Planleggingsalgoritmer for tilfeldig dataaksess på serpentinbånd

Sammendrag

For systemer med enorme behov for lagringskapasitet vil det av kostnadsmessige grunner være ønskelig å lagre informasjonen på digitale bånd. Den største ulempen med slik lagringsteknologi er at aksesstidene er uforholdsmessig lange. For å begrense dette problemet trengs det en god innsikt i den aktuelle lagringsteknologien på et lavt nivå. På den måten det kan utvikles en modell som beskriver søketidene. En slik modell kan så legges til grunn for å utvikle planleggings-strategier slik at tilfeldige aksesser mot båndstasjonen kan sorteres i den rekkefølgen som minimaliserer total søketid. Denne hovedoppgaven beskriver en avansert modell for Tandbergs MLR1 båndstasjon, og benytter denne modellen til å utvikle, teste og evaluere forskjellige planleggings-strategier for tilfeldig dataaksess på serpentinbånd.

Søking med MLR1 fra starten av båndet resulterer i pyramide-plottene som er karakteristisk for serpentinbånd. Gjennomsnittlig søketid fra starten av båndet til en tilfeldig posisjon på båndet er 65 sekunder. Gjennomsnittlig søketid mellom to tilfeldige posisjoner er 46 sekunder.

Med utgangspunkt i søke-resultater blir det presentert en matematisk modell av båndstasjonen. Denne modellen omregner logiske blokkposisjoner til fysiske posisjoner innenfor båndets fysiske lengde. Ved hjelp av en karakte-

riseringsprosess som kartlegger hvor mange blokker det er for hvert sporsett, kan man estimere hvor lang tid det vil ta å søke mellom to tilfeldige posisjoner på båndet. Forsøkene viser at dette kan gjøres med en gjennomsnittlig feil på under 2,5 sekunder.

Basert på båndstasjonsmodellen har det i denne hovedoppgaven blitt utviklet og testet forskjellige planleggings-strategier. READ er en metode som leser hele båndet fra start til slutt. Denne strategien er meningsløs for få forespørsler, men er den beste strategien for 800 eller flere forespørsler. K-SCAN er en algoritme som reorganiserer forespørslene utelukkende med hensyn på fysisk posisjon. Dette gjøres på en slik måte at alle forespørsler på forover-rettede sporsett leses først i rekkefølge, og deretter leses forespørslene på bakover-rettede sporsett i rekkefølge. K-SCAN er den beste strategien når man skal aksessere mellom 32 og 256 forespørsler. SLTF-algoritmen reorganiserer forespørslene ved å beregne tidskostnader for forflytning fra der man står i øyeblikket og til alle posisjoner som skal aksesseres. Aksessen med lavest kostnad plukkes ut og prosessen gjentas til alle forespørslene er behandlet. SLTF er den beste strategien for færre enn 32 forespørsler og for mengder på mellom 256 og 800 forespørsler. Prosesseringstiden (Cpu-tiden) det tar å utføre disse planleggingsalgoritmene er neglisjerbar i forhold til total aksesstid, og har også liten betydning for hvor lenge man må vente på første aksess. Av de uttestede algoritmene er SLTF i så måte den mest kostbare.

Forord

Denne rapporten er resultatet av en hovedoppgave med formål å utvikle og teste forskjellige algoritmer for planlegging av forespørsler mot digitale bånd. Oppgaven er gjennomført høsten 1997 ved *Gruppe for databasteknikk, Institutt for datateknikk og informasjonsvitenskap* ved *Norges teknisk-naturvitenskapelige universitet*. Den er en videreførelse av et prosjekt jeg gjennomførte våren 1997, og er knyttet til databasegruppas pågående utvikling av en videotjener.

Jeg vil takke faglærer Roger Midtstraum og veileder Olav Sandstå for god oppfølging og veiledning i forbindelse med hovedoppgaven. De ukentlige møtene har vært veldig fruktbare for min del, og jeg føler at jeg har blitt ledet i riktig retning hele veien. Samtidig har jeg fått den friheten jeg har hatt behov for med tanke på løsing av oppgaven. I tillegg vil jeg benytte anledningen til å takke de samme personene for samarbeidet med artikkelen som ble presentert på Norsk Informatikkonferanse på Voss, og som denne hovedoppgaven bygger på.

Jeg vil også rette en takk til medstudent Rune Sætre som i våres tok seg god tid til å forklare meg en del av båndstasjonens mange underfundigheter, og

som har tilrettelagt bibliotekfunksjonene som ble benyttet ved operasjoner på den. Bente Brembo takkes for korrekturlesing av rapporten.

Til slutt vil jeg rette en takk til Tandberg Data, og spesielt Karel Babcicky, som lot meg jobbe der i sommer. Dette ga meg mye verdifull erfaring med bruk av MLR1 båndstasjon, og det har jeg nytt godt av under arbeidet med hovedoppgaven.

Thomas Maukon Andersen

Trondheim

Desember 1997

"Evig varer kun det tape'te."
-- Visdomsord

Innhold

Figurliste

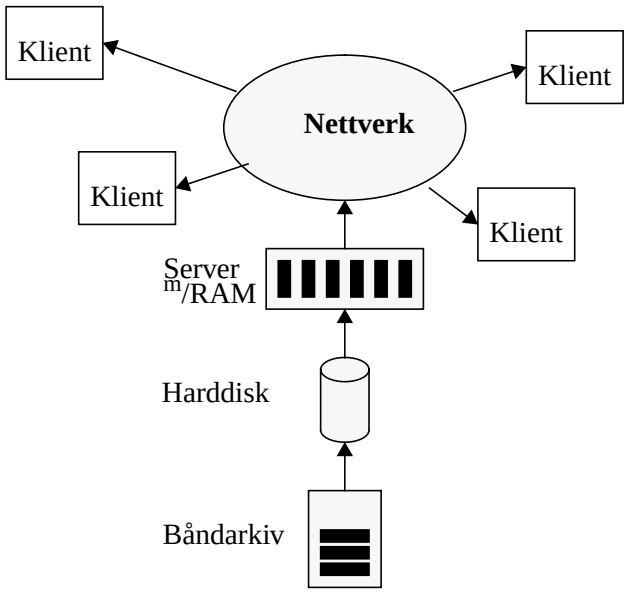
Tabell-oversikt

KAPITTEL 1

Innledning

Behovet for å planlegge forespørsler mot en båndstasjon blir stadig viktigere etterhvert som digitale bånd finner nye bruksområder. Tradisjonelt har digitale bånd stort sett blitt benyttet til sikkerhetskopiering, permanent arkivering, distribusjon av programvare, og som lagringsmedium for dataintensive målinger (i forbindelse med for eksempel oljeleting, medisinske undersøkelser og romferder). I den senere tid har behovet for lagring av store mengder data, som man også ønsker å kunne aksessere relativt hyppig, medført at digitale bånd igjen har kommet i søkelyset som en billig og anvendelig teknologi.

Institutt for datateknikk og informasjonsvitenskap (IDI) ved *Norges teknisk-naturvitenskapelige universitet* (NTNU) ønsker å benytte seg av båndteknologi for lagring av digitalisert informasjon i tilknytning til et pågående videoarkiv-prosjekt. Digitale bånd skal benyttes som tertiærlager for den audio/video-informasjonen som arkivet vil bestå av. Magnetiske disker benyttes som sekundærlager. Figur 1 viser en prinsippskisse av et slikt videoarkiv.



FIGUR 1. Prinsippskisse av et videoarkiv.

Selv om det er instituttets forskning på videotjenere som er utgangspunktet for denne hovedoppgaven, skal ikke dette sette begrensninger på modellen av båndstasjonen som benyttes, og ei heller på utviklingen av planleggingsalgoritmene. Det er derfor valgt å benytte bånd fulle av tilfeldige data, gruppert i blokker på 32kB, som utgangspunkt for forsøkene. Således blir ikke koblingen mot digital video særlig sterk, men det setter fokusen på planleggingsalgoritmene i stedet for på dataene som blir lagret. Lagring av digital video blir likevel diskutert, da dette er et typisk eksempel på en applikasjon som kan benytte seg av digitale bånd.

Problemstilling

Bruken av digitale bånd som lagringsmedium er ikke uten ulemper. Særlig er de uforholdsmessig lange aksesstidene ved tilfeldig I/O et stort problem. Det er ikke til å komme fra at dette er en ulempe som ligger i teknologiens natur ved at det er et lineært medium. Utfordringen blir derfor å minimalisere disse aksesstidene, og da særlig når man snakker om mange, tilfeldige aksesser etter hverandre.

En måte å redusere aksesstidene til et lineært medium på, er å benytte seg av serpentin-teknologi. Dette forklares nærmere i et senere kapittel, men det viktigste resultatet for båndene brukt i denne undersøkelsen er at ingen søk tar mer enn 126 sekunder. Dette er en fysisk karakteristikk ved båndteknologien som gir et rammeverk brukeren må forholde seg til. I siste instans betyr det at står man i ene enden av båndet og trenger noe som ligger i andre enden så tar det 126 sekunder enten man vil eller ikke. Dette kan man redusere ved å sørge for at man ikke står i enden, men heller stiller seg mot midten av båndet. På den måten vil ingen aksesser være mer enn 63 sekunder unna. Ytterligere redusering av maksimal aksesstid er ikke mulig om man snakker om tilfeldige aksesser.

For behandling av én og én forespørsel er det altså ikke mye å gjøre. Det kan imidlertid være mye å spare på å organisere aksessene på en hensiktsmessig måte dersom man har flere samtidige forespørsler. For å kunne estimere aksesstider slik at man kan planlegge en gunstig rekkefølge av forespørslene, trenger man en modell som kan beskrive båndstasjonens oppførsel. En slik modell har blitt utviklet av forfatteren i samarbeid med Olav Sandstå, Roger Midtstraum og Rune Sætre, [1997]. Ved å bruke denne modellen som referanse, lar det seg gjøre å lage forskjellige metoder eller algoritmer som søker å minimalisere aksesstidene. Utviklingen, testingen og vurderingen av disse algoritmene er selve kjernen i den oppgaven som ble gitt. I tillegg kan også modellen evalueres i forbindelse med forsøkene som er gjort.

Metode

For arbeidet med hovedoppgaven ble følgende metode benyttet:

1. Teoristudie.
2. Utvikling og implementering av planleggings-metoder og -algoritmer.
3. Testing av planleggingsalgoritmene mot båndstasjonen.
4. Tolking av resultatene.

Teoristudiet innebar et litteraturstudium av relevant bakgrunnsstoff om digital video og lagringsmedier. For båndstasjonen som skulle benyttes ble diverse manualer fra leverandøren studert. I og med at forfatteren hadde kjennskap til båndstasjonen fra tidligere prosjektarbeid og sommerjobb var det ikke nødvendig med utdypende utprøving av stasjonens funksjonalitet. En beskrivelse av båndstasjonens oppførsel ved søking, og en modell for dette anses også for å være en del av teoristudien. Det ble også studert tidligere arbeider med tilknytning til planlegging av forespørsler mot digitale bånd, såvel som generell planleggingsteori i forbindelse med magnetiske disker og operativsystemer.

De tre siste punktene, utvikling - testing - tolking, ble gjennomført for de enkleste algoritmene først. Disse punktene har således gjennomgått en iterativ prosess der testingen og tolkingen av resultatene førte til at nye algoritmer ble utviklet, eller at gamle ble forbedret.

Utviklingen av planleggingsalgoritmene og implementeringen av disse ble basert på trinnvis utvikling. Først ble metodene som ikke benytter seg av noen modell beskrevet. Deretter ble algoritmene som baserer seg på en modell for blokkenes fysiske posisjon på båndet laget. Til slutt ble det laget en algoritme som i tillegg benyttet en modell for tidsestimering til å sortere forespørslene.

Testingen av planleggingsalgoritmene ble utført ved at man kjørte tilfeldig trukne lister gjennom planleggingsalgoritmene og deretter kjørte resultatet mot båndstasjonen. For å oppnå et rimelig statistisk grunnlag har det blitt gjennomført mange trekninger for forskjellig antall forespørsler.

Tolkingen av resultatene ble basert på plotting av gjennomsnittlige aksesstider og gjennomsnittlige totaltider for sekvenser av aksessforespørsler. Disse plottene viser hvordan de forskjellige algoritmene oppfører seg.

Begrensninger

Begrensningene forbundet med denne hovedoppgaven kan summeres opp i tre punkter:

For det første tar det lang tid å gjennomføre testing mot båndstasjonen, rett og slett fordi det er et tregt lagringsmedium når man snakker om tilfeldige aksesser. Det ble i starten vurdert å lage en simulator av båndstasjonen for å spare tid på testingen. Dette ble ikke gjort fordi utviklingen av en slik simulator er tidkrevende i seg selv, særlig fordi båndstasjonen er en svært komplisert mekanisme det ikke er trivielt å forutsette oppførselen til. Forfatteren anså at det innenfor diplomperioden ikke var sannsynlig at det kunne utvikles en god nok simulatormodell til at det var forsvarlig å tolke resultatene med denne. Når det da ble besluttet å satse på 'real-life' testing av båndstasjonen ligger det jo også i oppgavens natur at det ikke er mulig å få satt i gang testene før mye av forarbeidet er gjort. Alt i alt har dette medført at testgrunnlaget kunne vært noe mer utfyllende, men tendensene som fremkommer er klare.

For det andre omfatter undersøkelsen oppførselen til et system med kun en båndstasjon. I et mer realistisk system ville det sannsynligvis vært nødvendig med flere båndstasjoner som "samarbeidet". Samarbeid vil i denne sammenheng bety at man har en robotmekanisme som tar seg av planlegging for mange stasjoner. Dette vil trolig innføre nye problemstillinger som ikke dekkes av denne oppgaven.

For det tredje begrenses arbeidet beskrevet her til bare å gjelde søking etter, og lesing av, tilfeldig informasjon lagret i blokker på 32kB. Dette er gjort for å forenkle testene samtidig som det gir et bra bilde på hvordan båndstasjonen oppfører seg. Det kan likevel tenkes at det for spesifikke applikasjo-

ner vil være mer hensiktsmessig med en annen blokkstørrelse. I tillegg begrenser testingen seg til bare å gjelde aksesser som leser kun en blokk på 32kB. Dette er en forenkling det ikke er sannsynlig at man vil møte i den 'virkelige' verden. En utvidelse til å la aksessene være av forskjellig lengde vil likevel ikke endre bildet som tegnes her særlig, og det diskuteres litt rundt dette senere i rapporten. Det som er viktig for denne hovedoppgaven er minimaliseringen av søketidene, og dette er uavhengig av blokkstørrelse.

Organisering av dokumentet

Struktureringen av rapporten er gjort på følgende måte: Kapittel 2 gir en del bakgrunnsinformasjon om digital video og masselager-teknologier, med særlig vekt på båndteknologier. I kapittel 3 presenteres Tandbergs MLR1 båndstasjon. Her blir det blant annet gitt en beskrivelse av dataformatet som benyttes og operasjoner på båndstasjonen. Kapittel 4 presenterer flere modeller for søking med båndstasjonen og velger mest hensiktsmessig modell for denne hovedoppgaven. Kapittel 5 gir en introduksjon til planleggings-strategier. Her diskuteres det generelt om planlegging, og de enkleste metodene blir vurdert. I kapittel 6 diskuteres to varianter av samme planleggingsalgoritme. Disse algoritmene benytter en modell av båndet for planleggingen. Kapittel 7 er en fortsettelse på forrige kapittel, og her presenteres implementeringen og testingen av en mer komplisert algoritme kalt SLTF. Til slutt gis det en oppsummering av resultater og status i kapittel 8.

KAPITTEL 2

Bakgrunn og motivasjon

Dette kapittelet tar først for seg en beskrivelse av digital video med tanke på koding, komprimering og lagring. Digital video er et typisk eksempel på en teknologi som krever stor lagringskapasitet, og som man vil være interessert i å aksessere relativt hyppig. Deretter presenteres forskjellige lagringsmedier for dette formål; optisk, disk og bånd.

1. Digital video

Digital lagring av video eller film er i dag mulig som et resultat av den teknologiske utviklingen som har funnet sted de seneste årene. Øket prosesseringskraft sammen med forbedret nettverksteknologi og lagringskapasitet har særlig bidratt til at det har blitt mer lønnsomt å benytte seg av digital lagring [Merok, 1993].

Uttrykket ‘video’ kan virke litt forrvirrende. Det benyttes både som et samlebegrep for bilde (video)- og lyd(audio)-informasjon, og som en betegnelse

på kun bilde-informasjon. Som oftest går det likevel klart frem av sammenhengen hva man mener, så det byr sjelden på store problemer. I det påfølgende vil 'digital video' betegne samlebegrepet for lyd og bilde, mens 'video' henspeiler på kun bilde-delen.

Mye av informasjonen om digital video er hentet fra [Fluckiger, 1995], som beskriver dette på en langt mer utfyllende måte enn det som er nødvendig for bakgrunnsinformasjon til denne hovedoppgaven.

1.1 Koding av analoge signaler

Koding, eller digitalisering, av analoge signaler omhandler måten det analoge signalet blir oppfattet og representert med binære tall. Analog-til-digital konvertering innebærer to trinn:

1. Sampling - eller tids-diskretisering.
2. Kvantisering - eller amplitude-diskretisering.

Det første trinnet betyr at bare et diskret sett med verdier blir beholdt for (regulære) tidsintervaller. Det andre trinnet består i seg selv av to prosesser. Først blir det samplede signalet kvantisert. Ikke alle verdiene blir beholdt etter kvantiseringen. Deretter blir et unikt sett med bits, kalt et kodeord, assosiert med hver kvantiserte verdi. Dette kalles kodeord-generering, eller bare kodingsprosessen. Kombinasjonen av disse to prosessene, kvantisering og kodeord-generering, kalles altså ofte for kvantiserings-trinnet, hvilket kan være litt villedende.

International Telecommunication Union (ITU) har definert flere standarder for digitalisering av analoge signaler. Eksempler på slike teknikker er vanlig, differensiell og adaptiv differensiell pulskode-modulasjon. Bare den vanlige pulskode-modulasjonsteknikken beskrives her.

1.1.1 Pulse Code Modulation - PCM

Den enkleste måten å digitalisere et analogt signal er ved hjelp av pulskode-modulasjon, som forkortes PCM. Selv om PCM er mest brukt av alle digi-

taliserings-metoder, er den ikke nødvendigvis mest effektiv. En viktig karakteristikk for PCM er at den ikke er spesifikk til noen type signal. PCM blir likevel ofte forbundet med koding av tale, ettersom den er mye brukt innenfor telefonien.

1.1.2 PCM kodet audio

Nyquists samplingsteorem sier at for å representere et analogt signal med maksimal frekvens *f* må samplingsraten minst være *2f*. For tale er det for eksempel tilstrekkelig med en samplingsrate på 8kHz. Dette tilsvarer en sampling hvert 125µs.

For kvantisering av verdien til hvert sample, og tilegning av et kodeord til hvert kvantum, er det to teknikker som er utbredt for audio. Disse kalles lineær og logaritmisk koding, og er beskrevet nøyere i [Fluckinger, 1995]. Med logaritmisk koding kan man representere amplituden til det digitaliserte signalet med 8 bit (2^8 = 256 trinn). Dette gir brukbar kvalitet for tale og kan overføres med 64kbit/s.

1.2 Komprimering av digital informasjon generelt

For mer sofistikert koding av analoge signaler øker mengden av informasjon som må lagres og/eller overføres. Det er derfor ofte nødvendig å komprimere den digitaliserte informasjonen for å holde seg innenfor begrensningene til dagens nettverk, samt begrense kravet til lagringskapasitet. For eksempel vil 90 minutter med ukomprimert digital video kreve ca. 120GB med diskplass, mens scanning av et 35mm fargebilde med en brukbar oppløsning på 2000 x 2000[1] vil generere en fil på ca 10MB [Fluckinger, 1995].

1. Dette betyr å beholde kun 4 millioner ut av de 20 millioner piksler bildet består av.

1.2.1 Komprimering med og uten tap

Komprimeringsteknikker kan deles i to typer:

- *Tapsfri komprimering*
  Med denne teknikken beholdes integriteten til informasjonen etter dekompresjonen. Det betyr at den dekomprimerte bit-strømmen er identisk med den opprinnelige bit-strømmen. Informasjon som skal leses av maskiner må som oftest komprimeres med en tapsfri teknikk da det ikke kan tillates at informasjon blir borte.
- *Komprimering med tap*
  Dette kalles ofte for '*lossy*' komprimering og medfører at man mister noe informasjon etter dekomprimering. Disse teknikkene er ofte brukbare for kontinuerlige media som for eksempel audio og video. Det er nemlig et viktig poeng at selv om den dekomprimerte informasjonen er forskjellig fra den opprinnelige, betyr ikke dette nødvendigvis at mennesker klarer å oppfatte forskjellen.

1.3 Komprimering av audio-data

Audio-data kan komprimeres på forskjellige måter. ITU-TS[2] har foreslått en rekke standarder for dette formålet. *Eksempler er sub-band adaptive differensial PCM, GSM*[3], *CELP*[4] og *MPEG*[5]*-Audio*. Disse standardene er kort beskrevet i [Fluckinger, 1995]. MPEG-Audio omtales som et eksempel i denne hovedoppgaven fordi den er rettet mot audio generelt, og ikke spesielt mot tale.

2. ITU-TS: International Telecommunications Union - Telecommunication Sector.
3. Mobiltelefon-standard.
4. CELP: Code Excited Linear Prediction.
5. MPEG: Moving Pictures Expert Group.

1.3.1 MPEG-Audio

MPEG-Audio er ikke bare én enkelt komprimeringsalgoritme, men derimot en familie av tre kodings- og komprimerings-metoder for audio. Disse kalles MPEG-Audio Lag-1, Lag-2 og Lag-3. Kompleksiteten til algoritmen øker med lag-nummeret. Alle lagene benytter seg av det samme prinsippet: En kombinasjon av transformasjons-koding og sub-bånd divisjon. Det ligger utenfor denne hovedoppgavens tema å forklare dette i detalj, men hovedpunktene er i følge Fluckinger [1997]:

- Audio-spekteret deles i 32 sub-bånd.
- En rask Fourier transform blir benyttet for å representere signalet i frekvens-domenet.
- En psyko-akustisk modell benyttes på det transformerte signalet for å estimere støynivåene som bare så vidt er hørbare.

Lagene skiller seg i hovedsak ut i måten de utfører det siste trinnet, kvantisering, på.

MPEG-Audio familien er laget for å operere på komprimerte bitrater fra 32 til 448kbit/s. Samplings-raten kan være 32, 44,1 eller 48kHz.

Det er viktig å merke seg at MPEG-Audio er lossy, men det er mulig å oppnå kvalitet som oppfattes som tapsfri.

1.4 Komprimering av video-data

Av samme grunner som for audio er det som oftest også nødvendig å komprimere video. Komprimering består av å eliminere overflødig informasjon. For bevegelige bilder er det to typer overflødighet, kalt korrelasjoner, som kan bli eliminert eller redusert: Rom-korrelasjoner og temporære korrelasjoner.

Kompresjons-prosesser med hensyn på rom-korrelasjoner opererer på individuelle bilder. Overflødigheten innen hver ramme fjernes eller reduseres.

Teknikkene for dette er vanligvis de samme som benyttes for komprimering av stillbilder.

Redusering eller eliminering av temporære korrelasjoner innebærer å identifisere overflødigheter mellom suksessive rammer. Dette gjøres som regel ved hjelp av en differensiell PCM teknikk, hvilket betyr at kun forskjellen mellom rammene blir kodet. Tanken bak denne metoden er at over et visst tidsintervall vil det være flere like rammer, og noen rammer vil være delvis konstruert av andre rammer.

I praksis finnes det to typer hyllevare-produkter for komprimering av bevegelige bilder:

- Produkter som kun reduserer rom-korrelasjoner. Fordelen med denne typen produkter er rask komprimering og lav følsomhet for overføringsfeil. Ulempen er lavere komprimeringsgrad.
- Produkter som reduserer både temporære korrelasjoner og rom-korrelasjoner. Dette er de vanligste implementeringene. Fordelen med denne typen produkter er høyere komprimeringsgrad. Ulempen ligger i at den er mer følsom for dårlig overføring fordi flere rammer må bufres ved kilden.

1.4.1 Motion-JPEG

Motion JPEG[6] er et eksempel på et produkt som kun reduserer rom-korrelasjoner. Hver ramme komprimeres individuelt. Det finnes hyllevareprodukter som kan generere en video bit-strøm med video av studio kvalitet for mellom 8 og 10 Mbit/s. Komprimeringsgraden ligger i området 25:1.

1.4.2 MPEG-1

MPEG-1 er beregnet for lagring av synkroniserte audio- og video-signaler på CD-ROM. Dette betyr at, i praksis, hvis audio-kanalen behøver mellom 200 og 250 Kbit/s for CD-kvalitet, så bør bit-raten til video-kanalen ikke

6. JPEG: Joint Photographic Expert Group.

overstige 1,15 eller 1,2 Mbit/s. MPEG-1 er derfor optimalisert for middels oppløsning. I optimal modus benytter den seg av Standard Interchange Format (SIF) [Fluckinger, 1995]. Komprimeringsgraden ligger i området 26:1.

1.4.3 MPEG-2

MPEG-2 sikter seg inn mot video av studio-kvalitet og multiple audiokanaler med CD-kvalitet ved 4 til 6 Mbit/s. Den har også blitt utvidet til å kunne benyttes av HDTV[7].

1.5 Digitale data i et video-arkiv

For lagring av digital data i et video-arkiv lister tabell 1 de mest aktuelle komprimeringsmetodene med typisk båndbredde og krav til lagringskapasitet.

TABELL 1. Komprimeringsteknikker.

Kompresjons-metode	Båndbredde		Lagringskapasitet 90 min
	Audio (Mbit/s)	Video (Mbit/s)	
MPEG-1	0,25	1,2	1,2GB
MPEG-2	0,25	5,0	3,2GB
Motion JPEG	--	9,0	2,0GB

Motion JPEG støtter altså ikke audio, så i stedet må f.eks. 8 bits PCM-kodet audio benyttes i tillegg. Siste kolonne viser lagringskapasiteten som trengs for en 90 minutters film, og det sier seg selv at et videoarkiv av en viss størrelse vil ha et stort behov for lagringskapasitet.

7. HDTV: High Definition Television.

2. Lagringsmedier

Kjell Bratbergsengen oppsummerer en del viktige egenskaper for lagringsmedier [Bratbergsengen, 1997]. Hvilke egenskaper som er viktigst vil avhenge av aktuell bruk. Tabell 2 gir noen eksempler.

TABELL 2. Viktige egenskaper for lagringsmedier.

Egenskap	Beskrivelse
Pris per MB lagret	Dersom det skal lagres virkelig store mengder data vil prisen på lagringsmediet, og maskinvaren forbundet med det, alltid spille en viktig rolle. F.eks. estimeres det i [Bratbergsengen, 1997] at Nasjonalbiblioteket ville trenge magnetiske disker til mellom 80 og 800 millioner kroner, eller båndteknologi til en tidel av prisen.
Overførings kapasitet	For applikasjoner som trenger store mengder data på en gang (f.eks. en video-avspilling, eller løsing av et stort ligningsett på superdata-maskin) vil overføringshastigheten spille en avgjørende rolle.
Mobilitet	Ofte vil det være ønskelig at mediet kan flyttes fysisk, f.eks. for lagring av sikkerhetskopier i brannsikre hvelv.
Holdbarhet	Holdbarheten til dataene kan være av betydning. Igjen vil f.eks. Nasjonalbiblioteket ønske lengst mulig holdbarhet. Optiske medier har gjerne best karakteristikker på dette området (>15 år), mens magnetiske medier ligger lavere (2-10 år)
Lagrings-kapasietet	Lagringstetthet sier noe om hvor stor fysisk plass mediene tar. Vanligvis vil man ønske så høy tetthet som mulig, særlig for virkelig store mengder data.
Aksesstid	I systemer der det lagres store mengder data, men hvor bare en del av dataene brukes av gangen vil det være viktig at aksesstiden, eller responstiden, blir så lav som mulig. Et typisk eksempel er et billettbestillingssystem. Tradisjonelt kommer båndteknologi særlig dårlig ut på dette området.

For lagring av store datamengder (flere gigabyte) er det i realiteten i dag bare tre medier som egner seg; magnetisk disk, magnetbånd eller optiske

lagringsmedier som f.eks. CD-ROM (Compact Disk, Read Only Memory), WORM (Write Once, Read Many) eller DVD (Digital Video Disc[8]).

2.1 Optiske lagringsmedier

CD-ROM har relativt liten overføringskapasitet samtidig som det typisk kan lagres maksimalt rundt 600MB per plate. Dette blir for lite i forbindelse med lagring av mye videoinformasjon. WORM-disker har langt større lagringskapasitet[9], men er til gjengjeld større og langt dyrere. Overføringskapasiteten er like dårlig som for CD-ROM. En annen alvorlig begrensning med disse mediene er at de forbruker lagringsmedium. Med dette menes at når man først har skrevet data til en disk, kan plassen disse dataene opptar ikke frigjøres ved sletting. Redigering av innholdet er dermed ikke mulig etter at det er lagret på disse mediene.

DVD, som representerer siste teknologi for optiske lagringsmedier, har kapasitet opptil 16GB og en gjennomsnittlig overføringskapasitet på 4,7 Mbit/s. Teknologien er enda på innføringsstadiet, men den nevnes her som en potensiell fremtidig konkurrent til magnetiske disker eller båndstasjonsteknologi for masse-lagring av data. DVD vil komme i en skrivbar versjon som refereres til som DVD-RAM [Tactical Marketing Group, 1996].

2.2 Magnetiske disker

Magnetiske disker har i den senere tid øket betraktelig i lagringskapasitet samtidig som prisen per MB har sunket mye. Dette gjør mediet til en viktig komponent i et videoarkiv-system. Magnetiske disker har en rekke fordelaktige egenskaper, blant annet svært rask tilgang til data, tilfeldig aksess og stor overføringskapasitet. I tillegg er magnetiske disker enkle å oppdatere. Ved riktig store mengder data (flere TeraByte) blir det likevel for dyrt med

8. Det er foreløpig litt uenighet om hva DVD egentlig står for. Et alternativ til Digital Video Disc er Digital Versatile Disc.
9. Hitachi OD321 kan lagre opptil 7000MB [Bratbergsengen, 1997].

lagring kun på magnetisk disk. Lagringstettheten er heller ikke god nok for slike datamengder.

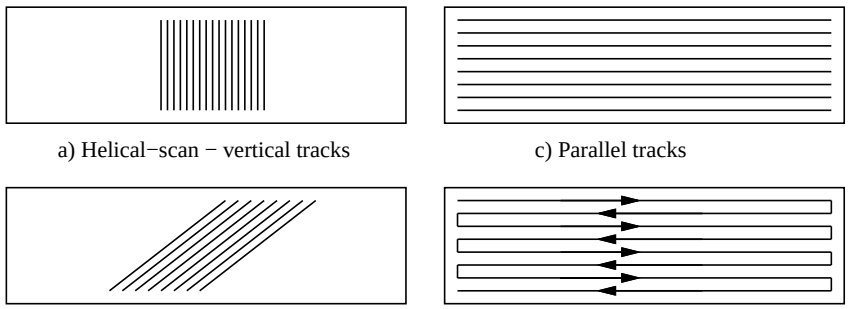
2.3 Magnetbånd

Magnetbånd har alltid vært mediet for langtidslagring av store datamengder. De beste båndene i dag kan lagre mellom 10 og 165GB per kassett for en kostnad på rundt $2500 per TB. Magnetbånd er sammenlignbare eller overlegne i forhold til magnetiske disker når det gjelder lagringstetthet, kostnad og sekvensiell overføringskapasitet [Hillyer og Silberschatz, 1996a]. Den store ulempen med magnetbånd-teknologien er at aksesstiden er 3 til 4 størrelsesordner høyere enn for magnetiske disker. Dette har tradisjonelt ført til at magnetbånd fortrinnsvis har blitt brukt i applikasjoner som for det meste krever lange sekvensielle datastrømmer. Dersom overføringstiden (altså skrive- eller lesetiden) er lang nok, vil det prosentvise tapet i form av søketid minske. Overføring av lengre audio- og/eller video-sekvenser vil typisk falle inn under denne kategorien.

Det skilles mellom to hovedteknikker for lagring av data på magnetbånd: Lineære teknikker og roterende hode[10] - teknikker. Forskjellen ligger i hvordan lese/skrivehodet gis relativ hastighet i forhold til båndets overflate.

10. Det kan være litt misledende å kalle teknologien for “roterende hoder”. Selve lese/skrivehodene sitter på en roterende sylinder som kalles “scanner”. (På engelsk: helical scan)

Figur 2 viser en skisse av sporenes plassering på båndet for forskjellige typer av roterende hode og lineære teknologier.



FIGUR 2. Båndteknologier: Roterende hode (a, b) og lineære (c, d).

2.3.1 Roterende hode - teknologi

Denne typen teknologi benytter et hode montert på en roterende sylinder som båndet bringes i nærkontakt med. Med roterende hoder kan båndet føres frem sakte ettersom den relative hastigheten frembringes gjennom rotasjonen. Hodet kan roteres på tvers av båndet eller det kan rotere skrått på langs av båndet [Bratbergsengen, 1997]. Se figur 2a) og b) for en illustrasjon av roterende hode - teknologi. Skråstilte roterende hoder benyttes blant annet i de velkjente analoge VHS videobåndspillerne folk har hjemme i stua. Andre (digitale) teknologier som benytter roterende hode er 4mm DAT, 8mm video. Exabyte Mammoth er et eksempel på 8mm teknologi.

Roterende hoder utnytter kapasiteten maksimalt ettersom sporene kan bli liggende svært tett og hele bredden av båndet blir utnyttet. Denne teknologien har imidlertid den ulempen at hodene sliter uforholdsmessig mye på båndet. Et bånd kan bli utslitt etter noen få tusen passeringer. I tillegg må hodet slutte å spinne når båndet stopper, ellers ødelegges båndet etter en stund. Dette medfører at hodet må startes igjen før neste leseoperasjon, og slike start og stopp tar tid.

2.3.2 Lineær teknologi

Denne teknologien benytter seg av faste hoder. Dermed må båndet føres forbi hodet med relativt høy hastighet. Data blir skrevet langs båndet i et eller flere spor. Vi skiller mellom lineære bånd som er skrevet i parallell eller med serpentinmønster. Figur 2c) og d) viser en illustrasjon av hvordan sporene ligger på båndet. For bånd som skrives i parallell er 9 eller 18 spor (1 eller 2 byte m/paritet på tvers av båndet) det mest vanlige [Bratbergsengen, 1997]. Ulempen er at sportettheten blir begrenset av hvor nær hverandre man klarer å legge lese/skrivehodene.

Et alternativ til parallelle hoder er å skrive båndet i et serpentinmønster. Det vil si at man skriver et (eller fler) spor til enden av båndet, deretter forskyves hodet litt, og man kan skrive i motsatt retning. Slik fortsetter man til hele båndbredden er brukt opp. På denne måten kan man oppnå en langt høyere sportetthet enn ved å bruke mange parallelle hoder. Man snakker også om antall spor når det gjelder serpentin-teknologi, men for brukeren vil det se ut som man har ett langt spor.

2.3.3 Kassettstandarder for serpentinbånd

To typer kassettstandarder for serpentinbånd er fremtredende på markedet i dag. Dette er QIC - Quarter Inch Cassette, og DLT™ - Digital Linear Tape.

QIC[11] er den enkleste av disse, opprinnelig med bare ett hode. Siste versjon, QIC-5010-DC (13GB)[12], benyttes av stasjoner med to hoder. Hodene er adskilt på en slik måte at sportettheten på båndet blir mye bedre enn hodetettheten. Navnet QIC henspeiler på båndets bredde som er en kvart tomme. Tandberg Data lager i dag båndstasjoner som kan bruke QIC-bånd med 42, 52, 56, 72 og 144 spor.

11.QIC-standarden er tilgjengelig på http://www2.qic.org/qic/html/qicstan.html (pr. 4.1.98).

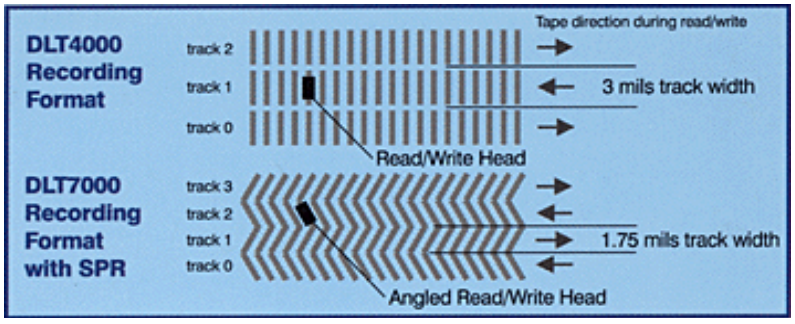
12.Tandbergs MLR1 båndstasjon kan benytte denne standarden, og det er QIC-5010-DC som blir benyttet under forsøkene beskrevet i denne rapporten.

QIC startet ut som en “billigløsning” med moderat hastighet og kapasitet, men med lav pris[13]. Etterhvert som sportettheten har øket, nærmer den seg DLT både i lagringskapasitet og overføringskapasitet.

DLT ble utviklet av DEC (Digital Equipment Corporation) på midten av 80-tallet og ble holdt innen selskapet som en proprietær teknologi i mange år [Bratbergsengen, 1997]. Først på 90-tallet ble teknologien åpent tilgjengelig, og i 1994 ble den solgt til Quantum. DLT opererer med en båndbredde på en halv tomme.

FIGUR 3. Quantum DLT^{tm} 7000

Toppmodellen til Quantum er DLT7000, som kan lagre 35GB med ukomprimert data per bånd. En av grunnene til denne høye kapasiteten er det store antall spor på båndet; 208 [Quantum Corp., 1997]. Dette oppnås ved å benytte det Quantum kaller *Symmetric Phase Recording* (SPR). SPR innebærer å vri hodene med alternerende vinkler for forskjellige spor for å unngå kryss-tale. Dette gjør at sporene kan legges tettere enn for tidligere teknologier som baserte seg på bredere spor og større mellomrom for å unngå kryss-tale. Figur 4 viser en illustrasjon av tradisjonell serpentinteknologi og SPR.



FIGUR 4. Sammenligning av tradisjonell serpentinbåndteknologi og Quantums Symmetric Phase Recording (SPR) [Intelligent Solutions Inc., 1997].

13.Per 8.4.1997 kostet et 13GB QIC-5010-DC bånd kr. 575,- eks. mva. Dette tilsvarer 4,4 øre per MB eks. mva.

En ulempe med DLT i forhold til QIC er at DLT har bare en spole. Opptakerspolen sitter i spilleren. Før en DLT-kassett kan tas ut må den altså alltid spoles tilbake og “hektes av”. Dette er operasjoner som utføres automatisk av båndstasjonen. I starten av hvert bånd er det satt av plass for en katalog. Ved innsetting av et bånd blir katalogen lest inn i båndstasjonen og båndet blir kalibrert. Alt dette tar relativt lang tid. Bratbergsengen [1997] har målt dette til gjennomsnittlig 48 sekunder. I så måte er QIC langt raskere å montere/fjerne ettersom begge spolene sitter i kassetten.

Tabell 3 viser noen karakteristikker for toppmodellene til Tandberg og Quantum.

TABELL 3. Sammenligning av Tandberg MLR1 og Quantum DLT7000.

	Tandberg MLR1	**Quantum DLT7000**
Overføringsrate[a] (MB/s)	1,5	5
Kapasitet (GB)	13	35
Båndets bredde (in)	0,25	0,5
Antall spor	144	208

a. Uten komprimering.

2.3.4 Båndarkiv

FIGUR 5. **Tandberg MLR™ Library**

Båndarkiv - eller '(tape-)library' som det ofte kalles - benytter seg av robot-teknologi for å sette sammen flere båndstasjoner og bånd i en enhet. Tandberg MLR™ 1440 Library er et eksempel på et slikt båndarkiv. Det har lagringskapasitet på 640GB[14] fordelt på 40 kassetter. Båndarkivet kommer som en kompakt enhet med kassetter og fire MLR1 båndstasjoner. Enheten består også av en robotmekanisme som mater båndstasjonene. Figur 5 viser et bilde av et slikt arkiv. Gjennomsnittlig overføringshastighet er 6,0 MB/s[14]. Tiden det tar å laste et bånd oppgis til < 9,9 sekunder, mens tiden det tar fra en kassett er matet til enheten er klar for å lese data oppgis til < 65 sekunder. [Tandberg Data ASA, 1997].

FIGUR 6. **ATL 7100**

FIGUR 7. **Alpine 460 og 228**

Det finnes også tilsvarende båndarkiv som benytter DLT-teknologi. Disse er gjerne noe større enn QIC-arkivene. Figur 6 og figur 7 viser bilder av noen

14.Forutsatt at det ikke benyttes datakompresjon.

av ATL og Alpine sine modeller. I tabell 4 oppgis det noen karakteristiske data for et QIC-arkiv og to DLT-arkiv.

TABELL 4. Sammenligning av forskjellige båndarkiv basert på [Tandberg Data ASA, 1997], [Dynamic Solutions International Corp., 1996] og [ATL Products Inc., 1997].

	Tandberg MLR 1440	Alpine 6176	ATL 7100
Båndstasjon type	MLR1	DLT 7000	DLT 7000
Antall stasjoner	4	6	7
Antall bånd	40	176	100
Kapasitet[a] (GB)	640	6160	3500
Overføringsrate (MB/s)	6	30	35
Vekt (kg)	--	431	227

a. Uten komprimering

Lagringsenheter som disse er primært utviklet for å ta vare på og behandle store mengder backup-data, det vil si data som ofte er lagret sekvensielt. Forfatteren har ikke noe kjennskap til hvordan disse arkivene oppfører seg dersom det for eksempel er nødvendig med mange hurtige utskiftninger av kassetter. Det må antas at slike operasjoner vil være praktisk gjennomførbare, men i hvilken grad det vil forringe ytelsen
er vanskelig å uttale seg om.

KAPITTEL 3

Tandberg MLR1 båndstasjon

Dette kapittelet beskriver de viktigste karakteristikkene ved Tandbergs MLR1 båndstasjon. Tekniske data er hentet fra [Tandberg Data ASA, 1996a] og [Tandberg Data ASA, 1996b], som gir en fyldigere beskrivelse enn det som er tatt med her.

Tabell 5 gir en kort oversikt over vanlige forkortelser brukt i denne rapporten:

TABELL 5. Forkortelser .

BOM	**Beginning Of Medium.** Den posisjonen langs mediet som kan nås ved å bruke en REWIND kommando.
BOP	**Beginning Of Partition**. Posisjonen på starten av den lovlige lagringsregionen til en partisjon. Hvis bare en partisjon er definert er denne posisjonen ekvivalent med BOM. Partisjoneringsmulighetene er beskrevet i “3. Partisjonering”
BOT	**Beginnning Of Tape**. Fysisk merke som angir starten av det brukbare området på båndet, dvs. begynnelsen av området man kan lagre brukerdata på. BOT er lokalisert ved BOM.

TABELL 5. Forkortelser (Fortsatt).

EOM	**End Of Medium**. Den ekstreme posisjonen langs mediet i retning fra opptaks-spolen. Denne posisjonen kan nås ved å bruke en LOAD/UNLOAD kommando med EOT-bitet satt til 1.
EOP	**End Of Partition**. Posisjonen på slutten av den lovlige lagringsregionen til en partisjon. Hvis bare en partisjon er definert er denne posisjonen ekvivalent med EOM.
EOT	**End Of Tape**. Tilsvarende BOT, men markerer slutten av det brukbare området på båndet. EOT er lokalisert ved EOM
SCSI	**Small Computer Systems Interface**. Industristandard grensesnitt for eksterne enheter tilkoblet en datamaskin. Blir brukt til å koble flere enheter sammen via en felles data- og kontroll-buss

1. Generelt

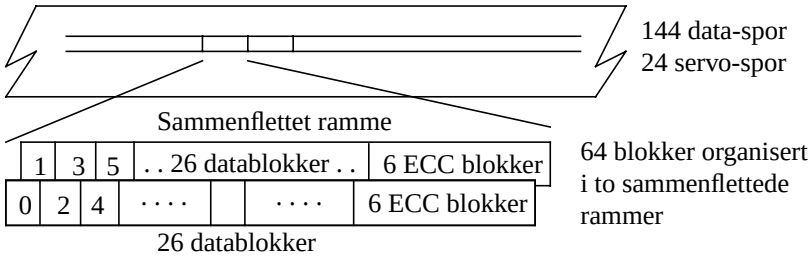
Tandberg MLR1[1] baserer seg på serpentinbånd-teknologi og støtter en rekke båndtyper og overføringshastigheter. Kassettstandarden som ble brukt i denne undersøkelsen var av type QIC-5010-DC[2] med 144 data- og 24 servo-spor, og en kapasitet på ca. 13GB ukomprimert. Med komprimering[3] vil det kunne lagres bortimot 26GB dersom man ser på gjennomsnittlige data. Ettersom video som regel er ganske godt komprimert i utgangspunktet, kan det ikke forventes effekt av komprimeringen i dette tilfellet [Sætre og Sandstå, 1996]. For å lage en generell modell som ikke er

1. MLR: Multi-channel Linear Recording.
2. Alle data som følger gjelder for QIC-5010-DC standarden. Referer til [Tandberg Data ASA, 96] for data som gjelder for andre QIC-standarder.
3. Komprimeringskretsen som benyttes i MLR1 er en ALDC1-20S-HA. Denne tilbyr datakomprimering basert på ALDC (Adaptive Lossless Data Compression) algoritmen som gir en gjennomsnitts komprimeringsgrad på 2:1. ALDC algoritmen er godkjent av flere standardiseringsorganisasjoner, bl.a. QIC.

mer avansert enn nødvendig, vil alle forsøkene beskrevet i denne rapporten bli gjennomført med komprimeringen skrudd av.

Skrive/lese-hodet til MLR1 benytter fire kanaler og består av induktive skrive- og MR[4] lese-elementer. Den ene lesekanalen følger de på forhånd lagrede servo-sporene, som inneholder posisjoneringsinformasjon. En ytterligere beskrivelse av mekanismen som posisjonerer hodet er gitt i “2. Posisjonering av lese/skrive-hodene” .

MLR1 benytter SCSI-II grensesnitt mot vertsmaskina. Data sendes fra vertsmaskina til båndstasjonens buffer. Her blir alle dataene organisert i logiske rammer som består av 64 blokker (á 512 bytes) med informasjon, hvorav 52 av disse blokkene inneholder brukerdata. Figur 8 viser en skisse av denne organiseringen.



FIGUR 8. QIC-5010-DC. Organisering av datablokker i rammer.

Rammen inneholder 52 datablokker og 12 ECC[5] blokker. Både datablokkene og ECC blokkene blir sammenflettet som vist. Dette betyr at datablokkene 0, 2, 4, 6, ... vil generere ECC blokkene 0, 2, 4, ..., 10, mens datablokkene 1, 3, 5, ... vil generere ECC blokkene 1, 3, 5, ..., 11.

For skriving og lesing av båndet benyttes to spor i parallell. Disse kalles et sporsett. Dette er organisert slik at partalls rammer blir skrevet på partalls

4. MR: Magneto Resistive.
5. ECC står for 'Error Correction Characters'. Dette er feilkorrigeringsinformasjon som genereres av stasjonen.

spor, mens odde rammer blir skrevet på odde spor. Første sporsett er sporsett 0, og siste sporsett blir dermed sporsett 71.

Data som skrives til båndstasjonen formateres inn i små blokker på 512 bytes. Til hver blokk legges det spesielle adresse- og sjekke-bytes. viser en skisse over en standard datablokk. Som det framgår av figuren er det en overhead på 29 bytes pr. 512 bytes data. ECC-blokker er da ikke tatt med.

Preamble	Block marker	Control Field	Data	CRC	Postamble
13 bytes	3 bytes	8 bytes	512 bytes	4 bytes	1 bytes

FIGUR 9. Standard datablokk - QIC-5010-DC [Tandberg Data ASA, 1996a]

Preamble består av et spesielt mønster som er spesifisert av QIC-5010-DC standarden. Dette brukes til å synkronisere VCO (Voltage Control Oscillator) i lese-elektronikken med datafrekvensen. Det er fire typer preamble:

1. Normal Preamble: Lagres foran hver blokk på sporet.
2. Elongated Preamble: Lagres i starten på den første blokken i en ramme som er lagt til allerede eksisterende data på et spor. Elongated Preamble blir alltid etterfulgt av en Normal Preamble.
3. Long Preamble: Lagres i starten på den første blokken på hvert spor. Long Preamble etterfølges alltid av en Normal Preamble.
4. Spacing Preamble: Ikke oppgitt av leverandør.

Se også tabell 6 på side 28 for et sammendrag av hvilke situasjoner de forskjellige typene Preamble benyttes i.

Block Marker identifiserer slutten på Preamble'n og starten på kontrollfeltet for hver blokk. Det består av et unikt mønster beskrevet i QIC-5010-DC standarden.

Control Field inneholder 8 kontroll bytes. Data og informasjonsblokker har et annet innhold enn ECC blokkene:

- Data og informasjonsblokker:
  - Blokk type
  - Komprimeringsinformasjon
  - Logisk blokk indikator
  - Logisk tidlig advarsel informasjon
  - Logisk blokkadresse fra vertsmaskin (32bit)
  - Fysisk blokknummer (16 minst signifikante bit)

- ECC blokker:
  - ECC for Byte 0 i kontrollfeltet til datablokker.
  - Sporsettnummer
  - *Filemark/Setmark* teller (16 bit)
  - Fysisk blokknummer (24 bit)

Datafeltet i hver blokk er av fast størrelse. Det består av 512 bytes som er lagret med NRZ1 (Non-Return to Zero, change on ONEs) metoden. For å unngå lange avstander på båndet uten flukstransisjoner (bare "0"-bits) blir informasjonen kodet i henhold til RLL 1,7 reglene. (Teori om NRZ1 kan finnes i [Stallings, 1994] og teori om RLL 1,7 finnnes i [Tandberg Data ASA, 1996a]). Datafeltet blir brukt på følgende måte:

- *Data Blocks*: Datafeltet inneholder 512 kodede data bytes til dataoverføring.
- *Filemark Blocks*: Blir brukt til å partisjonere båndet i filer.
- *Cancel Block*: Blir brukt lokalt til å slette (cancel) *Filemarks* på slutten av en lagring dersom det skal legges til mer data. Brukes også til å emulere overskrivningsfunksjoner i *TAR*-format (*UNIX*).
- *Setmark Block*: Brukes til å partisjonere båndet i *Sets*. Hvert *Set* kan igjen bli delt opp i *Files*, separert av *Filemark Blocks*.
- *EOD Block*: Indikerer End Of Data.
- *Filler*: Brukes til å fylle opp ufullstendige rammer slik at kun fulle rammer blir skrevet til båndet.

- *ECC*: (Error Correction Characters) Inneholder feilkorrigeringsinformasjon generert av stasjonen.
- *QIC Drive Identifier Block*: Inneholder informasjon om stasjonen og vertsmaskinen.
- *Media Management Block*: Inneholder informasjon om ødelagte deler av båndet.
- *Control Block*: Tilleggsblokk som kan brukes til leverandøravhengig informasjon.

CRC (Cyclic Redundancy Check) karakteren lages før dataene kodes. Følgende polynom blir benyttet:

$$P(x) = x^{32}+x^{28}+x^{26}+x^{19}+x^{17}+x^{10}+x^{6}+x^{2}+1$$

De fire CRC-tegnene blir skrevet med mest signifikante bit (bit 31) først. (Teori om CRC kan finnes i [Stallings, 1994]).

Postamble blir skrevet ihht. QIC-5010-DC standarden. Også her er det fire versjoner: *Normal, Elongated, EOD Marker* og *Media Header Postamble*. Tabell 6 viser et sammendrag av bruken av Preamble og Postamble for forskjellige skrivesituasjoner.

TABELL 6. Sammendrag av Preamble og postamble.

Type skriving	Preamble	Postamble
Normal data blokk i en fiksert ramme	Normal	Normal
Slutt på skriving	Normal	Normal + EOD Marker
Legg til - operasjon	Elongated + Normal	Normal
Første blokk på et spor	Long + Normal	Normal
Siste blokk på et spor	Normal	Normal + Elongated
Media Header område	Long+ Normal	Media Header

2. Posisjonering av lese/skrive-hodene

Mekanismen som støtter posisjonering av lese/skrive-hodene består av to funksjoner:

1. En *step-mekanisme* som plasserer det magnetiske hodet på magnetbåndet med oppløsning 0,0033mm. Det benyttes en dobbel-skrue (worm gear) kontrollert av en step-motor. Denne mekanismen benyttes for alle bånd, enten de er med eller uten servo-spor.
2. En *servo-mekanisme* som flytter hodet i området +-0,075mm. Denne mekanismen benyttes for å holde hodet i korrekt posisjon i forhold til de på forhånd skrevne servo-sporene. Dette gjør at lese- og skrive-operasjonene blir insensitive til relativ, tverrgående bevegelse mellom båndet og hodet. Det benyttes en 'voice coil' basert lineær motor med minimale magnetiske spredningsfelt, slik at den ikke ødelegger dataene på båndet.

Det er verd å legge merke til at for QIC-5010-DC (som har servo-spor) er det kun 10 trinn (steps) mellom hvert spor. For andre QIC-standarder uten servo-spor er det typisk fra 40 til 119 trinn mellom nabosporene. Servo-mekanismen er dermed hovedgrunnen til at QIC-5010-DC får plass til så mange som 144 data-spor på en kvart tommes båndbredde.

3. Partisjonering

Et bånd kan deles inn i flere partisjoner. Hver partisjon har sin egen BOP, opptaksområde, tidlig-advarsel-område[6] og EOP. For den første partisjonen (partisjon null) på et bånd med *n* partisjoner er BOP identisk med BOM. For den siste partisjonen (partisjon *n-1*) er EOP identisk med EOM. Partisjonering av et bånd er bare tillatt når det er posisjonert ved BOM. Partisjo-

6. Early-Warning Area: Dette er et merke som gir båndstasjonen en indikasjon på at den nærmer seg slutten på opptaksområdet. Båndstasjonen har da nok plass igjen til å skrive ut eventuelle data i bufferet.

nering kan resultere i at eksisterende data mistes (dersom eksisterende og ny partisjon er forskjellig slettes dataene).

Alle bånd har minimum en partisjon kalt partisjon null eller *default partition*. Når et bånd blir satt i båndstasjonen blir det posisjonert ved starten av partisjon null. Når en REWIND kommando sendes, spoles båndet tilbake til starten av den gjeldende posisjonen.

MLR1 allokerer minimum 2 sporsett (4 enkeltspor) til en partisjon. Siden det er totalt 144 spor på et QIC-5010-DC bånd, leder dette til at maksimalt *144/4 = 36* partisjoner er tillatt med en minimumsstørrelse på *13000MB/36 = 361MB*.

Alle partisjoner starter på BOT-siden av båndet. Dette betyr at båndstasjonen har mulighet til å aksessere enhver partisjon med minimal båndbevegelse etter at et nytt bånd er lastet inn.

4. Båndets bevegelse

Denne typen båndteknologi baserer seg på *streaming-prinsippet*, dvs. at båndstasjonen er designet for å kjøre hele båndlengden, normalt sett uten avbrudd. Unødvendige start og stopp operasjoner senker ytelsen betraktelig, og dersom man får for mange start/stopp over en kort båndlengde kan det føre til at båndet blir slakt. Dette vil degradere ytelsen dramatisk.

Båndet kan leses med en hastighet på 120 eller 60 ips[7]. Normalt vil båndet leses med 120 ips, men i tilfeller da vertsmaskinen ikke klarer å levere data fort nok kan båndstasjonen redusere hastigheten til 60 ips. Dette gjøres for å prøve og opprettholde streaming, det vil si å unngå at båndet må stoppes.

7. 120 ips = 3,05 m/s

For start- og stopptider oppgir Tandberg maksimalverdiene gjengitt i tabell 7.

TABELL 7. Maksimalverdier for start- og stopp-tider [Tandberg Data ASA, 1996a].

	Oppstartstid (ms)	Stopptid (ms)
60 ips	320	240
120 ips	425	450

5. Skriveoperasjoner

Data sendes fra vertsmaskina til båndstasjonen og lagres i stasjonens data-buffer[8]. Her blir dataene satt sammen i blokker på 512 bytes. Båndstasjonen legger til nødvendig adresseinformasjon og feilsjekkings-tegn før hele blokken skrives til båndet. Under skriving utføres read-while-write[9] sjekking, og blokker med feil blir automatisk skrevet om igjen lengre ned på båndet. Dette medfører at det blir skrevet ulikt antall blokker på hvert sporsett, avhengig av hvor mange dårlige blokker som oppstår og dermed må skrives om.

Merk også at QIC-5010-DC ikke benytter seg av sletting; gamle data skrives bare over av nye.

8. Lese og skrivebufferets kapasitet er på 1MB.
9. 'Read-while-write' henspeiler på at båndstasjonen leser blokken umiddelbart etter at den er skrevet for å sjekke at skriveoperasjonen gikk greit.

6. Leseoperasjoner

I lesemodus vil data leses fra båndet, adresse- og feilsjekkings-tegn fjernes fra blokkene og dataene blir overført til vertsmaskina via det innebygde databufferet til båndstasjonen. MLR1 kan maksimalt overføre 1,5 MB/s uten kompresjon. Med gjennomsnittlige data og kompresjonen på kan overføringsraten komme opp mot det doble. Det er viktig å legge merke til at MLR1 leser med samme hastighet som den spoler. Dette er en klar forbedring fra den forrige båndstasjonen til Tandberg, TDC4220, som hadde en hastighet for spoling og en lavere hastighet for lesing og skriving.

KAPITTEL 4

Søking på serpentinbånd

En forutsetning for å kunne beregne og planlegge aksesser på serpentinbånd er at det finnes en modell av datablokkenes plassering på båndet. En slik modell ble utviklet i forbindelse med en artikkel presentert på Norsk Informatikkonferanse på Voss, høsten 1997. Dette kapittelet oppsummerer mye av innholdet i denne artikkelen, som ble skrevet av Sandstå, Andersen, Midtstraum og Sætre [1997]. I tillegg er kapittelet om forberedelser basert på arbeider jeg utførte i forbindelse med et vårprosjekt [Andersen, 1997].

1. Forberedelser

For i det hele tatt å ha noe å søke på er det nødvendig å fylle et bånd med data. Det er dermed nødvendig å bestemme seg for hvor store datamengder som skal overføres i en skriveoperasjon. Dette kalles en logisk blokk, og blokkstørrelsen ble valgt til 32kB. 32kB er den største mengden data som kan overføres i en SCSI-kommando for enkelte systemer, og det er gunstig å

velge en blokkstørrelse av noe omfang for å minimalisere overhead ved data-overføringen.

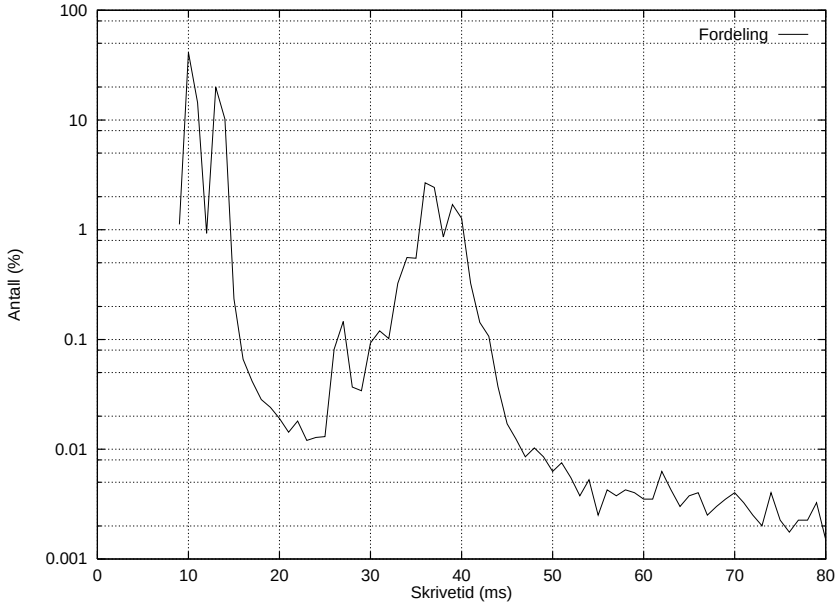
Et bånd ble skrevet helt ut med denne blokkstørrelsen ved hjelp av et lite skriveprogram (se “VEDLEGG 3: Kildekode”), som benyttet seg av et lavnivå tape/SCSI-bibliotek utviklet av Sætre i forbindelse med [Sætre, 1996]. Alle programmer som benyttet båndstasjonen og som er beskrevet i denne hovedoppgaven, benyttet seg av dette biblioteket. Komprimeringen ble på forhånd slått av, og kun en partisjon (default partition) ble benyttet. Det tok ca. 2,5 timer å skrive hele båndet[1].

Loggingen av skrivetidene fremviste følgende karakteristikker:

- Ved sporskifte gjør skrivetidene et veldig hopp. Dette skjer grovt sett rundt hver 5500 blokk og tilsier at det er omtrent så mange blokker på hvert sporsett. At skrivetidene blir så store akkurat ved sporskifte kommer av at stasjonen skriver så mye av blokka som den får plass til, deretter stoppes båndet, hodet reposisjoneres over neste spor, båndet startes igjen, og resten av blokka skrives.
- For blokkene som skrives mens stasjonen streamer[2] ligger skrivetidene på mellom 10 ms og 14 ms med topper på rundt 40 ms for hver åttende blokk.

Fordelingen av skrivetidene for en full kassett er gjengitt i figur 10. Legg merke til at y-aksen er logaritmisk. Denne framstillingsmåten er valgt fordi det ville blitt umulig å få vist forskjellene i fordelingen på en lineær skala. Toppen rett før 40 ms er ikke lett å forklare. Dette kan ha noe med måten båndstasjonen bufrer dataene på, men dette er kun gjetning.

1. I løpet av høsten har det blitt skrevet flere bånd, og gjennomsnittlig total skrivetid for et fullt bånd er funnet til å være 8865 sekunder, eller 2 timer, 27 minutter og 45 sekunder.
2. Det ble i forbindelse med [Andersen, 1997] også skrevet et bånd med en treg vertsmaskin som ikke klarte å levere data raskt nok til båndstasjonen. Dette resulterte i en del pussigheter når man studerte skriveloggen, blant annet skrudde båndstasjonen ned hastigheten til 60 ips med jevne mellomrom. For å få en dypere forståelse av hvordan båndstasjonen fungerer kan det anbefales å lese Vedlegg B i [Andersen, 1997].



FIGUR 10. Fordeling av skrivetider.

Tabell 8 viser noen statistiske data for søkeresultatene.

TABELL 8. Statistiske data for skrivetider.

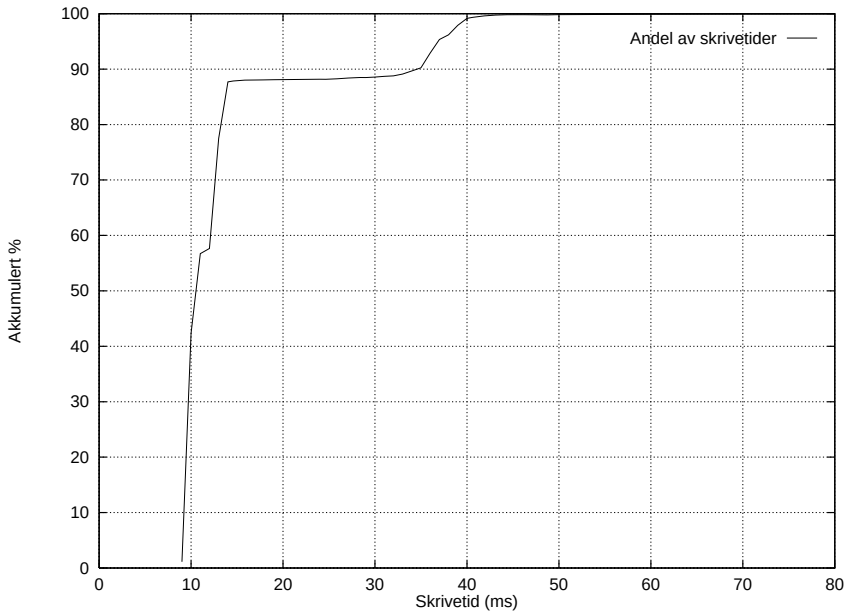
Minimum skrivetid	9 ms
Maksimum skrivetid	4418 ms
Gjennomsnittlig skrivetid	15 ms
Total skrivetid, fullt bånd	ca. 2,5 timer

For dette spesifikke båndet viser det seg at mer enn 87,7% av skrivetidene ligger i intervallet fra og med 9 ms til og med 14 ms. Høyeste prosentandel har 10 ms med 41,2%. Figur 11 viser en variant av figur 10, nemlig akkumulert prosentvis fordeling. Som man ser ligger over 99% av skrivetidene

under 40 ms. Tabell 9 oppsummerer skrivetidsfordelingen. Denne tabellen sier egentlig det samme som figur 11, bare med tall i stedet for en graf.

TABELL 9. Sammendrag av skrivetids-fordelingen.

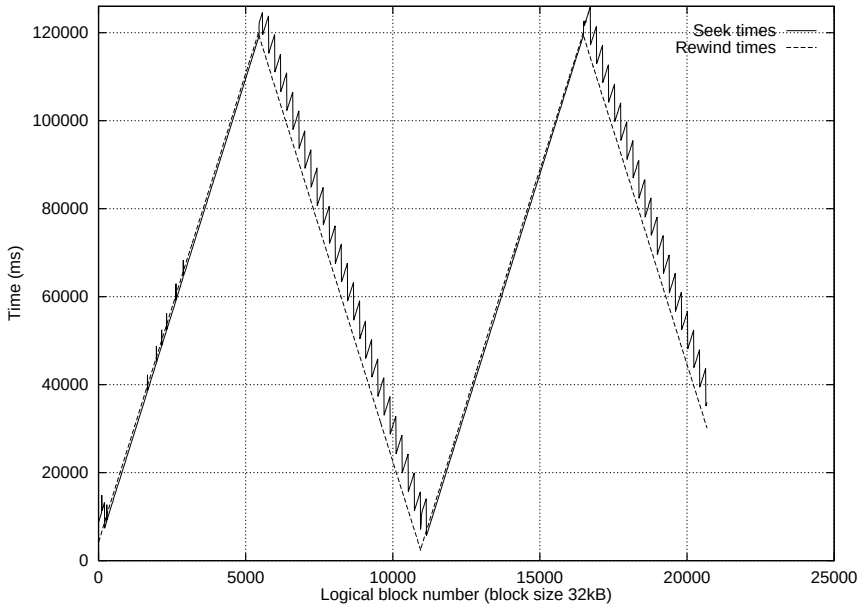
Intervall (ms)	Akkumulert %
9-14	87,7
9-35	90,2
9-40	99,2



FIGUR 11. Akkumulert prosentvis fordeling av skrivetidene.

2. Søking fra BOT

En karakteristikk som er interessant å studere, er hvordan søketidene forholder seg til blokknummerne når man søker fra starten av båndet (BOT). En slik studie ble gjort av Sandstå, Andersen, Midtstraum og Sætre [1997]. Tilsvarende forsøk har også blitt utført av Hillyer og Silberschatz [1996a, 1996b] samt av Sætre og Sandstå [1996].



FIGUR 12. Søke- og tilbakespolingstider for de første sporsettene på båndet [Sandstå, Andersen, Midtstraum og Sætre, 1997].

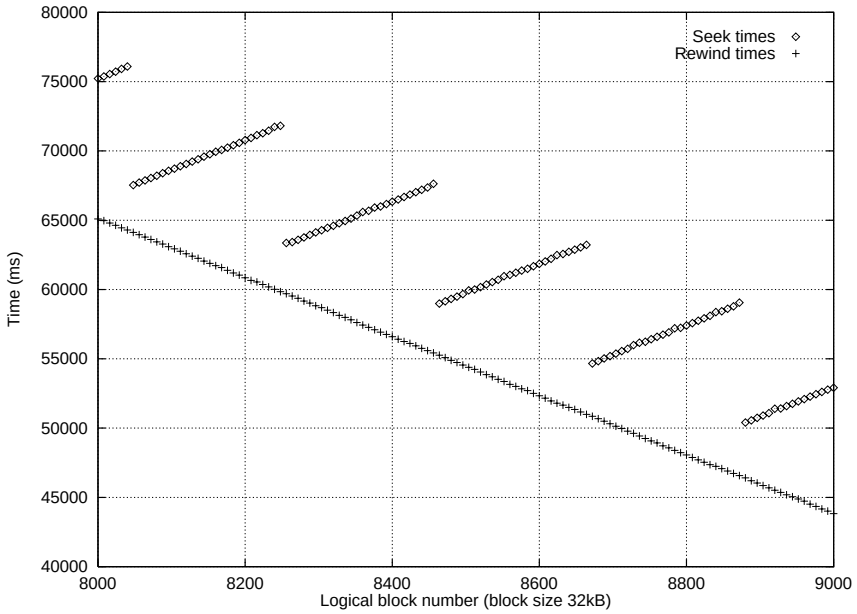
Figur 12 viser søke- og tilbakespolingstidene for de tre og et halvt første sporsettene på båndet. Langs horisontal-aksen vises logisk blokknummer mens vertikal-aksen viser antall millisekunder det tar å spole fra starten av båndet (BOT) til gitt logisk blokk. Disse tidene ble målt ved at man spolte båndet tilbake til BOT og deretter målte tiden det tok å søke til gitt posisjon. Tiden det tok å spole båndet tilbake til BOT ble også målt. Dette ble gjort for hver åttende blokk langs x-aksen[3]. Således er ikke testresultatene full-

stendige, men jeg anser dem likevel for å være mer enn gode nok til å gi et klart bilde av hvordan søketiden er avhengig av blokknummeret.

Fra figur 12 kan man se følgende:

1. For foroverspor er kurvene for søke- og tilbakespolingstidene både jevne og overlappende.
2. For bakoverspor har kurven for søketidene et sagtannmønster mens kurven for tilbakespoling er jevn og noe lavere.

For å forklare sagtannmønsteret på bakoverspor er det i figur 13 vist en forstørrelse av en liten del av bakoversporet i figur 12. Figur 13 viser at hver 'tann' består av en rett linje som representerer ca. 200 logiske blokker.



FIGUR 13. En liten del av første bakoverspor [Sandstå, Andersen, Midtstraum og Sætre, 1997].

3. Likevel kjørte båndstasjonen sammenhengende i ca. 2 dager for å lage denne grafen.

Grunnen til at dette oppstår er at for bakoverspor må stasjonen lese i motsatt retning enn for foroverspor. Når stasjonen står på starten av båndet og skal søke seg til en posisjon på et bakoverspor må den først spole over den blokken som skal leses for deretter å stoppe, snu retning, og lese tilbake til den finner blokken. Figur 13 indikerer at båndstasjonen benytter seg av et sett med predefinerte punkter til å bestemme seg for når den skal stoppe søket i foroverretningen og starte søket i motsatt retning. Disse punktene kalles *nøkkelpunkter* og korresponderer med den første blokken i hver sagtann. Hillyer og Silberschatz [1996a, 1996b] erfarte også sagtannmønster og nøkkelpunkter for en DLT 4000 stasjon. Forståelsen av begrepet 'nøkkelpunkter' er avgjørende for å kunne følge diskusjonen i de påfølgende kapitlene.

I tillegg til det initielle eksperimentet ble det også gjennomført flere søk fra BOT til tilfeldige posisjoner, samt søking fra en tilfeldig posisjon til en annen tilfeldig posisjon på båndet. Noen av resultatene er gitt i tabell 10, og disse er basert på omtrent 8000 forespørsler mot båndet.

TABELL 10. Søketider for søking fra starten av båndet (BOT) til en tilfedig posisjon, og fra en tilfeldig posisjon til en annen tilfeldig posisjon på båndet.

	Søking fra BOT	Søking fra tilfeldig posisjon
Minimum	4,02 sekunder	0,85 sekunder
Gjennomsnitt	65,4 sekunder	45,5 sekunder
Maksimum	126 sekunder	125 sekunder

Resultatene i tabell 10 sammen med kurvene som er presentert så langt i dette kapittelet danner grunnlaget for modellen som beskrives under.

3. Lavnivåmodeller for søking

Utviklingen av en modell for søking med MLR1 har pågått ved instituttet i omtrent et år nå. I de påfølgende underkapitlene gis det først en kort opp-

summering av arbeidet som ble gjort før eller parallelt med denne hovedoppgaven. Dette har i vesentlig grad handlet om å lage en modell for søking fra BOT til en tilfeldig posisjon. Deretter utvides modellen slik at den gjelder mer generelt for søking mellom tilfeldige posisjoner på båndet.

3.1 Tidligere arbeider

Sandstå, Andersen, Midtstraum og Sætre, [1997] ("VEDLEGG 2: Proceedings fra Norsk Informatikkonferanse, Voss 1997") presenterer utviklingen av en modell for søking fra BOT. Først lages en generisk modell basert på at det er et fast antall 32kB blokker per sporsett. Denne generiske modellen gir et uttrykk for *fysisk* posisjon til en gitt *logisk* posisjon på båndet. Uttrykket blir:

$$pos = |[(blokknummer + N) \ mod \ 2N] - N| \quad \textbf{(1)}$$

Der *N* er antall blokker på hvert sporsett.

For å uttrykke posisjonen som et forholdstall kan ligningen over deles på *N*. Dermed blir *pos* et tall mellom 0 og 1. Et matematisk uttrykk for søketiden med denne modellen vil være:

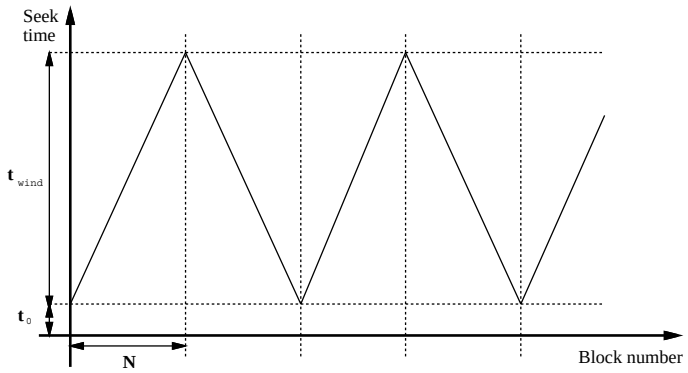
$$t = t_0 + \frac{t_{wind}}{N}|(pos_{stop} - pos_{start})| \quad \textbf{(2)}$$

Der t_0 er den minste tiden et søk kan ta, og t_{wind} er tiden båndstasjonen bruker på å spole båndet fra en ende til en annen.

For det tilfellet at man starter ved BOT forenkles uttrykket til:

$$t = t_0 + \frac{t_{wind}}{N} \cdot pos_{stop} \quad \textbf{(3)}$$

Modellen er presentert grafisk i figur 14.



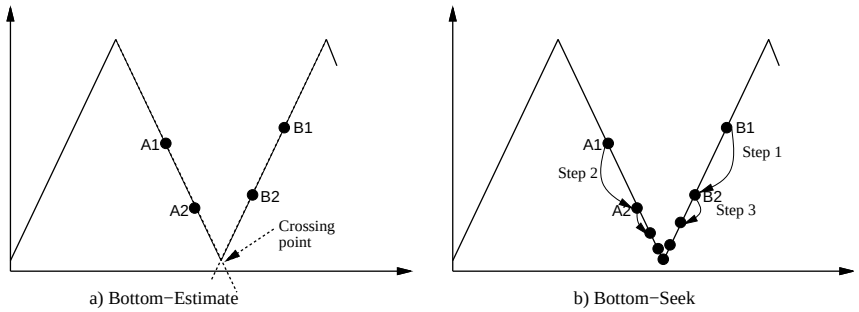
FIGUR 14. Generisk modell for karakterisering av serpentinbånd.

Denne modellen ble først validert i [Andersen, 1997] og senere i [Sandstå, Andersen, Midtstraum og Sætre, 1997] og [Sandståog Midtstraum, 1997]. Alle konkluderer med at resultatene med denne modellen ikke er gode nok, særlig fordi modellen har en tendens til å drive jo lenger ut på båndet man kommer. Gjennomsnittlig avvik ligger på mer enn 10 sekunder. Grunnen til det store avviket er at det ikke er hensiktsmessig å benytte en fast *N*, da antall blokker per sporsett varierer fra sporsett til sporsett og fra bånd til bånd.

For å bøte på problemet med drivingen ble det derfor i [Sandstå, Andersen, Midtstraum og Sætre, 1997] utviklet fire metoder for å karakterisere hvert enkelt bånd før man startet søkingen. Med karakterisering menes det å bestemme så nøyaktig som mulig antall blokker på hvert sporsett. Disse fire metodene ble kalt '*Exact tape-length*', '*Write-turn*', '*Bottom-estimate*' og '*Bottom-seek*'. En kort beskrivelse følger:

- *Exact tape-length* dividerer totalt antall blokker på båndet med antall sporsett og benytter dette som *N*.
- *Write-turn* undersøker skriveloggen og registrerer blokkene med lange skrivetider (> 2000ms) som startblokker på hvert sporsett.

- *Bottom-estimate* beregner endepunktene på odde sporsett (bunnene på pyramideplottet; herav navnet) ved å velge to punkter på hver side av antatt bunnpunkt og så måle tilbakespolingstidene fra disse punktene til BOT. Linjene som kan trekkes mellom disse punktene angir stigningslinjen for pyramideplottet, og man velger endepunkt der disse krysser hverandre.
- *Bottom-seek* er en variant av binærsøk og beregner også endepunktene på odde sporsett. Dette gjøres ved å velge et punkt på hver side av antatt bunnpunkt og måle tilbakespolingstidene. Punktet med høyest spoletid erstattes med et punkt som ligger mellom de opprinnelige punktene. Dette gjentas til man anser seg for å være nært nok til bunnpunktet.



FIGUR 15. Skisse av hvordan Bottom-estimate og Bottom-seek fungerer [Sandstå, Andersen, Midtstraum og Sætre, 1997].

Felles for *Exact tape-length* og *Write-turn* er at de trenger tilgang til skriveloggen fra den gang båndet ble skrevet, men de tar neglisjerbar tid å gjennomføre. For *Bottom-estimate* og *Bottom-seek* har man ikke behov for skriveloggen, men til gjengjeld tar de forholdsvis lang tid å gjennomføre (henholdsvis 75 og 131 minutter for hvert bånd).

I ettertid har Sandstå og Midtstraum [1997] kommet opp med en femte metode for karakterisering som de kaller '*Read-turn*'. Dette er en slags variant av '*Write-turn*' der det heller ikke er nødvendig med noen skrivelogg. '*Read-turn*' tar i gjennomsnitt 13 minutter å gjennomføre for hver tape.

Legg merke til at det er kun for den generiske modellen og '*Exact tape-length*' at ligning 2 gjelder. For alle de andre strategiene kan man ikke lenger snakke om én *N*, da hele vitsen her er at *N* varierer for hvert sporsett. For disse strategiene må man ved hjelp av karakteriseringsinformasjonen merke hver logiske posisjon med riktig sporsett-nummer og beregne et forholdstall for hver posisjon. Dette kan da uttrykkes som t_{karak} gitt ved uttrykket:

$$t_{karak} = t_0 + t_{wind}\left(\left|\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{start}}\right|\right) \quad \textbf{(4)}$$

I tillegg viser det seg at uttrykket i ligning 4 konstant underestimerer søketidene for bakoverspor. Dette kommer av snuoperasjonene og den ekstra spolingen som gir sagtennene. [Sandstå, Andersen, Midtstraum og Sætre, 1997] avsluttes derfor med å poengtere at det er hensiktsmessig å legge til en konstant kalt t_{rew} for bakoverspor. Ligning 5 gir dermed et uttrykk for en utvidet modell for søking med MLR1.

$$t_{karak} = \begin{cases} t_0 + t_{wind}\left(\left|\left(\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{start}}\right)\right|\right) & foroverspor \\ t_0 + t_{wind}\left(\left|\left(\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{start}}\right)\right|\right) + t_{rew} & bakoverspor \end{cases} \quad \textbf{(5)}$$

For å evaluere de forskjellige søkestrategiene ble det trukket 1000 tilfeldige forespørsler. Det ble så søkt fra BOT til disse forespørslene for alle strategiene og søketidene ble logget. Deretter ble det gjennomført 1000 søk mellom tilfeldige posisjoner. Alle disse testene ble kjørt på 3 forskjellige bånd. Tabell 11 oppsummerer karakteristikkene til de forskjellige strategiene. 'Gj.snittlig kostnad' refererer til tidskostnadene forbundet med karakterisering av båndet.

Merk! Det er kun nødvendig å karakterisere hvert bånd én gang; når båndet først er karakterisert kan man ta vare på denne informasjonen og benytte den om igjen neste gang båndet skal aksesseres.

TABELL 11. Resultater for testing av forskjellige strategier for modellering av båndstasjonen hentet fra [Sandstå, Andersen, Midtstraum og Sætre, 1997] og [Sandstå og Midtstraum, 1997].

Strategi	Gj.snittlig kostnad	Gj.snittlig feil ved søking fra BOT	Gj.snittlig feil ved søking mellom tilfeldige pos.
Utvidet modell	0 sekunder	12,0 sekunder	14,7 sekunder
Exact tape-length	0 sekunder[a]	5,35 sekunder	7,08 sekunder
Write-turn	0 sekunder[a]	2,30 sekunder	2,59 sekunder
Bottom-estimate	75 minutter	2,25 sekunder	Ikke testet
Bottom-seek	131 minutter	2,23 sekunder	Ikke testet
Read-turn	13 minutter	2,25 sekunder	2,89 sekunder

a. Avhengig av tilgang til skriveloggen

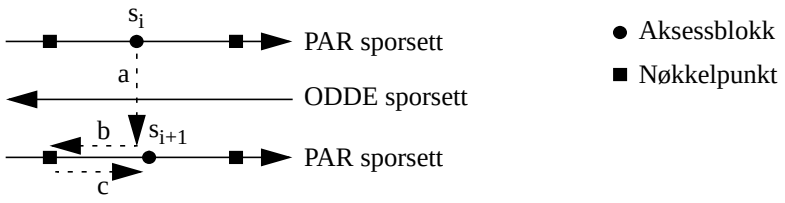
3.2 Videreutvikling av modellen

Modellen beskrevet så langt gir relativt gode resultater, hvilket går fram av tabell 11. For å kunne benyttes som grunnlag for planleggingsalgoritmer er den likevel ikke bra nok. Det tabellen ikke viser, er nemlig at modellen svikter når blokkene som skal aksesseres ligger (fysisk) tett på båndet. Hensikten med en planlegger er nettopp å reorganisere forespørslene slik at aksesstiden mellom hver forespørsel blir minimalisert. Dette må nødvendigvis medføre at blokkene reorganiseres slik at det er relativt liten fysisk avstand mellom dem.

I resten av dette kapittelet vil det derfor bygges en avansert modell som kan gi et godt estimat for søketidene mellom tilfeldige posisjoner uavhengig av avstanden mellom dem.

3.2.1 Shoeshining - en plagsom ulempe

Shoeshining er et begrep innen bånd-industrien som betegner den ugunstige situasjonen der båndet er nødt til å spole frem og tilbake over en kort distanse for å posisjonere hodet ved riktig blokk. Dette resulterer i en bevegelse der båndet 'pusser' fram og tilbake over lesehodet, og derav navnet. Shoeshining er illustrert i figur 16, og opptrer ved sporskifte når måladressen er fysisk nær den opprinnelige adressen. Det kan også inntreffe når man støter på dårlige områder på båndet, men det er en annen diskusjon.



FIGUR 16. Shoeshining; spoling for å få reposisjonert hodet.

I figur 16 kan man tenke seg at blokk s_i nettopp er aksessert og at man skal til blokk s_{i+1}, som ligger svært nære fysisk, men på et annet spor. Båndstasjonen må da stoppe og skifte spor, markert med pil *a*. Selv om hodet da egentlig står plassert nesten ved s_{i+1} må båndstasjonen først søke seg tilbake til nærmeste nøkkelpunkt illustrert ved pil *b.* Her må stasjonen snu og søke seg fram til s_{i+1} igjen, markert med pil *c*. Shoeshining strider altså klart med streaming-prinsippet, som er helt avgjørende for å optimalisere utnyttelsen av en båndstasjon.

3.2.2 Oppdeling av problemet med tilfeldige søk

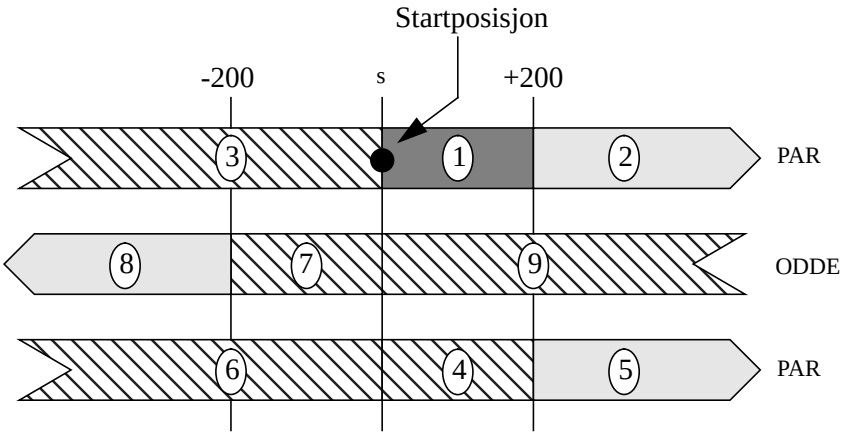
For å ta hensyn til shoeshining bør det skilles mellom følgende situasjoner som må kunne håndteres for tilfeldige aksesser:

Neste blokk ligger på:

1. Samme sporsett, i samme spoleretning og nære (< 200 blokker unna)
2. Samme sporsett, i samme spoleretning og langt unna(> 200 blokker)

3. Samme sporsett, i motsatt spoleretning
4. Et annet sporsett av samme type[4], i samme spoleretning og nære
5. Et annet sporsett av samme type, i samme spoleretning og langt unna
6. Et annet sporsett av samme type, i motsatt spoleretning
7. Et annet sporsett av forskjellig type, i motsatt spoleretning og nære
8. Et annet sporsett av forskjellig type, i motsatt spoleretning og langt unna
9. Et annet sporsett av forskjellig type, i samme spoleretning

Et eksempel på disse situasjonene er illustrert på figur 17. Skyggeleggingen henspeiler på hvilken del av ligning 6 (beskrevet under) som skal benyttes.



FIGUR 17. Ilustrasjon av de forskjellige situasjonene som må behandles for tilfeldige aksesser.

Som det går frem skilles det mellom blokker som ligger nære (< 200 blokker unna) og langt (> 200 blokker) unna. Dette er basert på estimatet om at det er ca. 200 blokker mellom hvert nøkkelpunkt, eller sagt på en annen måte: Hver sagtann er ca. 200 blokker lang. Poenget er altså å skille mellom blokker som kan ligge innenfor samme sagtann, og blokker som har minst et nøkkelpunkt mellom seg, og dermed ligger 'på en annen tann'.

4. Samme type henspeiler på enten par eller odde sporsett.

For tilfelle 1 regner man med at båndstasjonen ikke stopper, men leser neste blokk som skal aksesseres 'i farten'. De andre tilfellene benytter seg av modellen som er beskrevet i ligning 5, avhengig av om det er nødvendig med en snuoperasjon eller ikke (shoeshining). Et matematisk uttrykk for modellen som representerer søk mellom tilfeldige posisjoner kalles t_{rand} og er gitt i likning 6.

$$t_{rand} = \begin{cases} t_{wind}\left(\left|\left(\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{atart}}\right)\right|\right) & situasjon\ 1 \\ t_0 + t_{wind}\left(\left|\left(\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{start}}\right)\right|\right) & situasjon\ 2, 5\ og\ 8 \\ t_0 + t_{wind}\left(\left|\left(\frac{pos_{stop}}{N_{stop}} - \frac{pos_{start}}{N_{start}}\right)\right|\right) + t_{shoe} & situasjon\ 3,\ 4,\ 6,\ 7\ og\ 9 \end{cases} \quad \textbf{(6)}$$

Forsøk med MLR1 i [Sandstå, Andersen, Midtstraum og Sætre, 1997] viser at spoletiden for en hel tann er t_{saw} = 4 sekunder mens snutiden er t_{turn} = 2 sekunder. Det gjennomsnittlige tilfellet for t_{shoe} inntreffer når man må spole en halv tann tilbake, snu og spole en halv tann frem. Dette gir:

$$t_{shoe} = \frac{1}{2}t_{saw} + t_{turn} + \frac{1}{2}t_{saw} = 2 + 2 + 2 = 6\ sekunder \quad \textbf{(7)}$$

Verste tilfelle får man dersom begge aksessene ligger rett før et nøkkelpunkt. Dette gir t_{shoe} = 4+2+4 = 10 sekunder. I beste fall ligger begge punktene rett etter et nøkkelpunkt, hvilket gir t_{shoe} = 0+2+0 = 2 sekunder. Denne modellen innfører altså en teoretisk feilmargin på $\pm$ 4 sekunder.

KAPITTEL 5

Introduksjon til planleggings-strategier

Dette kapittelet tar først for seg momenter som gjelder for planleggings-strategier generelt. Deretter presenteres tre enkle typer planleggingsmetoder for søking på sekvensielle media. Disse kalles READ, FIFO og SORT. Idéen til å ta med disse metodene har sitt utspring i tidligere arbeider utført av Hillyer og Silberschatz [1996b]. Det kan synes drøyt å kalle READ og FIFO for 'planleggingsmetoder' ettersom de faktisk ikke benytter seg av en planlegger for å hente forespørslene. Likevel tas de med her ettersom det er velkjente, og som det skal vise seg, viktige strategier både for diskusjonen og som reelle alternativer. For enkelhets skyld begrenses diskusjonen i dette og de resterende kapitlene seg til å omfatte planlegging av forespørsler mot en enkelt båndstasjon der alle data ligger på et ferdig lastet bånd.

For at det ikke skal bli noen misforståelser defineres først noen begreper som går igjen i dette og senere kapitler:

- En *planleggingsmetode* er et begrep som kun beskriver at et sett, eller en mengde, med forespørsler behandles på en eller annen måte slik at det er mulig å aksessere båndstasjonen og hente ut dataene.

- En *planleggingsalgoritme* forutsetter at forespørslene blir kjørt gjennom en prosess som vurderer dem på en eller annen måte og omorganiserer dem etter gitte kriterier.
- *Planleggings-strategi* er et fellesbegrep for de to ovenforstående punktene.

1. Generelt om planleggings-strategier

Det er viktig å være bevist på hvilke kriterier som er avgjørende for planleggings-strategier. Derfor omtales dette først. Videre er det to ting som gjelder for alle planleggings-strategier: For det første er det nødvendig å definere en notasjon som gjør det mulig å beskrive dem på en fornuftig måte. For det andre er det behov for å bestemme hvilke start- og stopp-kriterier som skal gjelde for diskusjonen av ytelsen til algoritmene. I tillegg er det for denne hovedoppgaven hensiktsmessig å definere et bånd langs flere dimensjoner. Alle disse punktene behandles i det påfølgende.

1.1 Vurderingskriterier for planleggings-strategier

Et system som benytter seg av magnetbånd som tertiærlager bør ta hensyn til de uforholdsmessig lange søketidene som er forbundet med slike medier. En god planleggingsmekanisme (scheduler) som ordner forespørslene mot båndarkivet vil kunne redusere ventetiden for brukerne.

Det første som bør gjøres når man skal velge mellom flere algoritmer er å definere kriteriene som skal gjelde for utvelgelsen. Ettersom denne hovedoppgaven inngår i et prosjekt for lagring av video på digitale bånd, benyttes dette som eksempel i diskusjonen videre. Forskjellige kriteria kan være:

1. Kontinuerlig tilførsel av informasjon.
2. Minimalisering av gjennomsnittlig ventetid.

3. Mulighet for prioritering av enkelte forespørsler.
4. Unngå utsulting av forespørsler.

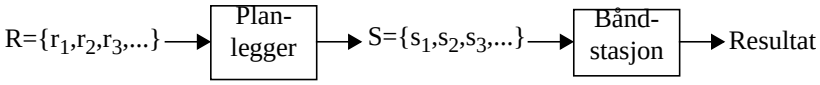
Prioriteringen av disse kriteriene kan variere med hva slags informasjon som skal lagres, det vil si hvilke typer forespørsler man kan forvente. For et videoarkiv som hovedsaklig skal benyttes til VOD må det antas at forespørslene er relativt få, men at hver forespørsel vil kreve lange overføringer. Overføringstiden vil derfor ofte være relativt mye lengre enn søketiden. For et videoarkiv som skal benyttes til lagring av metadata-intensiv informasjon, som for eksempel nyhetssendinger, vil det derimot være mange, relativt korte forespørsler. Dette vil dermed sette strengere krav til planleggeren.

Det første kriteriet anses for å være absolutt. Video må overføres kontinuerlig, ellers er det ikke brukbart. Det andre punktet er selve hensikten med å bruke en planlegger. Minimalisering av gjennomsnittlig søketid må imidlertid ikke gå på bekostning av punkt 1, så det er ikke sikkert at den optimale planleggingsalgoritmen kan benyttes. Viktigheten av det tredje kriteriet vil sterkt avhenge av hva slags forespørsler man ønsker. Prioritering av forespørsler vil jeg anse for å være et “kjekt å ha” kriterium, men ikke et absolutt krav. Det fjerde punktet henger sammen med hvordan man velger å implementere planleggeren. Det er to hovedmåter å gjøre dette på: Enten tar man en bolk med forespørsler, planlegger dem og henter dem uten at det tillates nye forespørsler å komme imellom. På denne måten vil man ikke kunne utsulte enkelte forespørsler. Eller så kan man tillate nye forespørsler å bli (plan)lagt inne i køen av allerede planlagte forespørsler. Dette vil i teorien kunne medføre at en forespørsel sultes dersom det ikke legges inn mekanismer (for eksempel prioritering som i punkt 3) for å forhindre dette. Punkt 4 må det altså tas hensyn til dersom man velger en algoritme som åpner for utsulting.

1.2 Notasjon

Hensikten med å sortere forespørsler er å minimalisere tiden det tar å hente ut en gitt mengde informasjon fra en kassett når dataene ikke ligger sekvensielt. Algoritmene under presenteres ved hjelp av samme notasjon som Hil-

lyer og Silberschatz [1996b] benyttet: La *I* betegne lese/skrive-hodets opprinnelige posisjon før planleggingen, *R* betegne listen av forespørsler som blir gitt til planleggingsalgoritmen, og *S* være den sorterte listen av forespørsler som blir produsert av planleggingsalgoritmen, det vil si at *S* er en permutasjon av *R*.



FIGUR 18. Visualisering av planleggings-systemet og notasjonen.

Figur 18 viser hvordan man tenker seg at denne planleggingen skal inngå i et system som minimaliserer aksesstidene mot for eksempel et digitalt videoarkiv.

I diskusjonen under forutsettes det at båndet det søkes på er fylt med blokker med fast størrelse. Videre antas det for enkelhets skyld at hver aksess omfatter en leseoperasjon som kun leser en blokk. En utvidelse til å la hver leseoperasjon omfatte flere blokker er triviell. Det eneste man i så tilfelle må ta hensyn til, er at startpunktet for aksess n er $s_{n-1}+x_{n-1}$, der x er antall blokker som skal leses. For lesing av kun én og én blokk blir startpunktet alltid $s_{n-1}+1$.

1.3 Start- og stopp-kriterier

Før man begynner å teste må man ta stilling til når man skal begynne å ta tiden og når man skal slutte, eller med andre ord hva man skal ta tiden på. Her kan det defineres fire alternativer med hensyn på lese/skrive-hodets posisjon ved start, *I*, og hodets posisjon ved stopp etter at alle aksessene er lest, her definert til *E*. En tilfeldig gitt posisjon på båndet beskrives med *i*:

1. Start tidsmålingen når hodet er ved BOT ($I=0$) og stopp tidsmålingen etter at hodet har returnert til BOT etter siste aksess ($E=I=0$).
2. Start tidsmålingen når hodet er ved BOT ($I=0$) og stopp tidsmålingen etter siste aksess[1] ($E=s_n+x_n$).

3. Start tidsmålingen når hodet står ved en tilfeldig posisjon på båndet ($I=i$) og stopp tidsmålingen etter at hodet har returnert til BOT etter siste aksess ($E=0$).
4. Start tidsmålingen når hodet står ved en tilfeldig posisjon på båndet ($I=i$) og stopp tidsmålingen etter siste aksess ($E=s_n+x_n$).

Grunner som taler for å starte tidsmålingen når hodet er posisjonert ved BOT:
Det antas at alle bånd som blir satt inn er spolt helt tilbake, det vil altså si at båndet er posisjonert ved BOT. Grunnen til dette er at båndstasjonen er nødt til å lese inn posisjoneringsinformasjon som ligger på starten av båndet når båndet blir lastet.

Grunner som taler for å starte tidsmålingen når hodet er ved en tilfeldig posisjon på båndet:
Dette vil være en konsekvens av punkt 2 eller 4 der man stopper med en gang siste aksess er lest. Etter at en gruppe forespørsler er ferdig vil man da befinne seg et tilfeldig[2] sted på båndet. Dersom det da ligger klar en ny gruppe forespørsler til behandling vil det være sløsing av tid å spole tilbake til BOT først. Planleggings-strategiene kan registrere nåværende posisjon og relativt enkelt ta hensyn til dette ved sorteringen av forespørsler. Man vil maksimalt tape tiden det tar å snu leseretning en gang, og dette er sannsynligvis langt mindre enn kostnaden ved å spole tilbake til BOT.

Grunner som taler for å stoppe tidsmålingen etter siste aksess:
Dette henger mye sammen med begrunnelsen for å starte tidsmålingen når hodet er ved en tilfeldig posisjon på båndet, som beskrevet ovenfor. I tillegg kan man se på det slik at for brukeren er prosessen over når man har fått lest siste objekt.

1. s_n er adressen til siste (n'te) forespørsel. I tillegg må det legges på x_n antall blokker som skal leses for siste aksess.
2. Hvor "tilfeldig" dette er vil avgjøres av en kombinasjon av hvilken planleggings-strategi som benyttes og hvor mange forespørsler man nettopp har behandlet. Dette omtales nærmere under beskrivelsen av de enkelte algoritmene.

Grunner som taler for å stoppe tidsmålingen etter at hodet har returnert til BOT etter siste aksess:
Dette alternativet vil kun ha mening dersom det benyttes sammen med tidtakings-start fra BOT. Ved å benytte denne strategien vil båndet alltid være spolt tilbake til BOT etter en gruppe forespørsler, og det vil da være raskt å skifte bånd.

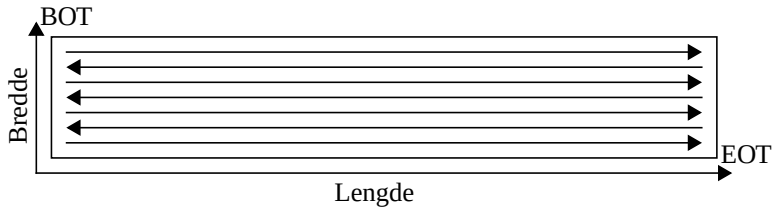
Det bestemmes at testene skal gjennomføres i henhold til tidtakingsstrategien i punkt 2. Årsaken til dette er at det er denne situasjonen man møter når man har lastet et nytt bånd, og det gjør implementeringen av planleggingsalgoritmene enklest mulig. For et reelt system vil en konsekvens av stoppkriteriet være at man taper tid dersom det er flere forespørsler som venter etter at en gruppe er ferdig. I dette tilfellet kommer da tilbakespolingstiden til BOT i tillegg til total aksesstid for neste gruppe forespørsler, og det vil medføre at første aksess i denne gruppen blir dårligere. Denne uønskede effekten forsvinner dersom det ikke er noen forespørsler som venter når man er ferdig. Da kan man spole tilbake til BOT i ro og mak uten at det har noen innvirkning for brukeren. Her må man altså se på hvor ofte en gruppe forespørsler ankommer. Ved hyppige ankomster og stor belastning vil alternativ 4 være bedre egnet, og en interessant utvidelse av forsøkene som skal gjennomføres ville vært å basert tidtakingen på dette alternativet.

1.4 Båndet i flere dimensjoner

Det er flere måter å betrakte et serpentinbånd på. Man kan for eksempel se på det som en eneste lang sekvens av logiske posisjoner fra BOT til EOT. Dette er den naturlige måten en bruker vanligvis betrakter båndet på. En slik tilnærming, der man kun betrakter aksessenes logiske posisjoner, defineres i denne hovedoppgaven til å være *null-dimensjonal*.

“KAPITTEL 4: Søking på serpentinbånd” viste at for utvikling av modeller for båndstasjonen er det nødvendig å omregne båndets logiske blokknummer til fysiske blokknummer. På denne måten ‘mapper’ man alle de logiske posisjonene til å være innenfor et område avgrenset av det antall blokker som går på et sporsett, eller sagt på en annen måte; innenfor båndets fysiske

lengde. En slik tilnærming der man betrakter aksessenes fysiske posisjoner, men ser bort fra hvilke sporsett de ligger på, kan derfor sies å være *én-dimensjonal* i båndets lengderetning.



FIGUR 19. Båndet definert langs to dimensjoner; lengde og bredde.

Det er mulig å gå videre og ta hensyn til blokkenes plassering på forskjellige sporsett, eller i bredderetningen som vist på figur 19. Dersom man innfører kostnader i forbindelse med sporskifte kan man ikke lenger bare organisere forespørsler etter fysiske posisjoner. I dette tilfellet blir det nødvendig å utvikle matematiske uttrykk for tidskostnadene forbundet med flytting fra en aksess til den neste. En slik tilnærming der man betrakter aksessenes fysiske posisjon både i lengde- og bredde-retningen, samt beregner tidskostnader forbundet med *locate*-operasjoner, kalles *to-dimensjonal*.

I denne hovedoppgaven vil det gradvis presenteres testresultater fra enkle, null-dimensjonale planleggings-metoder til en relativt kompleks to-dimensjonale planleggings-algoritme.

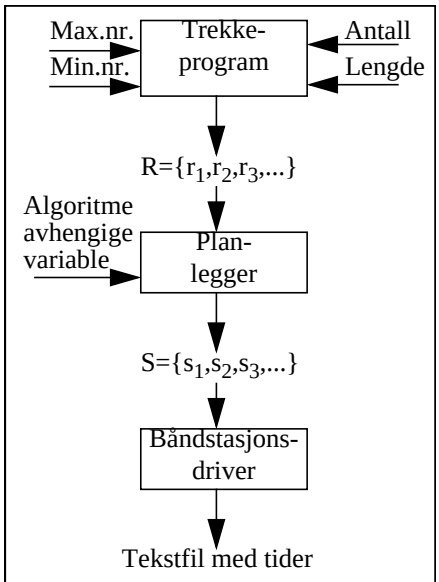
2. Implementerings-strategi

Implementeringen av de forskjellige programmene har blitt gjort i *C++* og kompilert med *Gnu*[3] sin kompilator (*g++*). Lavnivå-koden som styrer overføringen av data mellom vertsmaskinen og båndstasjonen var allerede laget

av Sætre [1996]. Dette biblioteket har blant annet støtte for skriving og lesing av blokker på 32kB, søking til en gitt posisjon, og tilbakespoling til BOT (rewind). I tillegg gir biblioteket mulighete for å skru av komprimeringen. Biblioteksfunksjonene er å finne i fila *tapestreamer.h* som er gjengitt i “VEDLEGG 3: Kildekode”.

Siden det er flere planleggingsalgoritmer som skal testes ut, er det viktig å ha en klar strategi for gjennomføringen for å spare tid. Det velges derfor å satse på en tredelt arkitektur for test-strategien der hvert trinn kan utføres separat, men der neste trinn avhenger av det forrige. Følgende test-strategi har blitt valgt:

1. Implementer et trekkeprogram som genererer et gitt antall tilfeldige forespørsler innenfor et bruker-bestemt intervall.
2. Implementer forskjellige planleggere som tar inn resultatet av punkt 1 og ordner forespørslene i henhold til beskrevne algoritmer. Mål tiden det tar å sortere forespørslene med algoritmene.
3. Implementer en båndstasjonsdriver som tar inn en liste med forespørsler, aksesserer disse i rekkefølge, og logger posisjonerings- og tidsforbruk-informasjon til en tekstfil.



FIGUR 20. Test-strategi.

Figur 20 gir en grafisk fremstilling av denne strategien.

Trekkeprogrammet er meget enkelt og lar brukeren spesifisere laveste og høyeste tillatte blokknummer, antall forespørsler som skal trekkes og maksimal lengde på segmentet som skal overføres oppgitt i et helt antall blok-

3. Gnu project C++ compiler; Copyright © Free Software Foundation Inc.

ker. Programmet genererer en tekstfil med to kolonner. Dette er symbolisert som en liste *R* med forespørsler $\{r_1,r_2,r_3,...\}$ på figur 20. Den første kolonnen i tekstfila er blokknummeret det skal søkes til, den andre kolonnen er antall blokker som skal leses. For alle listene gjaldt det at overføringssegmentet ble satt til kun en blokk, og det ble trukket tilfeldige forespørsler i intervallet [0..385000][4].

Beskrivelsen av implementeringen for de forskjellige planleggerne utsettes til de respektive planleggings-avsnitt som følger.

Båndstasjonsdriveren er et enkelt program som benytter lavnivå tape/SCSI-biblioteket til Sætre [1996] for å aksessere båndstasjonen. For å kunne validere testene, samt gjøre 'debugging' av planleggerne, ble følgende parametre logget for hver aksess:

1. Fysisk posisjon
2. Startposisjon (nåværende posisjon)
3. Målposisjon (posisjon å søke til)
4. Differansen mellom startposisjon og målposisjon
5. Målt søketid fra startposisjon og målposisjon
6. Antall blokker i overførings-segmentet (antall blokker som ble lest)
7. Målt lesetid for overførings-segmentet
8. Sporsett-nummer[5]
9. Beregnet søketid fra startposisjon til målposisjon[5]

4. Det er ca. 400.000 blokker på et fullt bånd. Det viser seg at de siste blokkene på båndet er forbundet med en del mer uregelmessigheter enn for resten av båndet. Dette er sannsynligvis et resultat av at båndstasjonen begynner å forberede seg på at det går mot slutten når den oppdager såkalte Early-Warning Areas. For denne hovedoppgaven var det ikke ønskelig å bli belastet med denne ekstra støyen i målingene, og derfor ble det valgt å ikke aksessere blokker utover de 385.000 første.
5. Dette ble kun logget for den siste algoritmen, SLTF, og er beskrevet i et senere kapittel.

3. Valg av forespørselsmengde

Med forespørselsmengde menes hvor mange forespørsler som skal planlegges på en gang. Hvor mange forespørsler er det realistisk å tenke seg at man må behandle i et arkivsystem? Det er ikke lett å svare på dette, men det må være rimelig å anta at man i et videoarkiv ikke kan tillate alt for mange, da dette vil gi uakseptabelt lange aksesstider. I tillegg er det lite sannsynlig at det i et videoarkiv vil være særlig mange "interessante" klipp på et bånd. Om man derimot tenker seg et bildearkiv, vil man trolig kunne akseptere flere forespørsler ettersom informasjonen som skal overføres ikke vil være alt for stor. Av samme grunn vil også antall "objekter", i dette tilfellet bilder, per bånd være mye større enn i et videoarkiv.

For denne undersøkelsen er det ønskelig å gi en best mulig karakteristikk av de forskjellige planleggings-strategiene som behandles. Det er derfor gunstig om man kunne gjennomføre planlegging for et bredt spekter av forespørselsmengder slik at det blir enklere å sammenligne de forskjellige strategiene. I utgangspunktet ble det bestemt å trekke lister med forespørselsmengde basert på log_2. Det er senere blitt trukket lister på 6, 12 og 768 forespørsler i tillegg for å få glattere kurver. I løpet av testperioden har det vist seg at forespørselsmengder innenfor intervallet [4..2048] er tilstrekkelig for å vise tendensene med de strategiene som er testet. Dette kommer til syne når testresultatene blir presentert senere i denne rapporten.

4. Null-dimensjonale planleggings-strategier

Som beskrevet tidligere i dette kapittelet kan man basere planleggingen av forespørsler på sortering av logiske blokkadresser, eller rett og slett la være å planlegge i det hele tatt. Begge deler går inn under null-dimensjonal planlegging, og er de enkleste alternativene, om ikke nødvendigvis de beste.

4.1 READ-metoden

READ metoden leser hele båndet sekvensielt fra BOT (den logiske starten av båndet) til EOT (logisk slutt). Dette betyr at det ikke er nødvendig å planlegge I/O, og det er heller ikke nødvendig med noen *locate* operasjon. Det tar ca. 148 minutter å lese et helt QIC-5010-DC bånd. Dette kan synes som et uaktuelt alternativ, men for tilstrekkelig antall forespørsler vil READ gi beste resultat, hvilket vil komme fram etterhvert.

Det kan argumenteres med at det ikke er nødvendig å lese lenger enn til aksessen med høyeste adresse har blitt lest. På den måten vil man for få forespørsler kunne redusere kjøretiden en del. Likevel er READ så håpløs for få forespørsler at det er liten vits i å dvele ved dette. (For én forespørsel vil man i gjennomsnitt måtte lese halve båndet). Det ble derfor besluttet å droppe testing av READ, og i stedet la plottet av den være en beregnet kurve som rett og slett ligger fast på 8865 sekunder. For gjennomsnittlig aksesstid kan man plotte kurven ved å dele 8865 sekunder på antall aksesser.

4.2 FIFO-metoden

FIFO står for *First In, First Out*. Denne metoden utfører locate operasjoner og leser dataene i samme rekkefølge som de blir presentert. Ingen reorganisering av rekkefølgen på forespørselene finner sted; $S = R$.

Ytelsen til FIFO er forutsigbar. Fra “KAPITTEL 4: Søking på serpentinbånd” vet man at gjennomsnittlig søketid fra BOT er 65 sekunder, mens gjennomsnittlig søketid fra en tilfeldig posisjon til en annen er 46 sekunder. Tidsforbruket til FIFO kan derfor estimeres til $65 + n \cdot 46$ sekunder for de tilfellene der man starter ved BOT og skal aksessere n forespørsler.

FIFO er så grundig testet i [Sandstå, Andersen, Midtstraum og Sætre, 1997] (8000 forespørsler) at det anses for å være unødvendig med ytterligere testing. I tillegg er FIFO såpass tidkrevende at det er bedre å bruke tid på å teste de mer aktuelle algoritmene som er beskrevet i de neste kapitlene. FIFO vil derfor bli plottet ut fra et estimert tidsforbruk, som beskrevet over.

4.3 SORT-algoritmen

SORT ordner forespørslene etter logisk blokknummer, og er den første algoritmen som presenteres. For båndstasjoner som benytter roterende hode teknologi vil dette være den mest effektive sorteringsalgoritmen da logisk blokknummer tilsvarer fysisk posisjon på båndet. Kompleksiteten er *O(n log n)*. For serpentinbånd vil SORT være uegnet for små *n*, men forholdsvis god hvis *n* blir så stor at det er mange aksesser på hvert spor. Det finnes flere alternative sorteringsalgoritmer, blant annet *INSERTION SORT, HEAP SORT, QUICK SORT, RADIX SORT og BUCKET SORT* [Cormen, Leiserson og Rivest, 1992]. For lange lister vil det være nødvendig å benytte en "smart" algoritme med lav sorteringstid. Forespørselslistene som skal behandles i denne hovedoppgaven er imidlertid ikke større enn at den enkleste algoritmen, *INSERTION SORT*, klarer seg bra.

SORT ble implementert og testet mot båndstasjonen. I første omgang ble det trukket ut fem lister med tilfeldige forespørsler for henholdsvis 16, 32, 64 og 128 forespørsler. Kjøringen av disse listene viste en tendens, men det var tydelig at dette ga et for tynt grunnlag. Det har derfor i ettertid blitt kjørt flere lister for de samme forespørselsmengdene, samt lister med andre mengder. Tabell 12 oppsummerer det statistiske grunnlaget og resultatene som fremkom under forsøkene med SORT. Kolonnen med gjennomsnittlig aksesstid i sekunder er bare en beregning av gjennomsnittlig totaltid dividert på antall forespørsler..

TABELL 12. Statistisk grunnlag og resultater for SORT .

Antall forespørsler	Antall lister	Gjennomsnittlig totaltid (s)	Gjennomsnittlig aksesstid (s)
4	40	200	50
6	40	289	48
8	40	362	45
12	40	570	48
16	40	720	45
32	40	1369	43

TABELL 12. Statistisk grunnlag og resultater for SORT (Fortsatt).

Antall forespørsler	Antall lister	Gjennomsnittlig totaltid (s)	Gjennomsnittlig aksesstid (s)
64	15	2717	42
128	5	4669	36
256	1	6444	25
512	1	7985	16
768	1	8287	11
1024	1	9026	9
2048	1	9929	5

Cpu-kostnaden for å planlegge 2048 forespørsler med SORT slik den er implementert her er 2,5 sekunder[6]. Dette må anses for å være neglisjerbart for tiden til første aksess og i alle fall for totaltiden

Når det gjelder antallet lister som er trukket for hvert antall forespørsler er dette basert på følgende punkter:

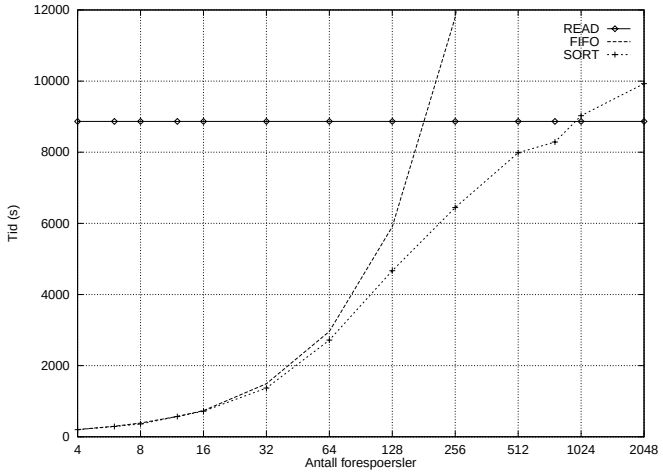
- Det er selvfølgelig raskere å kjøre tester med lister som inneholder få forespørsler. Man kan for eksempel kjøre 50 lister med 4 forespørsler på samme tid som man kan kjøre én liste med 2048 forespørsler. Det ville jo vært fint om man hadde rukket å teste med mange flere lister, men tiden strekker ikke til. Dette er den ene grunnen til at det ble valgt å kjøre et større antall lister med få forespørsler enn for lister med mange forespørsler.
- Den andre, og kanskje vesentligste, årsaken til listefordelingen er at det kreves mange lister med få forespørsler for å få et statistisk brukbart resultat. Dette kommer av at disse listene er veldig utsatt for den tilfeldige trekningen. For eksempel varierer totaltidene for lister med 4 forespørsler fra 90 sekunder til 324 sekunder; en forskjell på 260%. Lister med 128 forespørsler derimot, resulterer i totaltider fra 4396 sekunder til 4843 sekunder; en forskjell på 10%. De store listene er også utsatt for

6. Målt på *Raudeik*; en *Sun IPX 40MHz MicroSparc* med 32MB RAM.

trekningen, men i langt mindre grad. Dette er fordi forskjellen mellom lange og korte aksesser blir mindre (jo flere aksesser, dess kortere blir det mellom dem) og de forskjellene som er vil ha en tendens til å "jevne" seg ut i løpet av testen.

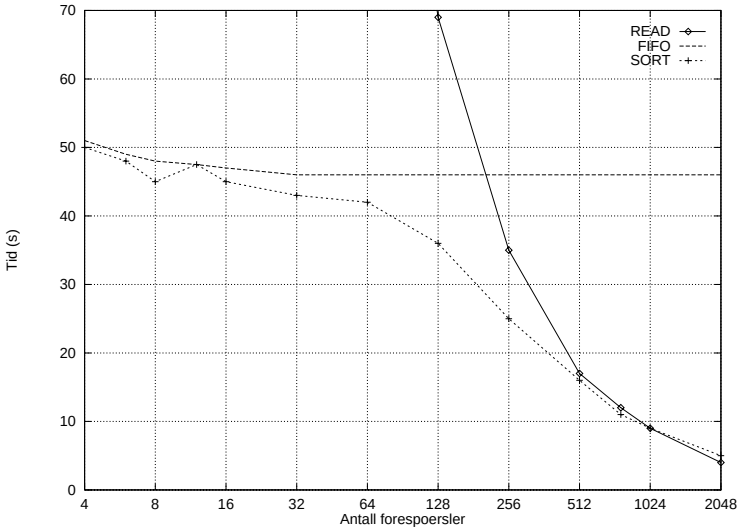
4.4 Sammenligning av READ, FIFO og SORT

Figur 21 og figur 22 viser et plot av henholdsvis gjennomsnittlig totaltid og gjennomsnittlig aksesstid for planleggings-strategiene beskrevet så langt.



FIGUR 21. Gjennomsnittlig totaltid som en funksjon av antall forespørsler for READ, FIFO og SORT.

Legg merke til at plottene for READ og FIFO er basert på beregninger og ikke på tester utført i løpet av denne hovedoppgaven. Av begge figurene går det fram at FIFO og SORT oppfører seg rimelig likt for få forespørsler. Dette virker også rimelig da en sortering av for eksempel 6 forespørsler sannsynligvis bare resulterer i en liste som er nesten tilsvarende tilfeldig hva fysiske posisjoner på båndet angår. Det er først når vi kommer opp i lister av noe størrelse (> 64 forespørsler) at SORT fremviser betydelig bedre ytelse enn FIFO. For slike mengder forespørsler øker sannsynligheten for at flere av dem ligger på samme sporsett og nære hverandre.



FIGUR 22. Gjennomsnittlig aksesstid for hver forespørsel gitt som en funksjon av antall forespørsler for READ, FIFO og SORT.

Uregelmessighetene for SORT-plottet (figur 22) ved små forespørselsmengder tyder på at det ikke har blitt gjennomført nok tester til å få et entydig statistisk resultat. Det ble ikke tid nok til å kjøre flere trekkelister for SORT, men selv om kurven ikke er helt glatt viser den en klar tendens.

For store forespørselsmengder ser det ut på figur 22 som om SORT og READ faller sammen. Dette er imidlertid ikke en korrekt slutning, hvilket kommer fram på figur 21. For så store forespørselsmengder som 2048 varierer gjennomsnittlig aksesstid bare med ca. et halvt sekund, men det utgjør likevel ca. 17 minutter på totaltiden. Hvorfor er READ bedre enn SORT for fler enn 1024 forespørsler? Grunnen til dette er at READ leser båndet med kontinuerlig (maksimal) hastighet, det vil si at READ streamer hele veien. SORT derimot, vil streame i de tilfellene blokkene ligger svært tett på samme sporsett, men vil stoppe opp for hvert søk som ligger mer enn 200 blokker unna som beskrevet i “KAPITTEL 4: Søking på serpentinbånd”. På den måten får man akkumulert opp en mengde t_0 som gjør at SORT etterhvert taper i forhold til READ.

KAPITTEL 6

SCAN-algoritmer; Én-dimensjonal planlegging

Av forrige kapittel gikk det fram at SORT og READ ga brukbare resultater for planlegging av mange forespørsler. For få forespørsler var de derimot ikke gode. SCAN-algoritmene søker å bedre ytelsen ved å innføre én-dimensjonal planlegging, altså planlegging basert på aksessenes fysiske posisjoner i lengderetningen.

SCAN-algoritmen kalles også for 'heis-algoritmen'[1] og er blant annet kjent som en viktig planleggingsalgoritme for magnetiske disker. Tanken bak å benytte denne metoden for serpentinbånd er at man skal kunne aksessere alle forespørslene i løpet av to 'scan' av tapen; ett forover-scan og ett bak-over-scan. Dette kan gjøres ved at man tilegner alle aksessene en fysisk posisjon på båndet basert på modellen beskrevet i "KAPITTEL 4: Søking på serpentinbånd". Deretter sorterer man først alle forespørslene på foroverspor i rekkefølge, og så sorteres forespørslene på bakoverspor tilsvarende. Da står man igjen med en liste der man fra BOT kan søke og lese i samme retning (forover) til man kommer til andre enden av båndet, så snur man og

1. Eng: *Elevator algorithm*

leser aksesser på bakoverspor i rekkefølge. SCAN-algoritmene tar ikke hensyn til ‘shoeshining’.

Ut fra betraktningene presentert i “KAPITTEL 4: Søking på serpentinbånd” kan det lages to litt forskjellige varianter av SCAN; en som benytter en generisk *N* for beregning av fysisk posisjon og en som benytter båndkarakteristikk-informasjon til å beregne fysisk posisjon. Disse kalles henholdsvis for *N-SCAN* og *K-SCAN*.

1. N-SCAN - En generisk algoritme

For **N-SCAN** kan fysisk posisjon for hver aksess beregnes med ligning 1 fra “KAPITTEL 4: Søking på serpentinbånd”. Det benyttes en fast *N*, som må baseres på et estimat for gjennomsnittlig antall 32kB blokker på et sporsett. Hver forespørsel må ‘tagges’ med beregnet fysisk posisjon og om den ligger på odde (forover) eller par (bakover) sporsett. Selve sorteringen av forover- og bakover-scannet gjennomføres ved å sortere de fysiske blokknummerne med *INSERTION SORT*.

Med en slik generisk *N* er det rimelig å anta at usikkerheten i beregningen av fysisk posisjon er så stor at for blokker som ligger fysisk nære hverandre vil det oppstå feil i rekkefølgen av sorteringen. Dette vil medføre at båndstasjonen må stoppe og snu for å lese feilsorterte blokker. Dette er tidkrevende og strider mot streaming-prinsippet.

2. K-SCAN - En algoritme basert på båndkarakteristikk

K-SCAN er en forbedring av N-SCAN. Den krever imidlertid at man karakteriserer båndet det skal søkes på. Dette kan for eksempel gjøres ved hjelp av en av karakteriseringsalgoritmene omtalt i “KAPITTEL 4: Søking

på serpentinbånd". Fordelen med dette er at man kan beregne de fysiske posisjonene mer nøyaktig enn med N-SCAN fordi man har et mer konkret bilde av blokkenes plassering på båndet. Dermed blir scan-rekkefølgen også mer presis og det antas at man kan unngå mange av snuoperasjonene som er forventet med N-SCAN. De fysiske posisjonene kan beregnes med ligning 1 dividert på *N* for gjeldende sporsett.

Sorteringen av forover- og bakover-scannet etter at forespørslene er tagget gjøres på samme måte som for N-SCAN.

3. Test-resultater

N-SCAN og K-SCAN ble implementert og kjørt mot båndstasjonen. Kildekoden til disse planleggerne er gjengitt i "VEDLEGG 3: Kildekode".

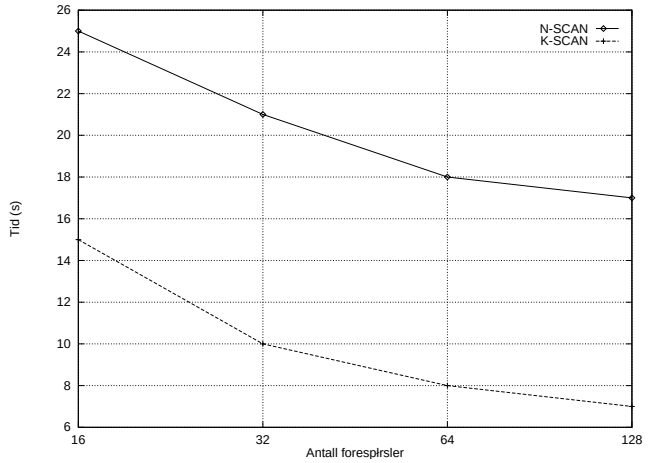
3.1 Innledende testing av N-SCAN og K-SCAN

N-SCAN ordner forespørslene ved å benytte en generisk *N = 5537*. Denne verdien ble valgt på grunnlag av studiene utført av Sandstå, Andersen, Midtstraum og Sætre [1997] i forbindelse med modellering av MLR1, beskrevet i "KAPITTEL 4: Søking på serpentinbånd". N-SCAN benytter *N* til å 'mappe' forespørslenes logiske adresser (blokknummer) til en *fysisk adresse* i intervallet [0..*N*]. Deretter blir disse *fysiske adressene* sortert med INSERTION SORT.

K-SCAN benytter karakteriseringsinformasjon for å sortere forespørslene. I dette tilfellet ble skrivetidene fra skriveloggen til båndet benyttet, det vil si at det ble kjørt en '*write-turn'* algoritme som beskrevet i "KAPITTEL 4: Søking på serpentinbånd" for å karakterisere båndet. K-SCAN tar dermed inn to tekstfiler; en med trekkelista og en med informasjon om båndets endepunkter. Ved å benytte informasjonen om disse endepunktene kan K-SCAN mappe forespørslenes logiske adresser til fysiske adresser på en mer

nøyaktig måte enn N-SCAN. Fortsatt ble INSERTION SORT benyttet til å sortere de *fysiske adressene*.

Som for testingen av SORT, ble det i første omgang trukket ut fem lister med tilfeldige forespørsler for henholdsvis 16, 32, 64 og 128 forespørsler. Dette gir selvfølgelig et dårlig grunnlag for presis vurdering av algoritmene, men figur 23 viser allerede klart at N-SCAN er langt dårligere enn K-SCAN. Ved å lytte på båndstasjonen mens den kjørte testene var det tydelig å høre at for N-SCAN ble det langt mer 'shoeshining' enn for K-SCAN.



FIGUR 23. Innledende tester viser at K-SCAN er langt bedre enn N-SCAN.

Det ble på dette tidspunktet bestemt å fortsette og kjøre flere tester med K-SCAN for å få et riktigere bilde av ytelsen. Samtidig ble det besluttet å ikke teste N-SCAN videre da denne algoritmen ikke var særlig interessant ettersom den var såpass mye dårligere enn K-SCAN.

3.2 Utvidet testing av K-SCAN

Det statistiske grunnlaget og resultatene av testingen med K-SCAN er gitt i tabell 13. Som for SORT ble det valgt å kjøre flere lister med få forespørsler

enn for lister med mange førespørsler. Grunnene til dette er de samme som beskrevet for SORT, men K-SCAN er mindre avhengig av trekningen[2].

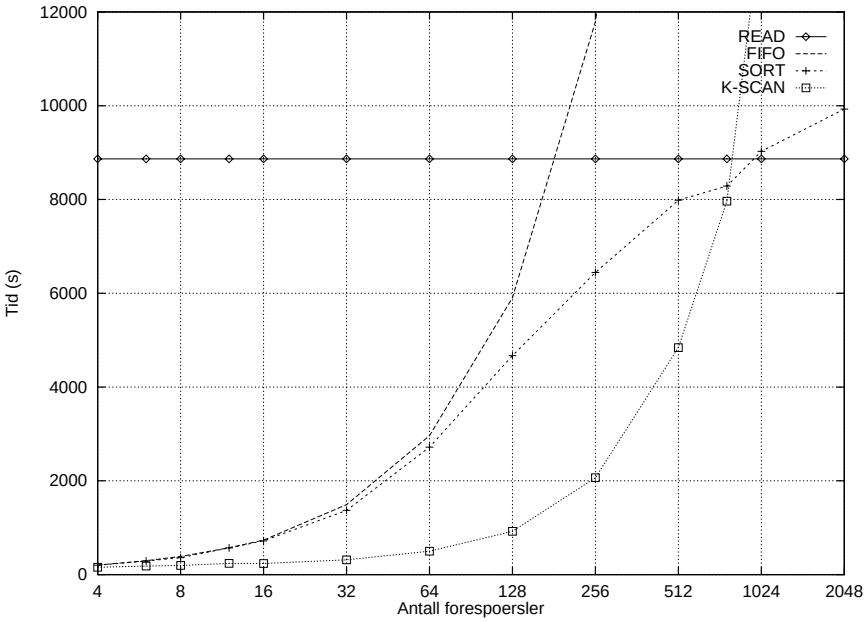
TABELL 13. Statistisk grunnlag og resultater for K-SCAN .

Antall forespørsler	Antall lister	Gjennomsnittlig totaltid (s)	Gjennomsnittlig aksesstid (s)
4	20	155	39
6	20	182	30
8	20	194	24
12	20	240	20
16	20	236	15
32	20	314	10
64	5	498	8
128	5	923	7
256	1	2065	8
512	1	4842	9
768	1	7964	10
1024	1	13896	14
2048	1	39447	25

Cpu-kostnaden for å planlegge 2048 forespørsler med K-SCAN slik den er implementert her er 3,4 sekunder[3].

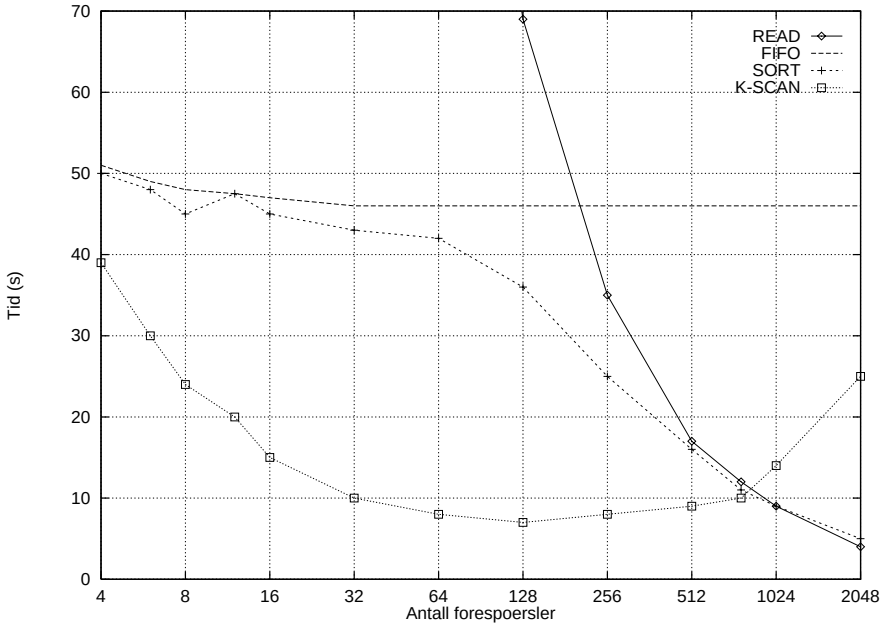
2. For 4 forespørsler med K-SCAN er forskjellen i totaltid 205%, mens den er bare 6% for 128 forespørsler.
3. Målt på samme maskin (*Raudeik*) som for SORT.

I figur 24 er kurven for K-SCAN lagt til plottet som ble vist i figur 21. Det går fram at K-SCAN er en klar forbedring fra de null-dimensjonale planleggingsmetodene fram til ca. 800 forespørsler.



FIGUR 24. Gjennomsnittlig totaltid som en funksjon av antall forespørsler for READ, FIFO, SORT og K-SCAN.

Figur 25 viser tydeligere fordelen med å bruke K-SCAN for lister med få forespørsler. Her er det mye å spare på gjennomsnittlig aksesstid.



FIGUR 25. Gjennomsnittlig aksesstid for hver forespørsel gitt som en funksjon av antall forespørsler for READ, FIFO, SORT og K-SCAN.

For mer enn 800 forespørsler begynner det å gå veldig dårlig for K-SCAN. Grunnen til dette er at forespørslene begynner å legge seg tett på forskjellige sporsett. Dette gjør at alt for mye tid går med til ‘shoeshining’ ved sporbytte[4].

4. Referer igjen til figur 16 i “KAPITTEL 4: Søking på serpentinbånd” for en forklaring og visualisering av dette fenomenet.

KAPITTEL 7

SLTF-algoritmen; To-dimensjonal planlegging

Hittil har det blitt presentert planleggings-strategier basert på såkalt null- og én-dimensjonal oppfatning av båndet. K-SCAN, som er et eksempel på sistnevnte, gir langt bedre resultater for planlegging av opptil 800 forespørsler enn de enklere metodene SORT og READ. I dette kapitlet tar man et skritt videre og utvikler en to-dimensjonal algoritme for planlegging av forespørsler. Denne algoritmen kalles SLTF og tar hensyn til tidskostnadene forbundet med forflytning både i lengde- og bredde-retningen. Sagt på en annen måte tar SLTF hensyn til at et sporskifte har en kostnad.

4. SLTF vs. K-SCAN

SLTF står for *Shortest Locate Time First*. Denne algoritmen har likheter med **SSTF** (*Shortest Seek Time First*) algoritmen [Bratbergsengen, 1997] for magnetiske disker. SSTF er en grådighetsalgoritme som starter ved *I* og

iterativt plukker ut det elementet i *R* som har minst kostnad, og som ikke har blitt aksessert ennå.

Den fundamentale forskjellen mellom SLTF og K-SCAN er at SLTF baserer planleggingen på et estimat av tidskostnader, som er en funksjon av både posisjonskostnad i lengderetningen og sporskiftekostnad i bredderetningen. Således er SLTF en to-dimensjonal algoritme til forskjell fra den én-dimensjonale K-SCAN. For SLTF må båndet karakteriseres som for K-SCAN, og det må regnes ut tidskostnader mellom de forskjellige aksessene. Man starter med å beregne kostnadene fra *I* (i vårt tilfelle BOT) til samtlige av aksessene i *R*. Aksessen med minst kostnad plukkes ut og legges i (den tomme) sorteringslista, *S*. Deretter beregner man tidskostnadene fra den nylig uttrukne blokka til alle resterende forespørsler i *R*, plukker ut det elementet som nå er minst, og legger det sist i *S*. Slik fortsetter det til alle forespørslene i *R* er plukket ut. Kompleksiteten til SLTF er dermed $O(n^2)$. For virkelig store lister vil dette kreve ganske mye cpu-tid, men for forespørselslistene benyttet i denne hovedoppgaven går dette greit. 2048 forespørsler planlegges på 7,8 sekunder[1]. Dette betyr at første aksess blir forsinket, men for totaltiden er det neglisjerbart.

SLTF skiller seg også fra K-SCAN når det gjelder hodets plassering etter siste aksess er lest. K-SCAN vil nærme seg starten av båndet jo flere aksesser som skal planlegges. Dette er en konsekvens av at det leses forespørsler på foroverspor først, og deretter leses det tilbake mot BOT. For SLTF vil hodet tendere til å bli plassert i motsatt ende så lenge man opererer med et moderat (< 32) antall forespørsler. Dette kommer av algoritmens natur, som er å hente aksessen med korteste tidskostnad først. For mange forespørsler viser det seg at SLTF også nærmer seg BOT, hvilket blir diskutert litt senere.

I lys av disse forskjellene kommer man tilbake til diskusjonen om når man skal anse gjennomføringen av en planlegging og aksessering av båndet for fullført. Dersom man ville at alle testgjennomføringer skulle ende ved BOT ville det for SLTF medført en tilbakespoling etter siste aksess på opptil båndets lengde (126 sekunder) så lenge man ikke har mange forespørsler.

1. Målt på samme maskin (*Raudeik*) som for SORT og K-SCAN.

For K-SCAN ville denne tilbakespolingen som regel være mindre. For de tilfeller der man opererer med mange aksesser vil den være minimal. Da det er blitt bestemt at tidsmålingen skal slutte etter siste aksess, diskuteres ikke dette videre her, men det er viktig å være klar over dette.

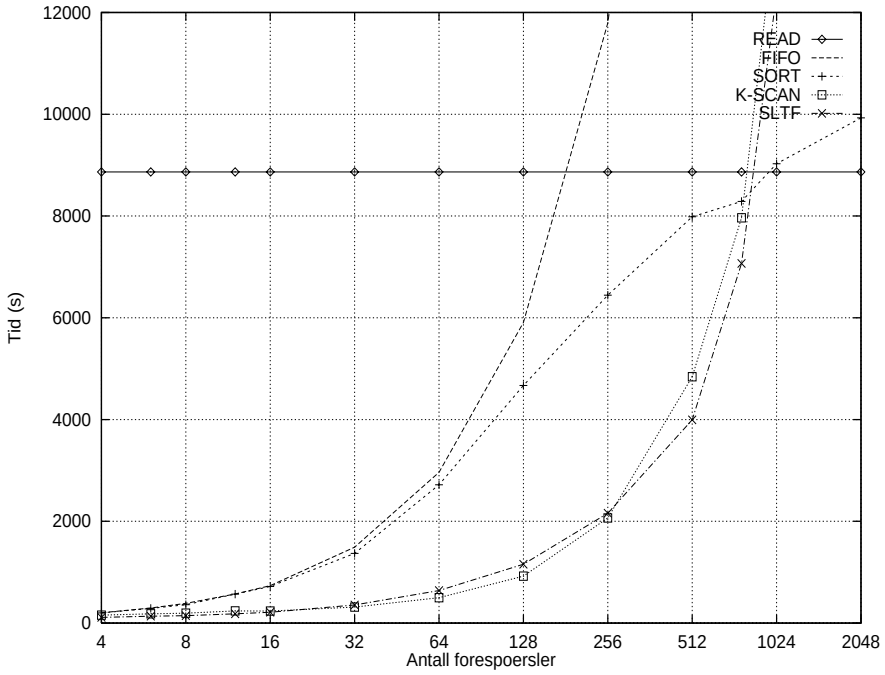
5. Test-resultater

SLTF ble implementert og kjørt mot båndstasjonen, og kildekoden til SLTF-algoritmen er som vanlig å finne i “VEDLEGG 3: Kildekode”. Implementeringen baserer seg på tidsberegninger i henhold til den avanserte modellen som er beskrevet på slutten av “KAPITTEL 4: Søking på serpentinbånd”. For karakterisering av båndet ble ‘*write-turn*’ algoritmen brukt på samme måte som for K-SCAN.

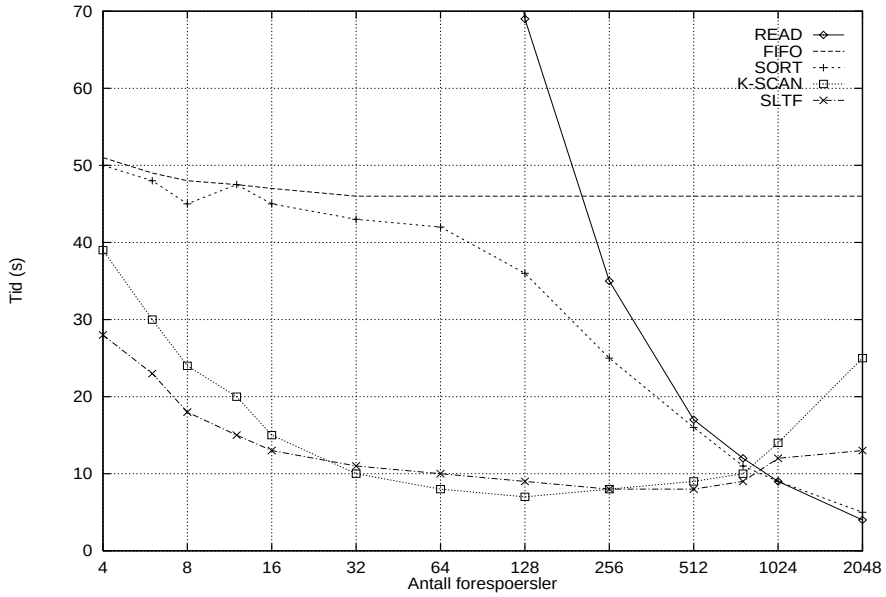
TABELL 14. Statistisk grunnlag og resultater for SLTF .

Antall forespørsler	Antall lister	Gjennomsnittlig totaltid (s)	Gjennomsnittlig aksesstid (s)
4	20	111	28
6	20	136	23
8	20	146	18
12	20	182	15
16	20	213	13
32	20	355	11
64	5	638	10
128	5	1157	9
256	1	2159	8
512	1	3993	8
768	1	7068	9
1024	1	12383	12
2048	1	27359	13

Tabell 14 oppsummerer resultatene for SLTF-testingen. Disse resultatene er videre presentert grafisk i figur 26 og figur 27.

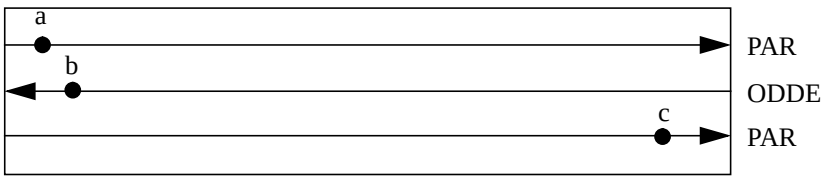


FIGUR 26. Gjennomsnittlig totaltid som en funksjon av antall forespørsler for READ, FIFO, SORT, K-SCAN og SLTF.



FIGUR 27. Gjennomsnittlig aksesstid for hver forespørsel gitt som en funksjon av antall forespørsler for READ, FIFO, SORT, K-SCAN og SLTF.

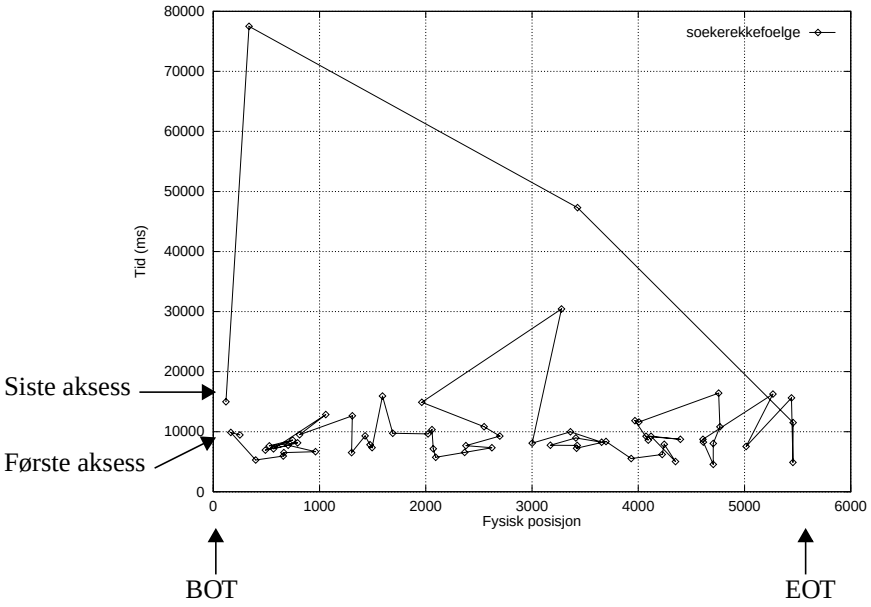
Det går fram at SLTF og K-SCAN oppfører seg ganske likt. SLTF viser noe bedre ytelse for få forespørsler (< 32). Dette virker også rimelig ettersom SLTF ikke diskriminerer mellom par og odde sporsett på samme måte som K-SCAN. I



FIGUR 28. Konstruert situasjon der K-SCAN taper mye i forhold til SLTF.

llustrasjonen på figur 28 viser en konstruert situasjon der blokkene *a*, *b* og *c* skal aksesseres. For K-SCAN resulterer dette i tilnærmet verste tilfelle ettersom det nesten er nødvendig med et fullt scan i hver retning. K-SCAN vil sortere disse aksessene i rekkefølgen *a-c-b.* SLTF er langt mer optimal for dette tilfellet fordi den vil sortere aksessene i rekkefølgen *a-b-c*. Dette resulterer i kun én gjennomløpning av båndet.

For mellom 32 og 256 forespørsler ligger kurven for K-SCAN noe under SLTF. Forskjellen er ganske liten, men den kommer tydelig fram på figur 27. I dette intervallet fremviser K-SCAN den beste ytelsen. Det ble antatt at grunnen til at SLTF er litt dårligere enn K-SCAN i dette intervallet er at SLTF, som er en grådighets-algoritme, rett og slett er for 'grådig'. SLTF tar ikke hensyn til det faktum at å velge den minst kostbare aksessen for øyeklikket, kan resultere i at man må gå en uforholdsmessig lang vei senere. For å sjekke dette ble søkerekkefølgen til SLTF plottet for forskjellige trekkelister.



FIGUR 29. Søkerekkefølgen til SLTF for en av testene med 64 forespørsler.

Figur 29 viser et av disse plottene. Den horisontale aksen viser fysiske posisjoner mens den vertikale aksen viser målt søketid mellom aksessene. Her går det klart fram at SLTF er alt for grådig. Resultatet er at det blir nødvendig med et helt scan tilbake til BOT for å få hentet blokkene som ble "vraket" i starten.

Mellom 256 og 800 forespørsler er det igjen SLTF som kommer best ut. Dette er i første omgang et resultat av at ytelsen til K-SCAN degraderes mer enn for SLTF. For begge disse algoritmene gjelder det at det blir for mye 'shoeshining' når det blir for mange forespørsler som ligger for tett.

6. Vurdering av tidsestimatene for SLTF; validering av modellen

Testingen av SLTF har en svakhet i forhold til testingen av K-SCAN og SORT; SLTF baserer seg på en modell som ikke har vært utprøvet før. Det ideelle hadde vært om man kunne validert modellen for seg først, og deretter benyttet modellen til implementering av SLTF. Av tidsmessige hensyn var ikke dette aktuelt for denne hovedoppgaven, og det ble derfor besluttet å implementere SLTF med modellen slik den var. Det er likevel mulig å si noe om modellen. Under kjøringen av listene planlagt med SLTF ble det nemlig i tillegg til den vanlige posisjonerings- og tidsmålings-informasjonen også logget beregnet tidskostnad mellom aksessene. Forskjellen mellom estimert og målt tidskostnad kan fortelle oss noe om hvor god eller dårlig modellen SLTF baserer seg på er.

Husk at modellen ble presentert på slutten av "KAPITTEL 4: Søking på serpentinbånd". Her ble det også vist at modellen burde ha en teoretisk feilmargin på maksimalt $\pm$ 4 sekunder. I tabell 15 er avvik for de forskjellige listene gjengitt.

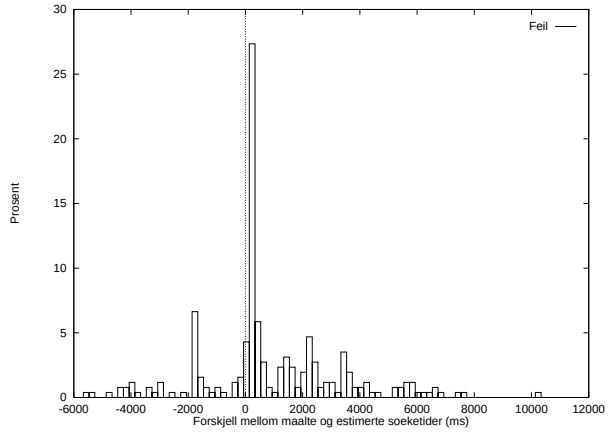
TABELL 15. Tallgrunnlag for validering av modellen som SLTF benytter.

Antall forespørsler	Gj.snittlig aksesstid (ms)	Max. avvik (ms)	Gj.snittlig avvik	
			(ms)	%
4	27750	3906	2638	9,5
6	22667	3216	1933	8,5
8	18250	5406	2359	12,9
12	15167	4674	2128	14,0
16	13313	5261	2042	15,3
32	11094	5029	1667	15,0
64	9969	6431	1535	15,4
128	9039	5961	1585	17,5
256	8434	10252	1823	21,6
512	7799	15473	1948	25,0
768	9203	21443	3258	35,4
1024	12093	37343	6404	53,0

Vi ser at for opp til og med 768 forespørsler ligger absoluttverdien av gjennomsnittlig avvik innenfor disse 4 sekundene. For lister med 1024 eller flere forespørsler har vi sett at SLTF ikke fungerer særlig bra, og at READ er et bedre alternativ. Vi ser også at gjennomsnittlig avvik for 1024 forespørsler er 6404 sekunder, eller 53,0%. Det er tydelig at her fungerer ikke tidsestimeringen, og dermed feiler denne planleggings-strategien.

Som et eksempel på hvordan forskjellen mellom målte og estimerte verdier fordeles, viser figur 30 feildistribusjonen for en test av SLTF med 128 forespørsler. “VEDLEGG 1: Feilfordelinger for modellen benyttet av SLTF” viser feildistribusjoner for forespørselsmengder fra og med 64 til og med 1028.

Basert på disse observasjonene kan man trekke slutningen at modellen oppfører seg som forventet opp til ca. 800 forespørsler. For planlegging av flere forespørsler viser det seg at modellen er for ideell da det virker som om båndstasjonen “blir forvirret” og begynner med alt for mye ‘shoeshining’ når aksessene kommer for tett. For dette tilfellet må man enten forbedre modellen radikalt, eller kanskje det vil vise seg at det bør utvikles en ny og annerledes modell som tar enda flere hensyn til båndstasjonens egenartede oppførsel.



FIGUR 30. Feildistribusjon for forskjellen mellom målte og estimerte aksesstider for 256 forespørsler.

KAPITTEL 8

Oppsummering

Dette kapittelet oppsummerer hva som har blitt gjort i løpet av hovedoppgaven, hvilke konklusjoner man kan trekke ut av resultatene, samt noen forslag til videre arbeid relatert til modellering av båndstasjonen og planlegging av forespørlser.

1. Utført arbeid

Det har blitt utført et studium av masselagersystemer generelt, og båndstasjonsteknologier spesielt. Videre har forfatteren satt seg inn i Tandbergs MLR1 båndstasjon og beskrevet dens viktigste egenskaper og oppførsel. Det har deretter blitt gitt en oversikt over tidligere arbeider gjennomført av forfatteren og andre ved instituttet i tilknytning til modellering av båndstasjonen. Basert på dette arbeidet har det så blitt gjort en videreutvikling av den beste modellen. Tilsammen danner dette grunnlaget for utviklingen av planleggingsstrategiene og forsøkene som ble gjennomført.

Forfatteren har presentert noen forskjellige strategier for planlegging av forespørsler mot båndstasjonen. Først ble de enkleste strategiene implementert, og det ble kjørt en rekke tester for å gi et bilde av ytelsen man oppnådde ved disse planleggerne. Etter dette ble det gjennomført tilsvarende tester for stadig mer komplekse planleggings-strategier. I alt har de tre planleggingsalgoritmene SORT, K-SCAN og SLTF blitt testet for et variert utvalg av forespørselsmengder. Algoritmene har blitt sammenlignet og diskutert underveis.

2. Konklusjon

Siden systemer som video- eller bilde-arkiv vil ha et enormt behov for lagringskapasitet, vil det av kostnads- og tilgjengelighets-messige hensyn være ønskelig å bruke digitale bånd for lagring. Ulempene forbundet med lange aksesstider må dermed minimaliseres. Dette vil blant annet kreve at man på et lavt nivå setter seg inn i den akuelle lagringsteknologien og klarer å utvikle en modell for båndstasjonen. Denne modellen må så benyttes som grunnlag for en planleggingsstrategi som ordner forespørsler på en slik måte at samlet aksesstid blir minimalisert.

Et ledd i utviklingen av en slik modell er studien av hvordan båndstasjonen oppfører seg ved søking fra begynnelsen av båndet (BOT) til tilfeldige posisjoner. For MLR1 resulterer dette i pyramide-plottene som er karakteristisk for serpentinbånd. For søking på partalls sporsett faller søketidene og tilbakespolingstidene godt sammen ettersom båndstasjonen leser/søker like fort som den spoler. For oddetalls sporsett får man overlagret tilbakespolingstidene med det karakteristiske sagtann-mønsteret for søketidene. Dette kommer av at båndstasjonen må spole forbi måladressen for så å snu og lese tilbake.

Videre viser det seg at det er nødvendig å karakterisere hvert bånd slik at man har en oversikt over hvor mye data som ligger på hvert sporsett. Forskjellige metoder kan benyttes til dette formålet. Om man ikke har tilgang til skriveloggen anbefales ‘Read-turn’ metoden. Denne har en gjennom-

snittlig tidskostnad på 13 minutter. Med tilgang på skriveloggen er 'Write-turn' metoden å foretrekke da den er tilnærmet gratis med tanke på tidskostnaden. Begge disse metodene benytter seg av at lese/skrive-tidene gjør store hopp ved sporskifte på grunn av at stasjonen må utføre en del mekaniske operasjoner i forbindelse med reposisjonering. Det har vist seg at skrivetidene ligger i all vesentlig grad under 40 ms for skriving av en 32 kB blokk når man ser bort fra sporskiftene.

Når båndet er karakterisert er det nødvendig å videreutvikle modellen ved å dele opp problemene forbundet med tilfeldige søk. Disse problemene oppstår på grunn av 'shoeshining'-effekten, som er den uønskede situasjonen der båndstasjonen er nødt til å spole fram og tilbake for å få reposisjonert skrive/lese-hodet. Ved å ta hensyn til 'shoeshining' kan det lages et matematisk uttrykk for en avansert modell av båndstasjonen.

Med en modell av båndstasjonen kan man benytte forskjellige strategier for å sortere grupper av forespørsler. Det viser seg at for mer enn 800 forespørsler er den mest hensiktsmessige metoden å lese hele båndet fra BOT til siste aksess. Lesingen av et fullt bånd tar i gjennomsnitt 8865 sekunder, (2 timer, 27 minutter og 45 sekunder) og ingen samling av forespørsler bør ta lenger tid å lese enn dette. For færre forespørsler viser algoritmene K-SCAN og SLTF beste resultat. K-SCAN er enklest av disse to og tar kun hensyn til blokkenes fysiske posisjon i lengderetningen ved sortering. I intervallet fra 32 til 256 forespørsler er K-SCAN beste algoritme, med SLTF hakk i hæl. SLTF baserer sorteringen på estimerte tidskostnader og tar hensyn til kostnader forbundet med sporskifte såvel som kostnader forbundet med forflytning i lengderetningen. I intervallet fra 4 til 32 forespørsler er SLTF klart best, og den gir også litt bedre resultater enn K-SCAN i intervallet 256 til 800 forespørsler. Tabell 16 oppsummerer mest hensiktsmessig valg av planleggings-strategi.

TABELL 16. Valg av planleggings-strategi for forskjellig antall forespørsler.

Antall forespørsler	Strategi
4-32	SLTF
32-256	K-SCAN
256-800	SLTF
800+	READ

3. Videre arbeid

En direkte videreføring av denne hovedoppgaven kan deles inn i fire deler: Ytterligere uttesting og statistisk innsamling av data, forbedring av modellen, forbedring av planleggings-strategiene, eventuell utvikling og testing av nye strategier, og utvikling av en “dynamisk” planlegger.

3.1 Ytterligere uttesting

Testing og bruk av båndstasjonen krever mye tid, og det er begrenset hvor mye man får gjort i løpet av en diplom-periode. Det bør gjennomføres flere tester av de mest aktuelle algoritmene K-SCAN og SLTF for å få et bedre statistisk grunnlag slik at man får en jevnere kurve med finere oppløsning enn de tendens-kurvene som er presentert her. I tillegg til å kjøre flere lister med det antallet forespørsler som er benyttet i denne hovedoppgaven, vil det være en fordel å trekke lister med andre mengder av forespørsler. Særlig rundt skjæringspunktene mellom de forskjellige strategiene vil dette kunne medføre at man mer nøyaktig kan sette en grense for når man skal skifte fra en algoritme til en annen.

3.2 Forbedring av modellen

Den avanserte modellen som er presentert i denne hovedoppgaven anses for å være rimelig god, og bra nok til å benyttes i et reelt system. Det hadde likvel vært en klar fordel å fått testet ut modellen skikkelig før man benyttet den i en planleggings-strategi. Uttesting, og eventuell forbedring, av modellen er ingen uoverkommelig oppgave med utgangspunkt i det arbeidet som er lagt ned i denne hovedoppgaven. Planlegging, implementering og gjennomføring (mot båndstasjonen) av en testrekke for dette formålet bør ikke ta mer enn en uke.

Selv om testing av modellen vil avsløre enkelte svakheter som kan forbedres, vil usikkerheten forbundet med plasseringen av nøkkelpunktene på hvert sporsett være en stor begrensning for denne modellen. Hadde det vært mulig å bestemme nøkkelpunktene på hvert sporsett, ville dette blant annet gi SLTF muligheten til å estimere tidskostnadene langt mer nøyaktig. Slik det ligger an i dag vil en karakterisering av alle nøkkelpunktene være alt for tidkrevende. På grunn av usikkerheten forbundet med feil/ødelagte blokker synes det heller ikke som om det er noen mulighet til å beregne disse nøkkelpunktene med tilstrekkelig nøyaktighet.

Men det er likevel håp: Det finnes nemlig en oversikt over samtlige nøkkelpunkter på starten av hvert bånd. Problemet er bare at det ikke er mulig å få lest ut disse fra båndstasjonen uten spesielle applikasjoner og en seriellkabel per idag. Dersom produsentene av båndstasjoner, for MLR1 sin del vil dette si Tandberg Data ASA, bestemmer seg for at det kunne være lurt å la brukeren få tilgang til denne informasjonen, ville det sannsynligvis vært mulig å lage mye bedre modeller av båndstasjonene. I følge Tandberg Data ASA er det ikke store tekniske problemer forbundet med en slik utvidet funksjonalitet, problemet ligger heller i at det reduserer produsentens frihet hva utvikling av nye formater for bånd angår. Inntil videre er man derfor nødt til å finne seg i at digitale bånd er en temmelig “lukket” teknologi, og basere seg på egenutviklede modeller.

3.3 Forbedring av planleggings-strategiene

Implementeringen av planleggings-strategiene er gjort på enkleste måte. Her er det derfor blant annet rom for å skifte ut INSERTION SORT med en raskere sorteringsalgoritme. For større antall av forespørsler vil dette medføre lavere cpu-kostnad, og enda viktigere vil det medføre at tiden til første aksess kan reduseres. For total- og gjennomsnitts-tiden vil dette likevel spille liten rolle.

En viktigere og mer interessant utvidelse vil være å teste de forskjellige algoritmene når hver forespørsel har en tilfeldig lengde i stedet for bare en blokk. Forfatteren antar at dette ikke vil medføre særlig forandringer (bortsett fra lengre lesetid) for SLTF da denne algoritmen baserer seg på utregning av tidskostnader. For K-SCAN vil det antakeligvis ødelegge betraktelig for tilfeller med mange forespørsler slik denne algoritmen er implementert i dag. Dette begrunnes med at lange lesestrekninger på båndet vil ødelegge scannet fordi man må spole tilbake dersom aksessene ligger tett.

3.4 Utvikling og testing av nye planleggings-strategier

Det finnes flere andre planleggingsalgoritmer det hadde vært interessant å teste ut. Hillyer og Silberschatz [1996b] har for eksempel simulert en variant av SLTF som de kaller LOSS. Denne algoritmen gir litt bedre resultater enn SLTF for båndstasjonen de har simulert. LOSS er mindre “grådig” enn SLTF fordi den tar hensyn til at det ikke nødvendigvis er best å velge korteste aksess dersom det medfører at senere aksesser blir veldig lange. Det kunne også være interessant å implementere en optimal algoritme basert på ‘travelling salesman’-teorien. Problemet med ‘travelling salesman’ er at den er NP-komplett og praktisk uløselig for selv relativt liten mengde forespørsler. En tilnærming til en optimal algoritme kan baseres på spenntrærteori som garanterer at verste tilfelle kun er dobbelt så stort som en optimal algoritme.

Hvor mye det er å vinne på å implementere disse strategiene er det vanskelig å si noe om på forhånd. Det er imidlertid forfatterens oppfatning at disse

algoritmene ikke vil gi nevneverdig bedre ytelse før man får modellert båndstasjonen bedre.

3.5 Utvikling av en “dynamisk” planlegger

Planleggings-strategiene som er beskrevet i denne hovedoppgaven kan karakteriseres som statiske. Mengden av inndata for en liste er konstant og tilgjengelig ved starten av planleggingen. En interessant utvidelse ville vært å tillate nye ankomster av forespørsler å bli planlagt fortløpende, altså mens båndstasjonen er opptatt med å hente allerede planlagte forespørsler. I så tilfelle kunne man betegnet dette som en dynamisk planleggings-strategi. Dette ville innebære en mer komplisert algoritme for planleggingen, blant annet måtte man da tatt hensyn til at enkelte forespørsler kunne bli utsultet.

Referanser

Andersen, T. M., *Modellering av MLR1 båndstasjon*. (Prosjektoppgave, 1997, Institutt for datateknikk og informasjonsvitenskap, NTNU).

ATL Products Inc., 1997. *ATL Products, Inc.*
Tilgjengelig fra: http://www.atlp.com/7100.html [Aksessert 4 jan 1998].

Bratbergsengen, K., *Filsystemer. Lagring og behandling av store datamengder.* (Lærebok i faget *Filsystemer*), Trondheim: Institutt for datateknikk og informasjonsvitenskap, NTNU, Høsten 1997.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., *Introduction to Algorithms*. New York: McGraw-Hill Book Company, 1992.

Dynamic Solutions International Corp., 1996. *Alpine DLT Libraries.*
Tilgjengelig fra: http://www.dynamicsolutions.com/pages/products/dlt-libs.htm [Aksessert 4 jan 1998].

Fluckinger, F., *Understanding networked multimedia - applications and technology.* Hertfordshire: Prentice Hall International (UK) Limited, 1995

Hillyer, B. K., Silberschatz, A., On the Modeling and Performance Characteristics of a Serpentine Tape Drive. *Proceeding fra ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Philadelphia, Pensylvania, USA, 23-26 mai 1996a, ss 170-179.*

Hillyer, B. K., Silberschatz, A., Random I/O Scheduling in Online Tertiary Storage Systems. *Proceeding fra ACM Sigmod Int. Conference on Management on Data, Montreal, Canada, 3-6 juni 1996b, ss 195-204.*

Intelligent Solutions Inc., 1997. *DLT FAQ*. Tilgjengelig fra: http://www.intel-sol.com/solutions/dlt/wp_faq.html [Aksessert 4 jan 1998].

Merok, P., *Datamodeller for digitale film- og video-arkiv.* Diplomoppgave, Institutt for datateknikk og telematikk, NTH, desember 1993.

Tandberg Data ASA, *Tandberg MLR1 Series. Streaming Tape Cartridge Drives. Reference Manual*. Oslo: Tandberg Data ASA, 1996a.

Tandberg Data ASA, *Tandberg MLR1 Series. Streaming Tape Cartridge Drives. SCSI Interface Functional Spesifications*. Oslo: Tandberg Data ASA, 1996b.

Tandberg Data ASA, 1997. *Tandberg Data - Tape Storage Solutions*. Tilgjengelig fra http://www.tdata.no/products/mlr_library.html [Aksessert 4 jan 1998].

Quantum Corp., 1997. *Quantum DLT(tm) 7000 Online Spesifications*. Tilgjengelig fra: http://www.quantum.com/products/dlt/dlt7000/q_d70_sp.html [Aksessert 4 jan 1998].

Sandstå, O., Midtstraum, R., *Low-Cost Access Time Models for a Serpentine Tape Drive*. (Teknisk rapport, 1997, Institutt for datateknikk og informasjonsvitenskap, NTNU).

Sandstå, O., Andersen, T. M., Midtstraum, R., Sætre, R., Access Time Modeling of a MLR1 Tape Drive. *Proceeding fra Norsk Informatikkonferanse, Voss, 24-26 november 1997*, ss 267-278.

Stallings, W. *Data and Computer Communications*, 4. utgave. Macmillan Publishing Company, 1994.

Sætre, R., *Video på digitale band - Bibliotek for aksessering av video på band*. (Teknisk rapport, 1996, Institutt for datateknikk, NTNU).

Sætre, R., Sandstå, O., *Video på digitale band*. (Teknisk rapport, 1996, Institutt for datateknikk, NTNU).

Tactical Marketing Group, 1996. *Digital Video Disc Background*. Tilgjengelig fra: http://www.tacmar.com/dvd_background.htm [Aksessert 5 jan 1998].

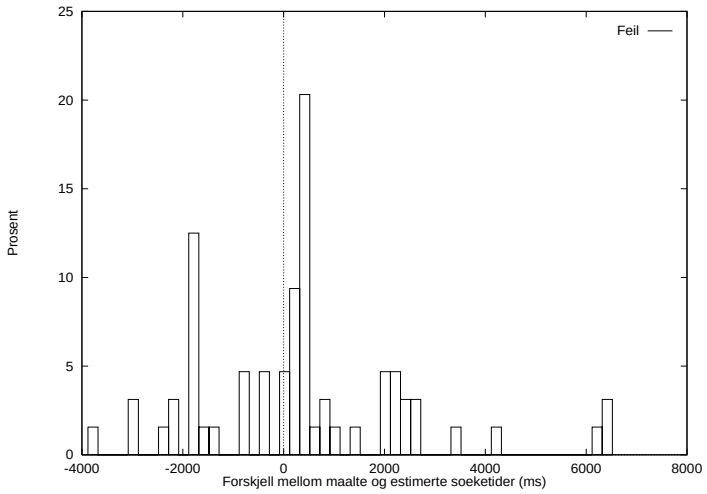
VEDLEGG 1

Feilfordelinger for modellen benyttet av SLTF

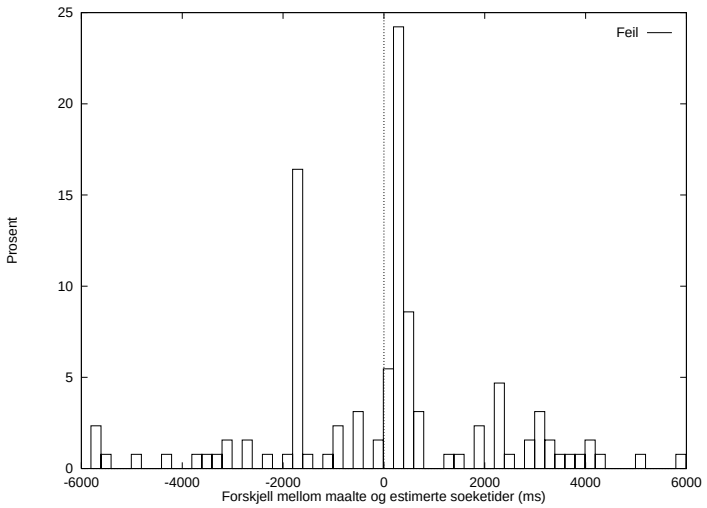
I dette vedlegget finnes feilfordelingene til et utvalg av listene som ble benyttet i forbindelse med testingen av SLTF-algoritmen. Feilfordelingene til listene med færre en 64 forespørsler er ikke tatt med, da disse blir ganske grove og lite interessante. Tallgrunnlaget er for enkelhets skyld gjentatt i tabell 17.

TABELL 17. Tallgrunnlag for validering av modellen som SLTF benytter.

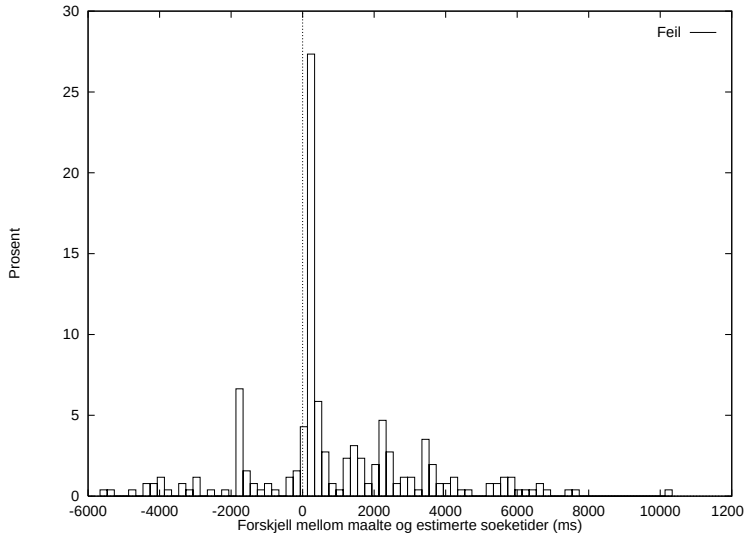
Antall forespørsler	Gj.snittlig aksesstid (ms)	Max. avvik (ms)	Gj.snittlig avvik	
			(ms)	%
64	9969	6431	1535	15,4
128	9039	5961	1585	17,5
256	8434	10252	1823	21,6
512	7799	15473	1948	25,0
768	9203	21443	3258	35,4
1024	12093	37343	6404	53,0



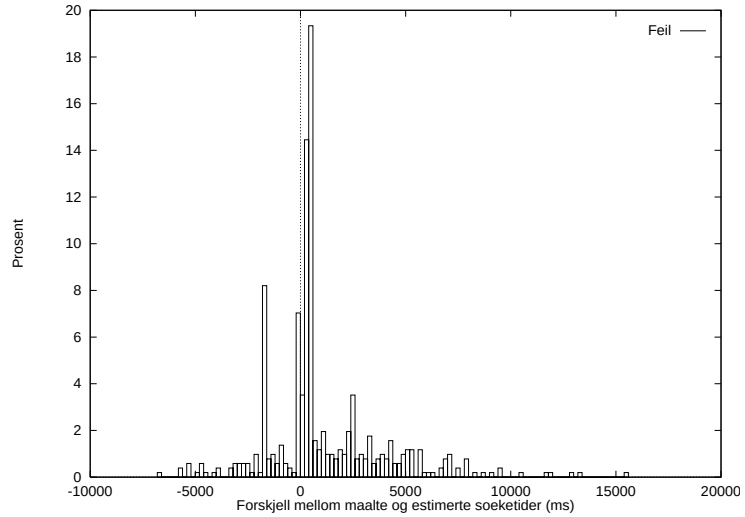
FIGUR 31. Feildistribusjon for 64 forespørsler.



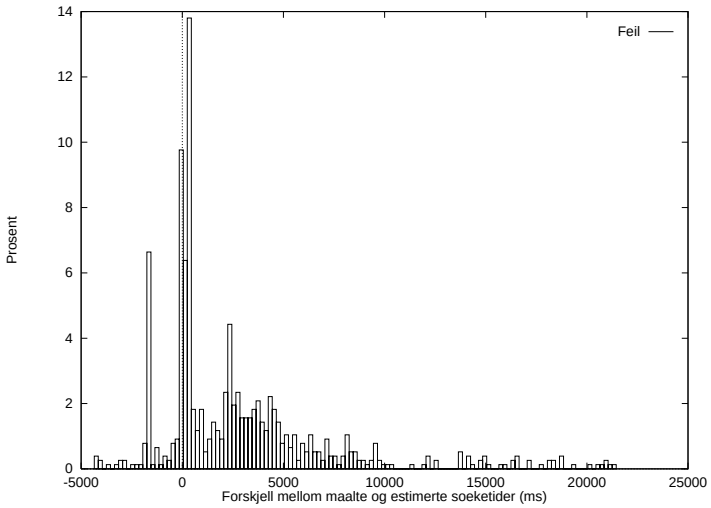
FIGUR 32. Feildistribusjon for 128 forespørsler.



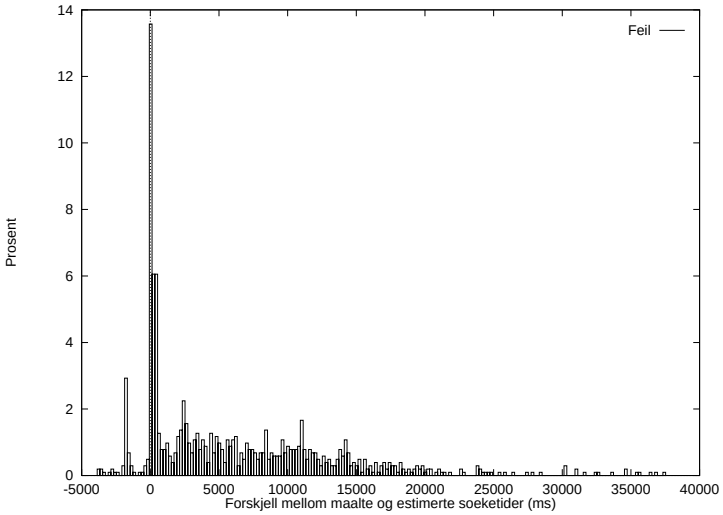
FIGUR 33. Feildistribusjon for 256 forespørsler.



FIGUR 34. Feidistribusjon for 512 forespørsler.



FIGUR 35. Feildistribusjon for 768 forespørsler.



FIGUR 36. Feildistribusjon for 1024 forespørsler.

VEDLEGG 2

Proceedings fra Norsk Informatikkonferanse, Voss 1997

I dette vedlegget finnes en utskrift av artikkelen som ble presentert av forfatteren på Norsk Informatikkonferanse på Voss, 1997. Artikkelen oppsummerer mye av det arbeidet Olav Sandstå, Roger Midtstraum, Rune Sætre og forfatteren har arbeidet med før (og litt i parallell med) denne hovedoppgaven. Dette arbeidet danner grunnlaget for modellene som ligger i bunn for undersøkelsene som er utført, og "KAPITTEL 4: Søking på serpentinbånd" baserer seg mye på innholdet i denne artikkelen. Referansen til artikkelen er [Sandstå, Andersen, Midtstraum og Sætre, 1997].

Artikkelen er gjengitt på originalt format (L^AT_EX), bortsett fra at sidetallene er endret til å passe inn i denne rapporten.

VEDLEGG 3

Kildekode

Alle programmene er implementert i *c++* og kompilert med *g++*. Av praktiske hensyn er kildekoden formatert med programmet *c++2latex* og deretter skrevet ut med tekstbehandleren $\LaTeX$. Tabell 18 gir en kort oversikt over filene som er vedlagt.

TABELL 18. Oversikt over kildekode-vedlegget .

Filnavn	**Type**	**Kort Beskrivelse**
tapestreamer.h	Bibliotek	Bibliotek som inneholder definisjoner for operasjoner mot båndstasjonen.
write_log.cc	Skriveprogram	Skriver kontinuerlig blokker på 32kB til båndstasjonen.
trekk.cc	Trekkeprogram	Trekker tilfeldige forespørsler innenfor et gitt intervall.

TABELL 18. **Oversikt over kildekode-vedlegget (Fortsatt).**

Filnavn	Type	Kort Beskrivelse
SORT.cc	Planleggings-algoritme	Planlegger forespørslene trukket med *trekk.cc* i henhold til INSERTION SORT algoritmen.
NSCAN.cc	Planleggings-algoritme	Planlegger forespørslene trukket med *trekk.cc* i henhold til fysiske posisjoner og en generisk *N*.
KSCAN.cc	Planleggings-algoritme	Planlegger forespørslene trukket med *trekk.cc* i henhold til fysiske posisjoner og karakteriserings-informasjon.
SLTF.cc	Planleggings-algoritme	Planlegger forespørslene trukket med *trekk.cc* i henhold til fysiske posisjoner og tidsestimering for forflytning mellom disse.
MLR1driver.cc	Båndstasjons-driver	Program som kjører en liste med forespørsler mot båndstasjonen i rekkefølge.

Merk:
Biblioteket *tapestreamer.h* er utviklet av Sætre [1996], og den originale koden med kommentarer er gjengitt, bortsett fra filbeskrivelsen på starten av fila. Denne beskrivelsen er lagt til i etterhånd.