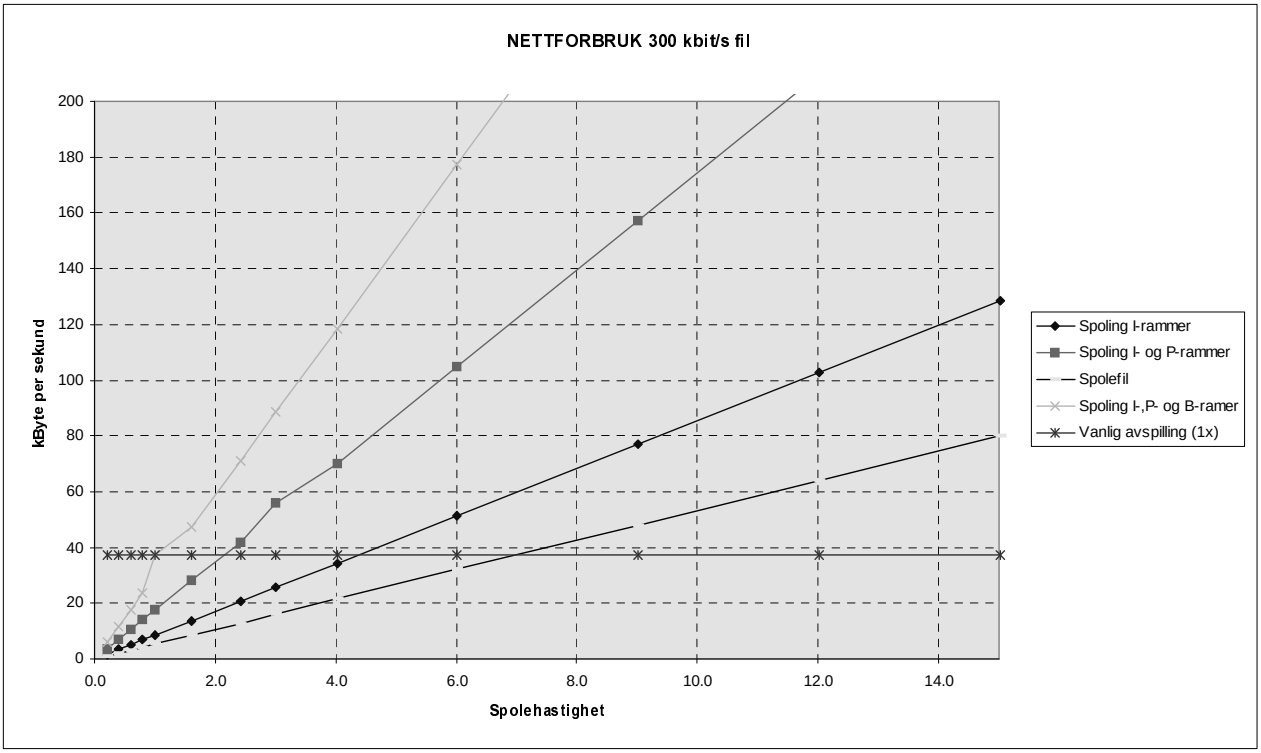
6,4 I-rammer*23ms/ramme = 147.2 ms.

Dette er nesten en dobling av lesetid i forhold til at man leser alle dataene. I tillegg kommer et høyere prosessorforbruk på grunn av hyppigere lesinger. I dette tilfellet vil det altså ikke lønne seg idet hele tatt å lese hver I-ramme. Ved å lese enda mer data per lesing kan man spare mer disktid per leveranse (se Figur 8.3). Ved å doble lesemengden fra 128 kByte til 256 kByte øker lesetiden med 46%, ved igjen å doble fra 256 til 512 kByte øker lesetiden med 68%. Det betyr at man nok kunne brukt en segment størrelse større en 128 kByte med fordel. Noe av bakdelen med å lese mer data, er at reaksjonstiden for skifte mellom ulike spolemetoder vil øke. Man må også ta hensyn til overhead ved innlesning av ubrukte data, ved skifte fra vanlig avspilling til spoling med spolefil. Dette kan kompenseres noe, ved å lese mindre data inn første gangen etter et skifte for så å lese en større mengde i de senere innlesninger.



Figur 8.3 Lesetider for lesing av data fra disk.

8.2.2 Nettverksbelastning 300 kbit/s

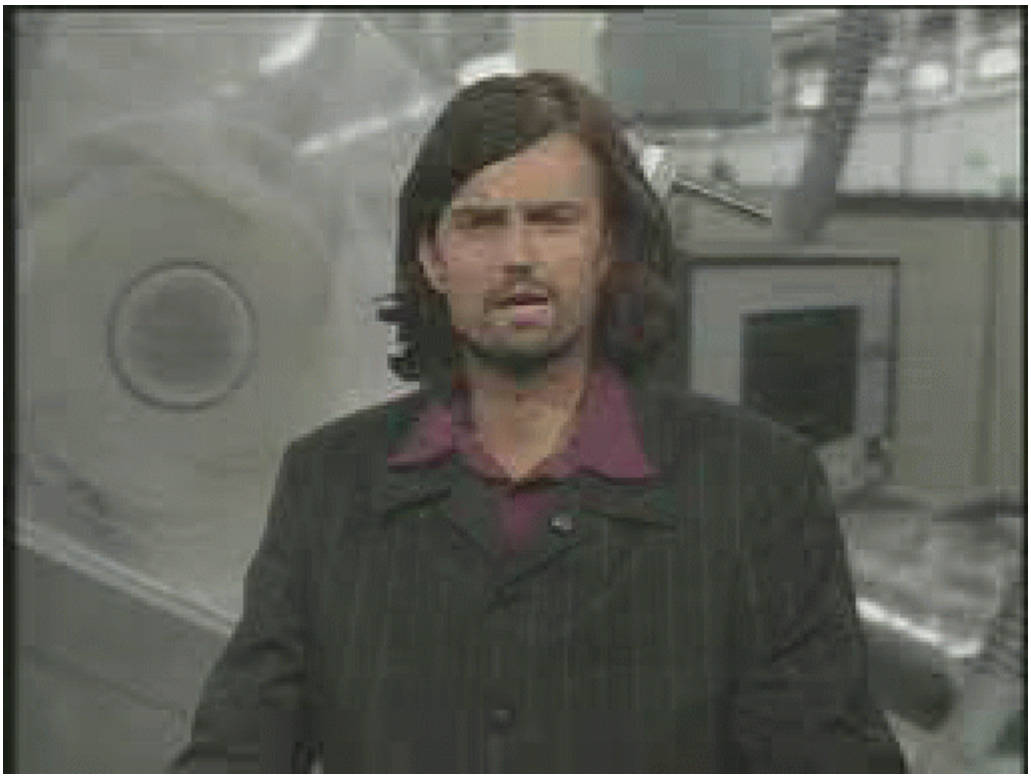


Figur 8.4 Data levert til nett ved spoling 300kbit/s fil

Figur 8.4 viser hvordan nettbelastningen er for de forskjellige metodene. Selv når alle ubrukte data er fjernet, er fremdeles spoling med spolefil minst belastende. Spoling med spolefil når en hastighet på ca 7 ganger før man overstiger belastning til vanlig avspilling det gir en rammerate på ca 12 rammer per sekund, mens spoling med I-rammer når ca 4 ganger som gir en rammerate på ca 7 rammer per sekund. Disse spolemetodene gir omtrent samme kvalitet på spolingen. Spoling med I- og P-rammer vil gi en bedre spolekvalitet, særlig i de lavere spolehastigheter, fordi 3 ganger så mange bilder blir vist, men ved denne spolemetoden når man knapt 2 ganger hastighet før belastningen overstiger belastningen ved vanlig avspilling. I stedet for å bruke spoling med I- og P-rammer bør man isteden vurdere å lage en ny spolefil for hastigheter mellom 1 og 4 ganger der man benytter I- og P-rammer fra den originale filen. Man måtte kodet spolefilen med andre rammer enn kun I-rammer for å kunne nå 4 ganger spoling uten å overstige belastningen til vanlig avspilling. Det blir en vurderingssak om man tåler lavere kvalitet, for at flere leveranser skal være mulig. Man må også huske at diskbelastningen for spoling med I- og P-rammer er 50% høyere enn diskbelastningen ved vanlig avspilling.

8.2.3 Bildekvalitet 300kbit/s

Vi vil her prøve å illustrere hvor mye bildekvalitet man taper ved å bruke spolefil for 300kbit/s fil. Vi har valgt ut et bilde fra denne videoen som skal være representativt. Man må være obs på at spolefilens størrelse ved denne bitraten ble ca dobbelt så stor som det ideelle.



Figur 8.5 Bilde fra 300 kbit/s fil ved vanlig avspilling

Hvis vi sammenligner kvaliteten på bildene i Figur 8.5 og Figur 8.6 ser vi klare forskjeller i bildekvaliteten. Nå har bildene riktig nok en størrelse som er ca dobbelt så stor som den kodede bildestørrelsen tilsier. Den originale bildestørrelsen er 192x256 punkter. På Figur 8.5 er bildet noe diffust, men de viktigste detaljene i bilde er intakt, og ved vanlig avspilling er bildekvaliteten absolutt godkjent ved angitt bildestørrelse. Figur 8.6 er helt på grensen til det

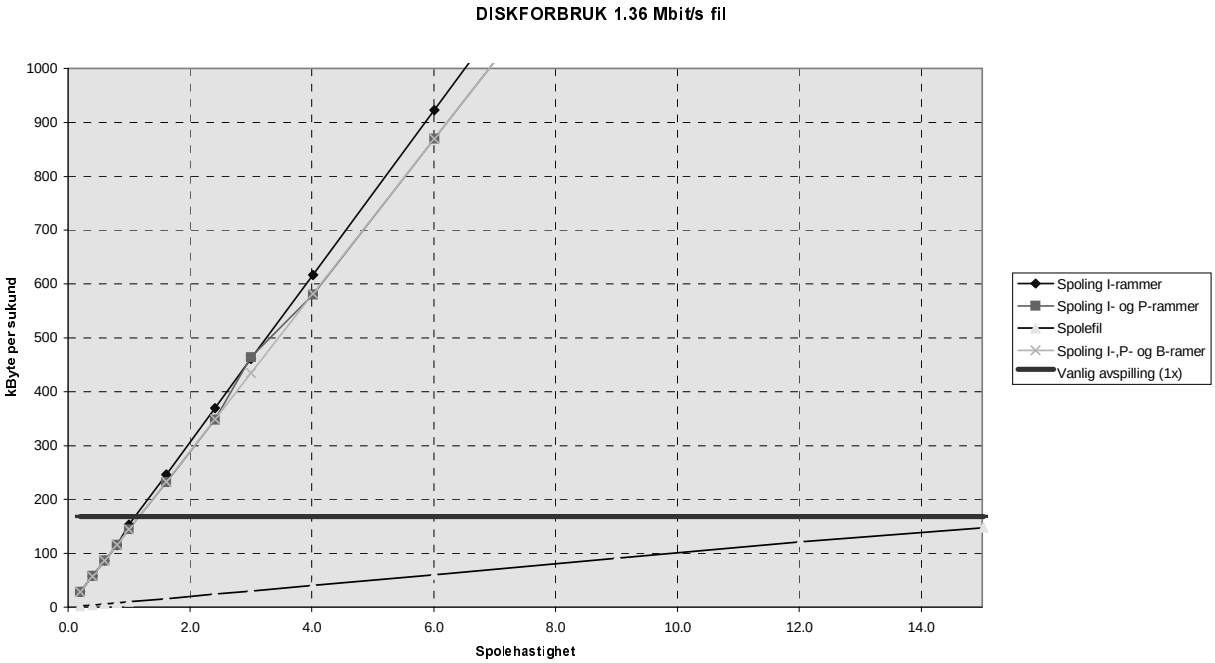
som kan aksepteres av bildekvalitet, selv for spolefiler. Man ser helt klart at makroblokkene i bildet gjør for store avrundinger, og bommer derfor mange steder ganske grovt på farge og lys. Dette er særlig tydelig i de deler av bildet som har raske skifter mellom mørkt og lyst. Selv om kvaliteten er dårlig ser man hva bilde forestiller, og vi mener det er nok informasjon i bildet til å bruke det til spoling. Husk at mange detaljer i bildet blir oversett når bildet skifter med jevne mellomrom som ved spoling. Ved en enda hardere komprimering vil hver makroblokk nærme seg en avrunding som gjorde dem ensfarget. Da er det kun DC-komponenten i hver makroblokk som lagres, og vi får en I-ramme som er lik en D-ramme.

Figur 8.6 Bilde hentet fra spolefil for 300 kBit/s fil

8.3 Spoling i fil kodet med bitrate 1.36 Mbit/s

Her presenteres måleresultater for lesing fra disk og sending av data over nett, for 1.36 Mbit/s fil. Dataene presenteres i byte, noe som tilsvarer ca 170 kByte/s for 1,36 Mbit/s.

8.3.1 Diskforbruk



Figur 8.7 Diskforbruk ved spoling i 1.36 Mbit/s fil

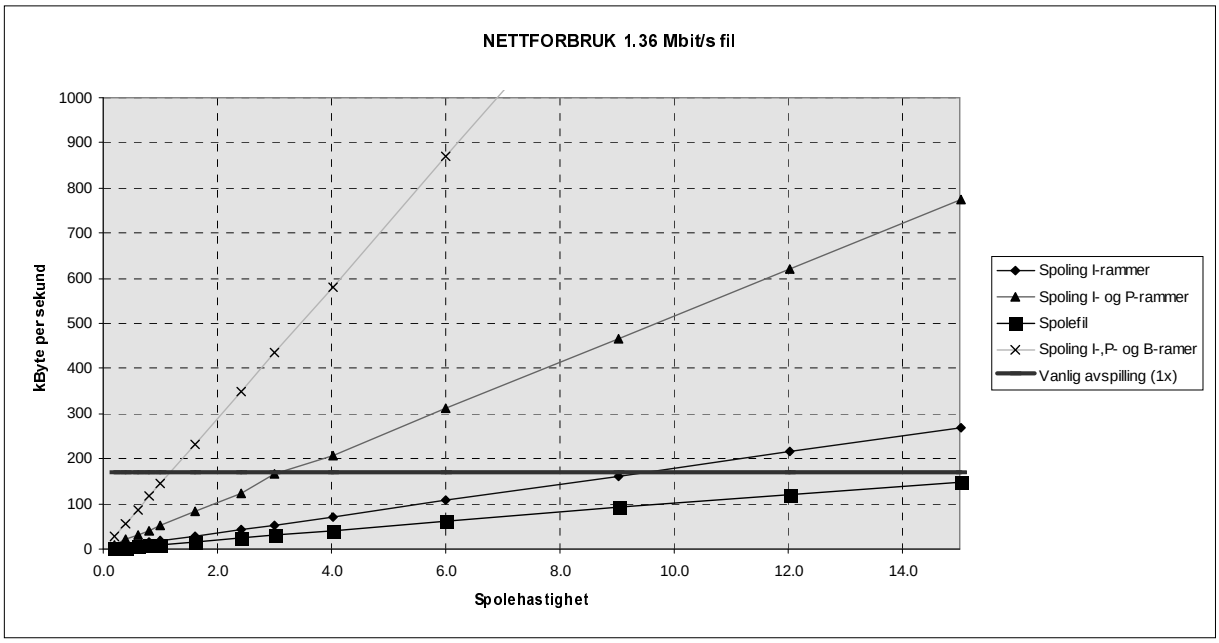
Vi ser av Figur 8.7 at vi får noe av de samme resultatene som i målingene ved 300kbit/s. En forskjell er selvfølgelig at bitratene er nesten 5 ganger større, men den største forskjellen er at spoling med spolefil ligger enda lavere her. Hvis vi gjør samme sammenligningen ved 6 ganger spoling som tidligere, ser vi at spoling med spolefil gir en diskforbruk på 60 kbyte/s mens man med de andre metodene har et snitt på ca 900 kByte/s. Det vil si at man ved å bruke spolefil kan oppnå 15 ganger flere videoleveringer, forutsatt at det skal være mulig for alle å spole samtidig og at diskkapasiteten er den begrensende faktor.

Her kan man også vurdere å kun lese inn I-rammene ved spoling med I-rammer. I denne filen er størrelsen på I-rammene ca 17 kByte. Videre er størrelsen på en bildegruppe ca 90 kByte og vi vil da i snitt lese in 1.4 I-rammer ved lesing av alle data. Figur 8.3 viser at lesing av 20 kByte (som vil tilsvare innlesning av 17 kByte på grunn av segmentstørrelsen på disk) fra disk vil i gjennomsnitt ta 37 ms, mens lesing av 128 kByte som er benyttet i implementasjonen, gir en lesetid på 86 ms. Vi ser da at tidsmessig vil det lønne seg å lese inn en og en I-ramme i dette tilfellet:

1.4 rammer *37 ms/ramme = 51.8 ms

Først ved lesing av blokker på 512 kByte vil man oppnå en tilsvarende effektivitet i diskbruk. Det må tillegges at lesing av en og en ramme vil belaste prosessoren mer en de andre metodene, men en klar fordel vil være at det er plass til flere I-rammer i delt minne samtidig og vi derfor ikke får de samme problemer som ellers med at rammer ikke er sendt når plassen deres i delt minne skal skiftes ut. Det må påpekes at man vil ligge langt unna effektiviteten man oppnår ved bruk av spolefil, uansett hvilken metode man velger.

8.3.2 Nettverksbelastning



Figur 8.8 Nettverksbelastning ved spoling i 1.36 Mbit/s fil

Figur 8.8 viser hvordan nettbelastningen er for de forskjellige metoden ved bruk av 1.36 Mbit/s fil. Vi ser de samme tendensene som i målingene for 300 kbit/s fil. Noe bedre resultat er det for de fleste metodene. Grunnen til dette må være at størrelsesforholdet mellom I-, P- og B-rammer er noe forskjellig i denne filen. Vi ser at spoling med I- og P- rammer når 3 ganger spolehastighet før bitraten overstiger bitraten for normal avspilling. For spoling med I-rammer når vi ca 9 ganger, mens spoling med spolefil overstiger bitraten til normal avspilling først ved 15 til 16 ganger normal hastighet.

8.3.3 Bildekvalitet 1.36 Mbit/s

Figur 8.9 Avspilling 1.36 Mbit/s

Hvis vi sammenligner Figur 8.9 og Figur 8.10 ser vi ikke den samme kvalitetsforskjellen som ble vist for 300kbit/s. En av grunnene til det er at man ved komprimering alltid har en maksimal komprimeringsgrense. Hvis denne grensen overstiges vil kvaliteten på det komprimerte bilde synke drastisk. Vi var mye nærmere denne grensen ved 300 kbit/s enn det vi er her. Det er allikevel en liten forskjell som er tydeligst å se i ansiktskonturene og logoen på bildene. Av Figur 8.8 ser vi at størrelsen på bildene i spolefilen er ca halvparten av størrelsen på de tilsvarende I-rammene i hovedfilen.

Figur 8.10 Spolefil 1.36 Mbit/s

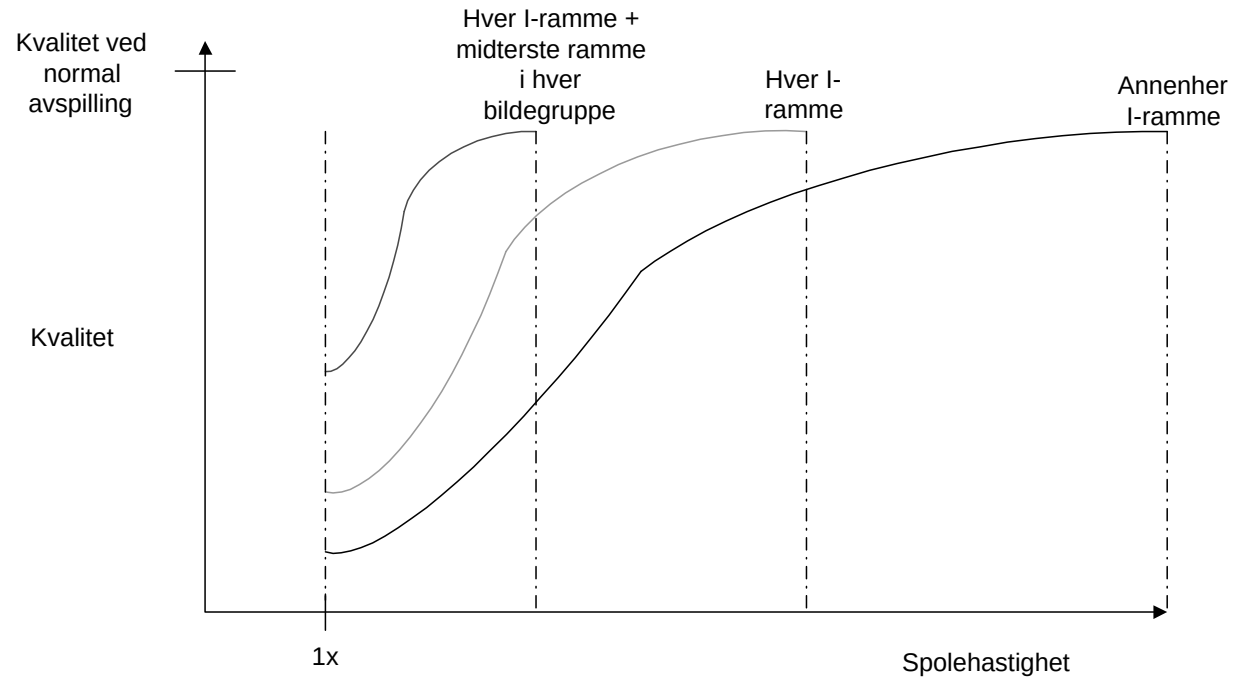
8.4 Oppsummering

Målingene viser på en god måte at bruk av spolefil er mindre ressurskrevende i en videotjener enn andre metoder. Vi har i dette eksperimentet benyttet en spolefil der størrelsen på spolefilen har vært det viktigste. Det gjør at bilde kvaliteten blir lidende under dette.

For 300 kbit/s kodede filer vil det nok lønne seg å lage en spolefil der man benytter både I- og B-rammer. Det vil kunne senke bitraten for spolefilen ned til den ønskede bitrate, som er ca halvparten av det som er realisert. Dette kan gjøres ved f.eks å kode filen med to B-rammer mellom hver I-ramme. Hvis hver B-ramme er like, og i størrelsesorden 1/6 dels I-ramme vil vi få en spolefil som ligger under ønsket bitrate. Denne metoden vil gjøre at vi må lage en spolefil for hver spoleretning. Vi vil risikerer også å få større hopp i filmen ved overgang til spoling med spolefil siden vi da må finne nærmest I-ramme i spolefilen i forhold til siste ramme vist ved vanlig avspilling.

For 1.36 Mbit/s er spolefilen som er laget grei å bruke. Man kunne kanskje tenke seg å ha en spolefil i tillegg der man viser f.eks. dobbelt så mange rammer fra filmen. Den vil kunne brukes ved lavere spolehastigheter for å gi en bedre kvalitet på spolingen i området 2 til 5 ganger spoling.

Ved å benytte flere spolefiler, ment for ulike spolehastigheter kan kvaliteten på spolingen økes. Dette er forsøkt fremstilt i Figur 8.11. Her viser vi hvordan spolekvaliteten synker ved lavere spolehastighet for ulike spolefiler.



Figur 8.11 Spolekvalitet for forskjellige spolefiler.

Ved å bruke alle tre spolefilene vist i figuren, vil man kunne bruke den spolefilen som i det ønskede området har den beste spolekvaliteten, og dermed oppnå en høyere spolekvalitet enn ved bruk av kun en spolefil. Denne kvalitetsøkningen ligger mest på antall rammer som vises per sekund.

Vi vil nå se på rammerater ved spoling. De laveste rammeratene får man ved lav spolehastighet. Den laveste spolehastigheten som benyttes vanligvis er 2 til 3 ganger normal hastighet. Ved en spolehastighet på 2 ganger vil man ved spoling fra spolefilen i målingene ha en rammerate på ca 3 rammer i sekundet. Ved å lage en ny spolefil med en ekstra ramme i mellom hver I-ramme vil rammeraten komme opp i 6 rammer per sekund. Det vil gi en klar kvalitets forbedring. Ved å inkludere enda flere rammer mellom hver I-ramme vil man kunne øke kvaliteten ytterligere. Akkurat hvor stor forskjell i antall rammer hver spolefil skal ha avgjøres av hvilke spolehastigheter man ønsker å oppnå best kvalitet på.

Ulemper med å bruke spolefil er at den må genereres. Man må altså ha tilgang på en dekoder der man kan få plukket ut og gitt de dekodede rammene videre til en koder på en eller annen måte. Bruk av spolefil vil ta ekstra lagringsplass i forhold til de andre metodene, fordi det er en ekstra fil.

9. Oppsummering og konklusjon

Dette kapittelet vil oppsummere arbeidet som er gjort i denne hovedoppgaven. Det vil deretter bli gjort en vurdering av arbeidet og en konklusjon treffes. Til slutt oppsummeres arbeid som gjenstår for videreføring av videotjeneren til et komplett videosystem, og hvilke utvidelser som er tenkt ut.

9.1 Arbeid som er utført

Det har blitt gjort et grundig studium av MPEG-1 standarden, både for audio og video. Det er særlig sett på oppbygningen av de ferdig kodede strømmene, oppdelingen av disse i hierarker, hvor kritiske data ligger og hvordan de er sammensatt. Man har også funnet endringer som kan gjøres i videostrømmen uten at standarden brytes.

Kunnskapen som er ervervet er benyttet for å utvikle måter å manipulere audio og video, slik at man oppnår VCR-funksjonalitet. Det er funnet tre hovedmetoder for å oppnå dette:

- Utplukking og eventuell manipulering av data som er uavhengig av andre data i filen
- Konstruksjon av egen fil for spoling basert på data fra hovedfil
- Bruk av sanntids dekoder/koder for dekoding av video, utplukking av ønskede bilder for spoling, og koding av disse igjen.

For konstruksjon av egen fil for spoling, er verktøy for dekoding av MPEG-1 video, og lagring av dissse i fil, funnet og benyttet. Det er også funnet verktøy for koding tilbake til en ferdig spolefil.

En videotjener er konstruert, delvis basert på ELVIRA [Langørgen-94]. Hovedgrunnen til at videotjeneren er laget, er for å kunne teste ut de utviklede prinsipper for spoling. Det er i tillegg brukt prinsipper for dataoverføring fra disk til nettverk som er spesielt ressursbesparende. Videotjeneren kan også gi en mulighet for sammenligning av to videotjenere der den ene benytter MPEG-1 og den andre benytter Motion-JPEG. Under konstruksjonen av videotjeneren er det tatt mest hensyn til den ønskede funksjonaliteten, og ikke så mye hensyn til å lage en videotjener med størst mulig kapasitet.

Det er implementert en videotjener der 3 av spolealternativene fra kapittel 5 er lagt inn. Disse er:

- Spoling ved hjelp av I- og P-rammer
- Spoling ved hjelp av I-rammer
- Spoling ved hjelp av spolefil.

I tillegg er funksjoner som sakte film i begge retninger, og tilfeldige hopp i videoen implementert. Det har blitt valgt å bare bruke lyd ved normal avspilling, slik det er vanlig å gjøre for vanlig video.

Ut fra implementasjonen er det gjort målinger for å sammenligne de ulike metodene og finne hvilken av disse som er best å bruke utfra de kriterier som stilles. Målingene baserer seg fremst på diskforbruk og nettverksbelastning, siden det er disse faktorene som har vist seg som flaskehalser i tidligere utviklede system.

9.2 Konklusjon

Utfra det som står i utført arbeid vil vi konkludere med at man har løst de oppgavene som var forespeilet i oppgaveteksten. Vi har funnet prinsipper for spoling i MPEG-1 video. Disse prinsippene er implementert og testet, slik at man har sett at de fungerer i praksis. En videotjener som bruker dette er laget. Det mangler riktig nok et helt system for overføring over nett og dekoding/avspilling av data levert fra videotjeneren. Det gjør at man ikke har fått testet f.eks videotjenerens leveringskapasitet, men de viktigste målingene for denne hovedoppgaven er likevel blitt gjort.

Målingene viser at de ulike prinsippene har alle sine bruksområder, men at det er en metode som skiller seg klart ut når det gjelder ressursforbruk. Hvilken man skal velge å bruke er avhengig av ressursknapphet i videotjener og nettverk kontra kvalitetskrav fra brukeren.

Spoling med spolefil er den beste metoden med hensyn til ressursforbruk. Den er den eneste metoden som kan utarbeides slik at den ikke belaster videotjeneren, disk og nettverk mer en det som gjøres ved vanlig avspilling, og samtidig gir et bra resultat. Man må tåle en noe lavere bildekvalitet, men det burde være noe man er vant med i forbindelse med spoling i vanlige videoavspillere. Man kan også lage flere spolefiler med ulik kvalitet til samme filmen, slik at man kan styre kvaliteten på spoling ut ifra kapasiteter i videotjeneren og på nettverk. Denne kvalitetsforskjellen kan gå på både bildekvalitet og antall bilder som vises per sekund. Ulempen med spolefil er at den gir en mer komplisert videotjener ettersom den må kunne skifte mellom filer. Bruk av spolefil krever også ekstra lagringsplass for data i forhold til de andre to metodene.

De andre to prinsippene vil kunne gi en bedre bildekvalitet en spoling med visse typer spolefiler. Kvalitetsforskjellen er avhengig av hvordan spolefilen er laget, dvs. om den er laget for minimal bitrate eller optimal kvalitet. Det viser seg (vist i målinger) at den største ulempen med disse metodene er at de gir et meget høyt diskforbruk. Særlig blir dette tydelig ved høyere spolehastigheter. De vil også gi en høyere nettverkstrafikk enn en spolefil. Vi mener derfor å ha vist at disse ikke bør benyttes i en videotjener. Den eneste grunnen til å benytte disse må være at man ikke har mulighet for å lage en spolefil.

9.3 Videre arbeid

Det gjenstår en del arbeid med å få et komplett system med videotjener. De ting som gjenstår er:

- Integrasjon av katalogtjener og administrator
- Riktig tolkning av kommandoer overført fra klient til videoleverandør
- Kommunikasjonssystem

- Klient

Etter at dette arbeidet er utført, kan målinger av videotjenerens leveransekapasitet måles. Her kunne man tenke seg målinger som tok utgangspunkt i målingene gjort på ELVIRA [Sandstå-96], for å se om man har bedret kapasiteten for levering av video, for de brukte maskiner.

Man kan tenke seg en utbygning av videotjeneren til å kunne levere MPEG-2-kodet video. Dette vil gi en god mulighet for å sammenligne disse to standardene. En slik utvidelse av videotjeneren kan gjøres ved å bytte ut de to MPEG-1 spesifikke objektene.

For å gi videotjeneren kapasitet for flere leveranser, bør lokal striping av data vurderes. Dette vil kunne gi muligheter for en betraktelig økning av antall samtidige leveranser i forhold til den nåværende kapasiteten.

Den implementerte versjonen av videotjeneren kan enkelt parallelliseres ved å benytte flere maskiner. For å få dette til må imidlertid administratoren utvides til å takle dette.

For å oppnå best mulig spolekvalitet i forskjellige spolhastigheter kan man utvide videotjeneren slik at den klarer å håndtere flere spolefiler for hver video. Man kan f.eks ha en spolefil for området 1 til 5 ganger og en for området 5 til 15 ganger. Dette vil helt klart bedre kvaliteten på spolingen i lavere spolehastigheter.

10. Referanser

[ActiveX-96] ActiveX Resources
http://www.microsoft.com/activex/

[Apple-93] Inside Macintosh: QuickTime
Apple Computer Inc, mars 1993
Apple Technical Library

[Apple-96] QucikTime 2.0 Description
http://cgi.info.apple.com/cgi-bin/read.wais.doc.pl?/wais/TIL/-QucikTime!2.0!!Description

[ATM-96] ATM Service Categories: The Benefits to the User
The ATM Forum 1996

[CD-94] A Fundamental Introduction To The Compact Disk Player
Dr. Kevin M. Buckley
Department of Electrical Engeneering, University of Minnesota
http://www.tc.umn.edu/nlhome/g496/eric0139/Papers/paper.html-#developments

[Fluckiger-95] Understanding Networked Multimedia, -Applications and Technology
Fraqois Fluckiger
Prentice Hall 1995

.

[Gonzalez-92] Digital Image Processing
Rafael C. Gonzalez, Richard E. Woods
Addison Wesley 1992

[H.261] H.261
CCITT/ITU-T standard.

[Hilton-94] Compressing still and moving images with wavelets
Michael L. Hilton, Bjørn D. Jawrth og Ayan Sengupta
Multimedia Systems:side 218-227 ,nr. 5 1994

[Hodge-94] Interactive Television (A Comprehensive Guide for Multimedia Technologists)
Winston William Hodge
McGraw-Hill Series On Visual Technology ,1994

[Infoworld-96] Product Comparison Sidebar
http://www.infoworld.com/pageone/testcenter/pcsb042996.html

[ISDN] Integrated Services Digital Network
CCITT/ITU-T standard.

[JPEG] Joint Photographic Expert Group (JPEG)
ISO/IEC 10918 -1

[Koegel-94] Multimedia Systems
John F. Koegel Buford
ACM Press 1994

[Koteng-96] Fleksibel avspilling av MPEG-video
Roger Koteng
IDT, våren 1996

[Langørgen-94] Eksperimentell videotjener for ATM. (Hovedoppgave NTH)
Stein Langørgen
IDT, høsten1994

[Liu-96] Performance Comparison of MPEG-1 and MPEG-2 Video Compression Standards
Sam Liu
Side 199-203, Proceedings of COMPCON 1996

[MPEG-92] Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbit/s.
Draft International standard ISO/IEC DIS 11172 1, 2 and 3, 1992

[MPEG-95] Generic Coding of Moving Pictures and Assocated Audio
ISO/IEC 13818-1,2 and 3, 1995

[Murray-94] Encyclopedia of Graphics File Formats
James Murray og William Van Ryper
O'Reilly & Associates 1994

[Pan-95] A Tutorial on MPEG/Audio Compression
Davis Pan
IEEE Multimedia, Side 60 -74, Volume 2, Number 2, Summer 1995

[Sandstå-96] Olav Sandstå, Stein Langørgen og Roger Midtstraum
Design anvd Implementation of the Elvira Video Server
Prodeeding fra Norsk Informatikkonferanse, Alta,
18-20 november 1996,
side 257-270

[Tanenbaum-96] Computer Networks (Third edition)
Andrews S. Tanenbaum
Prentice Hall 1996

[Tekalp-95] Digital Video Processing
A, Murat Tekalp
Prentice Hall 1995

[Tiger-96] The Tiger Video Fileserver
William J. Bolosky, Joseph S. Barrera III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold og Richard F. Rashid
Proceedings of the NOSSDAU'96 Zusli, Japan, april 1996

[Ward-95] The MPEG Library Version 1.2
Greg Ward (greg@pet.mni.mcgill.ca)
21 juni 1995

[zdnet-96] Cable Modems
http://www.zdnet.com/wsources/960617/featsub4.html

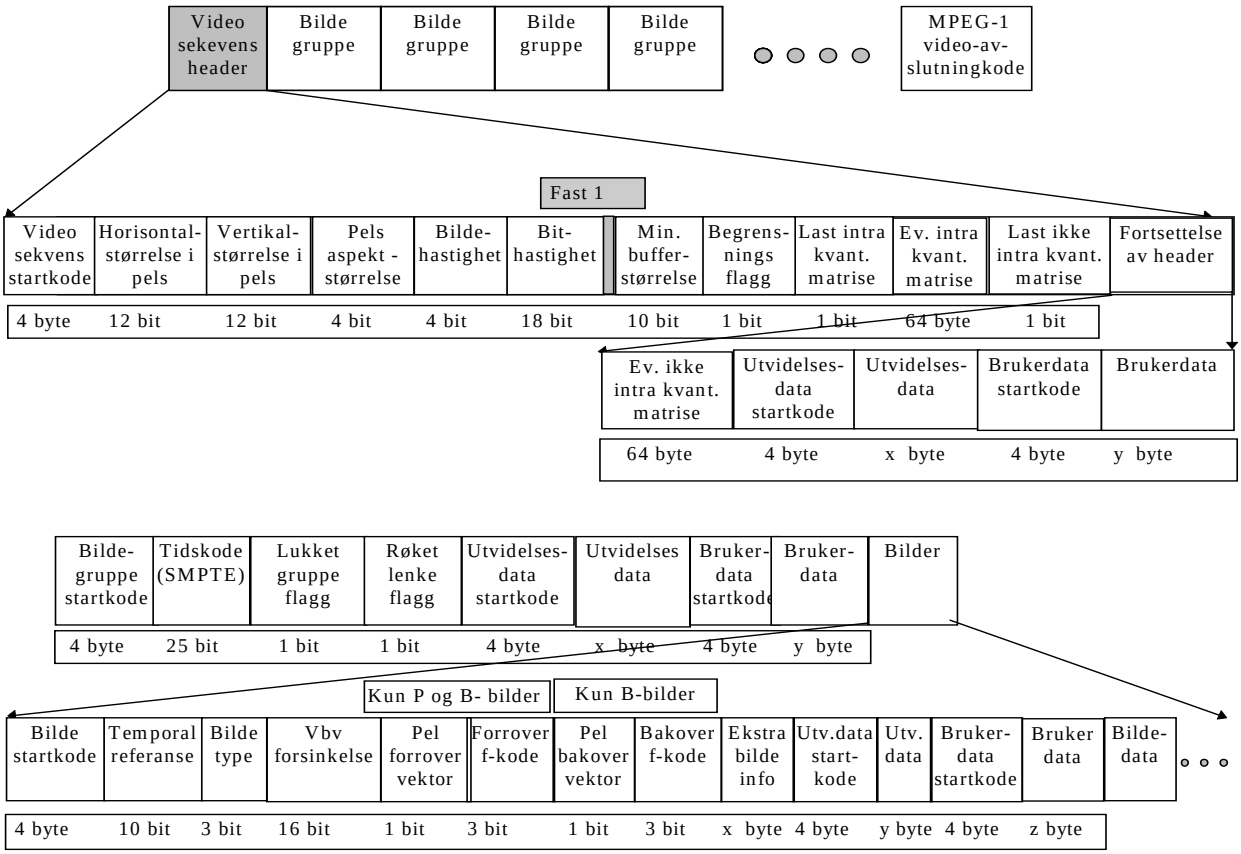
11. Vedlegg 1

MPEG-1 videobitstrømshierariki

11.1 INNLEDNING

Det vil her bli gitt en oversikt over synkronisering og identifikasjonskoder for gjenkjenning av MPEG-1 bitstrømmens enkelte komponenter. De vil bli lagt vekt på å forklare de deler som har betydning for løsning av oppgaven. Alle deler av standarden vil ikke bli gjennomgått. For en fullstendig oversikt over standarden henvises til ISO 11172 standarden [MPEG-92].
Først blir oppbygningen til videostrømmen gjengitt, så gjengis audiostrømmens viktigste holdepunkter. Til slutt vises systemstrømmens oppbygning. Alle opplysninger gitt her er hentet fra ISO 11172 standarden [MPEG-92].

11.2 VIDEOSTRØM SYNTAKS



Figur 11.1. Videostrøm hierarki