FFIE/671/161.2

Godkjent
Kjeller 6 august 1997

Reidar Skaug
Forskningssjef

DATADISTRIBUSJON I DIVISJONENS K2IS

FARSUND Bodil Hvesser, JENSVOLL Audun,
SANDER Jostein

FFI/RAPPORT-97/03389

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2007 Kjeller, Norge

NORWEGIAN DEFENCE RESEARCH ESTABLISHMENT (NDRE)
FORSVARETS FORSKNINGSINSTITUTT (FFI)

POST OFFICE BOX 25
N-2007 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-97/03389	UNCLASSIFIED	49
1a) JOB REFERENCE	2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	
FFIE/671/161.2		

4) TITLE

DATADISTRIBUSJON I DIVISJONENS K2IS

(DATA DISTRIBUTION IN THE C2IS OF THE DIVISION)

5) NAMES OF AUTHOR(S) IN FULL (surname first)

FARSUND Bodil Hvesser, JENSVOLL Audun, SANDER Jostein

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited.
(Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:		IN NORWEGIAN:	
a)	X.400	a)	X.400
b)	Distributed databases	b)	Distribuerte databaser
c)	Battle view	c)	Situasjonsbilde
d)	Traffic load	d)	Trafikkbelastning
e)		e)	

THESAURUS REFERENCE:

8) ABSTRACT (continue on reverse side if necessary)

A survey on the message handling protocol X.400 and distributed databases is presented. The traffic load is measured when using both the X.400 protocol and the automatic distributed database update SQL*Net. The different sizes of the information packages is also looked into. Finally the feasibility of keeping an updated view on both own and enemy situation is discussed.

9) DATE	AUTHORIZED BY This page only	POSITION
6 August 1997	Reidar Skaug	Director of Research

ISBN 82-464-0182-3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD Side

DATADISTRIBUSJON I DIVISJONENS K2IS

1 INNLEDNING

Dette notatet beskriver resultater kommet frem under arbeidet med prosjekt 671-KKI-HÆR. Dette prosjektet skal utarbeide et nytt og moderne konsept for divisjonens ledelsessystem som skal understøtte det nye manøverkrigføringskonseptet. Denne rapporten beskriver informasjonssystemet, og de viktigste elementene dette består av. Informasjonssystemet omfatter i denne sammenheng ikke sambandssystemet. Vi har imidlertid sett på informasjonssytemet i lys av sambandsbelastningen, da mye tyder på at sambandet vil utgjøre en begrensende faktor i et fremtidig Kommando Kontroll og Informasjonssystem (K2IS).

I denne rapporten har vi først og fremst tatt for oss den delen av informasjonssystemet som omhandler formidling og lagring av informasjon. Når det gjelder formidling av informasjon så har vi sett spesielt på ISO-protokollen X.400, da dette er den protokollen som meldingssystemet i divisjonen sannsynligvis vil benytte. Vi har sett nærmere på denne protokollen i kapittel 2. Til å lagre informasjon vil databaser bli benyttet. I divisjonen er man avhengig av å ha distribuerte databaser for å oppnå tilstrekkelig redundans, og for å begrense avhengigheten av at sambandet fungerer. De ulike egenskapene til en distribuert database er belyst i kapittel 3.

I kapittel 4 har vi sett nærmere på andre mye brukte protokoller for lagring og formidling av informasjon, nemlig Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP) og Telecommunications Network (TELNET). I kapittel 5 er det gjengitt resultater etter måling av nettverkstrafikken ved bruk av de ulike protokollene. I kapittel 6 har vi sett på hvordan informasjonen kan representeres og funnet generelle uttrykk for total trafikk på typiske informasjonspakker. Videre har vi i kapittel 7 blant annet sett nærmere på sambandsbelastningen som resultat av oppdatering av situasjonsbildet og muligheter for konferanseapplikasjoner i divisjonen.

2 X.400

Det er anbefalt at meldingshåndteringssystemet i hæren skal være gjennomgående (1). Det vil blant annet si at det skal interoperere med meldingstjenesten i Forsvarets Digitale Nett (FDN). Siden dette vil bety at meldingshåndteringssystemet må benytte X.400, har vi sett nærmere på denne protokollen. Meldingshåndteringssystemet vil bli vurdert som bærer av kommando og kontroll-informasjon i tillegg til formidling av meldinger, så det kan komme til å spille en viktig rolle i divisjonens informasjonssystem.

2.1 Kort beskrivelse av komponentene i X.400

X.400-serien består av flere komponenter som hver beskriver ulike egenskaper ved et elektronisk meldingssystem. Det er beskrevet 4 logiske hovedkomponenter som samarbeider om å utføre de elektroniske meldingstjenestene. Dette er i følge (2) og (3):

- User Agent (UA)

Alle brukere vil ha sin egen UA, og dette er brukerens innfallsport til systemet. Oppgaven til UAen er å skrive, lagre, sende, motta og lese meldinger. Utforming av enkelte deler av en UA, f eks brukergrensesnitt og tekstbehandlingssystem, spesifiseres ikke i standarden. Standarden spesifiserer den informasjonen som skal utveksles mellom UAene. Når en bruker ønsker å sende en melding, må den informasjonen meldingssystemet trenger for å levere meldingen (f eks mottakerens adresse) samt den informasjonen som skal formidles til mottaker oppgis av avsender til avsenders UA. UAen er bindeleddet mellom bruker og Message Transfer System (MTS).

- Message Transfer Agent (MTA)

MTAen kan sees på som meldingssvitsjen i et X.400-nettverk, og sammenkoblede MTAer danner MTSet. Hver UA er koblet til en MTA, og hver MTA er innlemmet i et MTS. MTAene vil samarbeide om å bringe posten fra avsender til riktig(e) mottaker(e). MTAene bruker "store and foreward"-prinsippet, d v s at hele meldingen mottas i en MTA før den sendes videre til neste MTA. Det betyr at det ikke finnes noen garantert leveringstid, og at det er opp til de som konfigurerer nettet, samt MTAene å gjøre at leveringstiden blir så kort som mulig. MTAene er igjen gruppert i forskjellige Management Domain (MD). Av disse finnes det to typer; Administative Management Domain (ADMD) og Private Management Domain (PRMD) som blir styrt av henholdsvis en administrasjon eller en organisasjon.

- Message Store (MS)

MSet sørger for en sikker lagringsmekanisme, og dens posisjon er mellom UAen og MTAen. Den er imidlertid ikke nødvendig og kan utelates. Når UA ligger på en annen maskin enn den tilhørende MTA er den spesielt interessant, og den ble innført i X.400 standarden for å fjerne noe av problemet med at PCer kan være avslått. UAen vil da være utilgjengelig for MTAen. Når MS brukes, vil alle meldinger som er tiltenkt en UA leveres til MSet istedet, og MSet blir brukt som en slags postkasse. Bruker kan når som helst hente posten sin fra MSet ved hjelp av sin UA. En UA vil også sende utgående melding gjennom MSet dersom MSet er brukt mellom UA og MTA. Ett MS tjener kun en UA.

- Access Unit (AU)

AUen skal sørge for at brukere som har tilgang til meldingsformidling via andre kommunikasjonssystemer (f eks telex eller teletex) kan kobles til meldingssystemet. En melding som sendes til en bruker som bare har telex tilknytning vil da kunne leveres til denne brukeren som en vanlig telex. En spesiell AU er Physical Delivery Access Unit (PDAU) som

fungerer slik at meldingen kan skrives ut og leveres til brukere ved hjelp av det tradisjonelle postsystemet.

2.2 Hva skjer når man sender en melding

Nedenfor følger en oversikt over de forskjellige operasjonene som utføres ved en meldingsformidling (2). Punktene refereres til den skjematiske fremstillingen i Figur 2.1.

1) Avsender skriver innholdet og adressen til mottakeren (evt flere mottakere) og lager en melding v hj a UA software. Avsender gir så beskjed om å sende meldingen.

2) Avsenders UA komponerer P2-meldingen, og utfører en Message Submission Operation, hvor han overfører Message Submission Envelope og Content of the Message (P2-meldingen) til MTAen.

3) Avsenders MTA analyserer de nødvendige parametrene i Message Submission Envelope, og MTAen forteller så avsenders UA om den aksepterte eller forkastet meldingsoverleverelsen ved å sende en Message Submission Operation Result.

4) Hvis avsenders MTA aksepterte meldingsoverleveringen, lager denne MTAen en P1-melding av parametrene i Message Submission Envelope og noe internt generert informasjon. Denne P1-meldingen består av P1-envelope (Message Transfer Envelope) og P2-melding. P1-meldingen eller kopier av denne blir overført fra MTA til MTA til den når den MTAen som tjener mottakerens UA.

5) Mottakerens MTA analyserer P1-konvolutten og lager en liste over parametre som trengs for å overføre P2-meldingen til mottakerens UA. Denne analysen kan også føre til at MTAen ser at han ikke kan levere meldingen til mottakerens UA, f eks på grunn av formatet meldingen er skrevet på. Hvis MTAen konkluderer med at han kan levere meldingen utfører den en Message Delivery Operation. MTAen overleverer da UAen en Message Delivery Envelope og Message Delivery Content (P2-meldingen).

6) Mottakerens UA analyserer feltene i Message Delivery Envelopen for å se om den kan utføre meldingsoverleveringen til mottakeren. UAen gir så tilbakemelding til MTAen om den aksepterte meldingen ved å sende en Delivery Operation Result.

7) Etter at mottakerens MTA har utført overleveringen, undersøkes P1-headeren for å se om en Delivery Notification skal skrives. Eventuelt hvis den ikke har kunnet overlevere meldingen blir en Negative Delivery Notification skrevet. Mottakerens MTA sender da Report Transfer Envelope og Report Transfer Content til avsenders MTA. Denne Delivery Notificationen blir sendt fra MTA til MTA til den når avsenderen sin MTA.

8) Avsenderen sin MTA analyserer så Delivery Notificationen og lager en liste over parametre som trengs for å sende til avsenderens UA. Så utfører MTAen en Report Delivery Operation, hvor den sender avsenderen sin UA en Report Delivery Envelope og en Returned Content. Sistnevnte er den samme som Report Transfer Content. Informasjonen i Re-

port Transfer Envelopen blir ikke gitt UAen, men blir brukt av MTAen til administrasjonsformål.

9) Avsender sin UA angir om den aksepterte Report Deliveryen ved å sende en Report Delivery Operation Result til avsenderen sin MTA.

10) Mottakeren sin UA gir mottakeren mulighet til å lese meldingen.

2.3 X.400 og OSI-modellen

X.400-protokollen er plassert i applikasjonslaget i OSI-modellen (4) og benytter tjenester fra de generelle applikasjonstjenesteelementene Association Control Service Element (ACSE) (7), Remote Operations Service Element (ROSE) (8) og Reliable Transfer Service Element (RTSE) (7). I tillegg til de generelle applikasjonstjenesteelementene er det definert fem spesielle applikasjonstjenesteelementer for MTS:

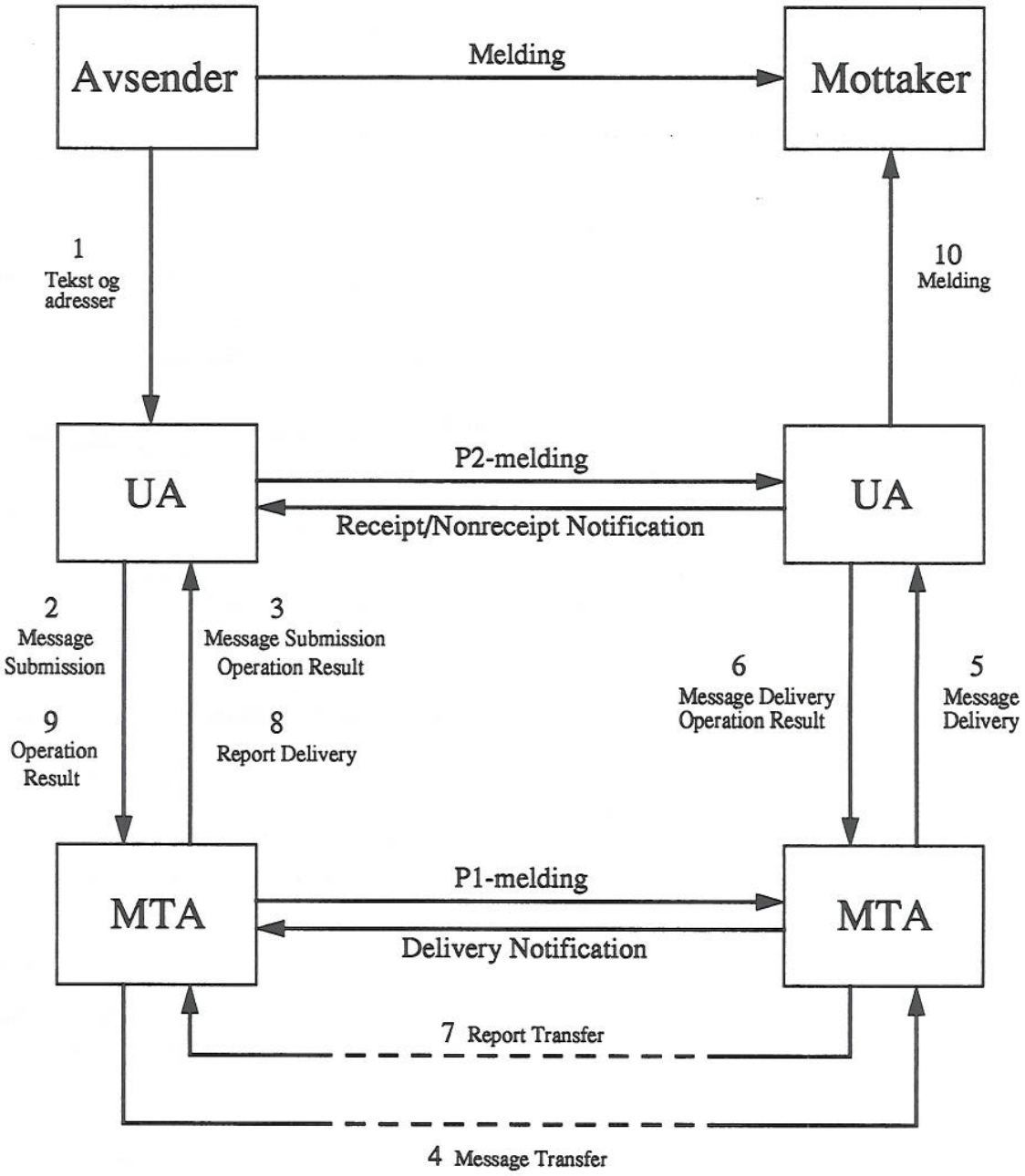
- Message Administration Service Element (MASE): Tillater en UA som interopererer med en MTA eller MS å utføre administrative funksjoner som å endre passord.
- Message Submission Service Element (MSSE): Tillater en UA å overlevere en melding til en tilkoblet enhet, f eks et MS eller en MTA.
- Message Delivery Service Element (MDSE): Tillater en MTA å levere en melding til en UA.
- Message Transfer Service Element (MTSE): Tillater to MTAer å utveksle meldinger pålitelig.
- Message Retrieval Service Element (MRSE): Tillater en UA å motta en melding fra et MS.

Videre benyttes tjenester fra presentasjonslaget (6) og kodingsreglene for Abstract Syntax Notation one (ASN.1) (5). Tjenester fra de lavere lagene benyttes på vanlig måte. Vi har konsentrert oss om lagene ned t o m sesjonslaget i OSI-modellen, det vil si applikasjonslaget, presentasjonslaget og sesjonslaget. Nedenfor er disse lagene beskrevet. (Innholdet er hentet fra (2) og (3).)

2.3.1 Applikasjonslaget

På applikasjonslaget er det definert 3 protokoller i X.400. Disse blir kalt P1, P3 og P7 og blir beskrevet nærmere nedenfor. Figur 2.2 viser hvilke komponenter som benytter hvilke protokoller.

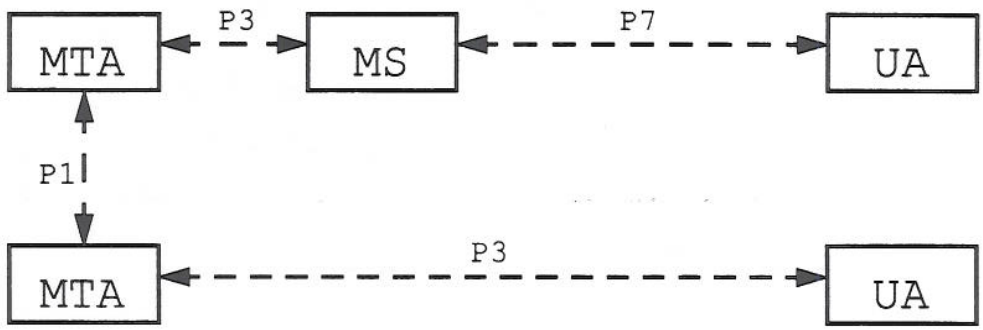
- P1-protokollen (MTA Transfer Protocol)



Figur 2.1 Skjematisk oversikt over hva som skjer når en melding blir sendt og X.400-protokollen blir benyttet.

- P3-protokollen (MTS Access Protocol)
- P7-protokollen (MS Access Protocol)

P1-protokollen beskriver hvordan to MTAer utveksler meldinger i X.400. P1-protokollen inneholder det MTS-spesifikke tjenesteelementet MTSE, hvor MTSE igjen benytter seg



Figur 2.2 Oversikt over hvilke komponenter som benytter hvilke protokoller

av ACSE og RTSE for henholdsvis å opprette /terminere forbindelser, få overført data over linken, utføre sjekking og flytkontroll o s v. ACSE og RTSE benytter seg av tjenester fra lavere lag i OSI-modellen.

P1-protokollen spesifiserer overføring av 3 typer meldinger. Dette er vanlige meldinger (konvolutt og innhold), tomme meldinger eller "probes" (konvolutt uten innhold som benyttes for å sjekke om en melding kan leveres) og kvitteringer (angir om en melding kan leveres eller ikke).

Av praktiske grunner blir P1 protokollen implementert oftere enn de to andre protokollene på applikasjonslaget. Dette er fordi man er avhengig av at sammenkoblete MTAer er fullstendig tilpasset hverandre, også to MTAer som tilhører to forskjellige MDer. Hvis derimot et MD velger noe annet enn P3- og/eller P7-protokollene ved de respektive grensesnittene vil dette ikke få noen konsekvenser ved meldingsutvekslingen mellom eller innenfor de forskjellige MDene.

Denne protokollen benyttes mellom UAen og MTAen, og den spesifiserer hvordan meldinger skal overleveres fra UA til MTA, og hvordan meldinger skal leveres fra MTA til UA. Her blir henholdsvis MSSE og MDSE benyttet. Videre benytter den seg av MASE til administrative formål. Disse tre Service Elementene bruker igjen ACSE for å opprette og terminere forbindelser, RTSE for å overføre meldinger i begge retninger, og ROSE for operasjoner som krever "request/reply"-type operasjoner (f eks endring av akkrediteringer). P3 benyttes også mellom MS og MTA, og en MTA vil dermed ikke skille mellom en UA og en MS.

Denne protokollen benyttes mellom UA og MS, og den spesifiserer hvordan meldinger skal hentes av UA til MS, og hvordan meldinger skal overleveres fra UA til MS. P7-protokollen er veldig lik P3-protokollen, bortsett fra at den bruker MRSE istedenfor MDSE. MRSE er det elementet som sørger for at UA kan få listet, laget sammendrag av eller slettet meldinger i MSet.

2.3.2 Presentasjonslaget

Presentasjonslaget har som formål å skjule forskjeller i måten kommuniserende applikasjonsprosesser representerer informasjonen på. Meningsinnholdet i informasjonen som utveksles må beholdes, mens forskjeller vedrørende formater og språk skjules.

Presentasjonslaget har 4 primære oppgaver:

- Lar brukeren få mulighet til å utføre sesjonstjenesteprimitivene.
- Angir en måte å spesifisere komplekse datastrukturer på.
- Ha kontroll på mengden av datastrukturer som for tiden blir krevd.
- Konvertere data mellom intern og ekstern form.

De tre nederste oppgavene henger sammen, og omhandler hvordan datastrukturer blir beskrevet, brukt og kodet på ved transmisjon (4).

Alle tjenestene som X.400 bruker på presentasjonslaget blir mappet over til tilsvarende tjenester på sesjonslaget. Det er derfor rimelig å tro at overheaden som blir lagt på meldingen på dette laget ikke har noen stor betydning for den totale overheaden som blir lagt på meldingen. De tjenestene som legger på overhead, d v s der hvor parametre blir sendt over til mottakerens presentasjonslag, ser ut til å være kun de som blir brukt ved opprettelse og terminering av en forbindelse.

2.3.3 Sesjonslaget

Sesjonslaget styrer dialogen mellom applikasjonsprosesser ved hjelp av sesjonsforbindelser, og tillater to sesjonsenheter å styre sin dialog/forbindelse. Når en ser på to sammenkoblede MTAer inkluderer dette følgende (2):

- Sesjonslaget tillater begge sider å overføre meldinger på en ryddig måte.
- Det tilbyr synkronisering, feilsjekking og tjenester for å gjenoppta transmisjon fra riktig sted i meldingen etter feil ved synkroniseringen.
- Det tilbyr transmisjon av meldinger i begge retninger gjennom en toveis alternerende sesjon i tilfeller hvor en har kun en kanal.

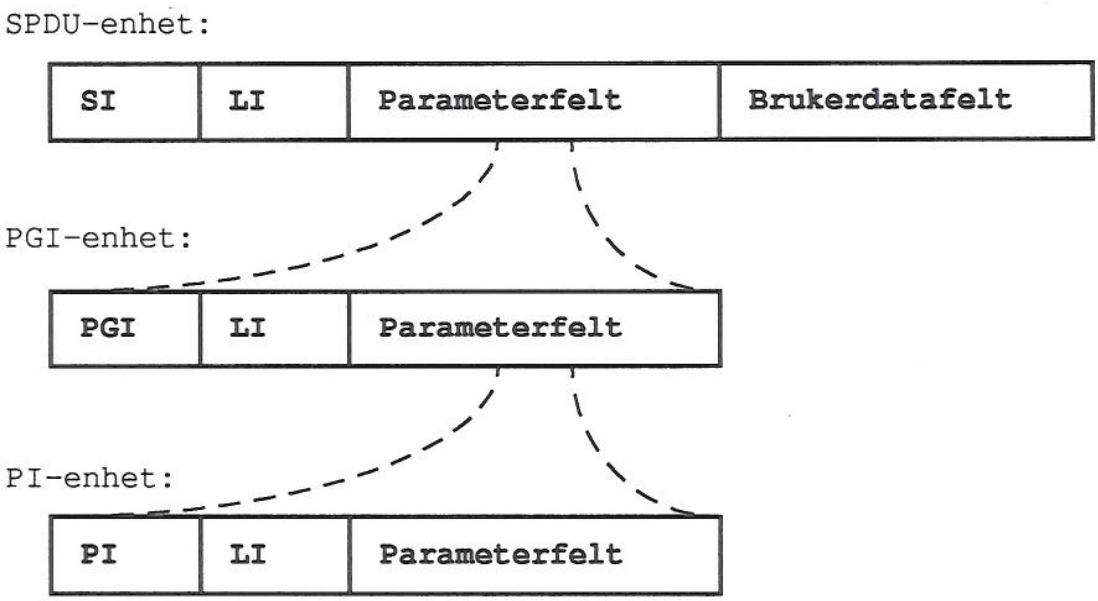
Sesjonslaget holder oversikt over meldinger som blir sendt ved at hver melding som blir sendt tilsvarer en aktivitet. Dette gjør at høyere lag hos mottakeren kan bestemme når en melding starter og slutter ved at det blir angitt når aktiviteten starter og slutter. Det kan sendes flere meldinger på en sesjon.

Videre tilbys tjenester for synkronisering av datastrømmen. Synkroniseringstjenesten gjør det mulig for applikasjonsprosessene entydig å definere og referere til synkroniserings-

punkter i datastrømmen. Det er da enkelt å oppdage feil med forbindelsen, og ved feil med synkroniseringen kan forbindelsen gjenopptas fra et nærmere angitt synkroniseringspunkt. Man slipper derfor å sende hele meldingen på nytt.

Under oppkobling av en sesjonsforbindelse avtaler de to applikasjonsprosessene hvilke dialogregler som skal gjelde for dialogen mellom dem. Alternativene er duplex, halv duplex og simplex. Ved bruk av halv duplex kan dialogen i tillegg styres av tokener som fungerer som stafettpinner. De regulerer hvilke av partene som til enhver tid har lov til å sende data eller utføre bestemte operasjoner.

Sesjonslagene sender hverandre Session Protocol Data Units (SPDU). Det finnes forskjellige SPDUer, f eks Connect-SPDU, Data-SPDU og Give Token-SPDU. Deres funksjoner er henholdsvis å opprette en forbindelse, sende brukerdata og å gi tokenen til den andre parten slik at han kan sende. Informasjonen i en SPDU er angitt i parameterfeltet, bortsett fra brukerdata som blir gitt i brukerdatafeltet. Parametre som er relatert til hverandre er samlet i parametergrupper. En SPDU starter med en SPDU-identifikasjon (SI), kalt SPDU-identifikasjonsfeltet, etterfulgt av en lengdeidentifikator (LI), som igjen etterfølges av de SPDU-spesifikke parametrene. LI assosiert til en SPDU inkluderer ikke lengden på SI-feltet, LI-feltet eller lengden på brukerdata hvis det er snakk om en Data-SPDU. En parametergruppe blir angitt med en parametergruppeidentifikator (PGI) og et lengdefelt (LI) i tillegg til parametrene. Likeledes angis hver og en parameter med en parameteridentifikator (PI) og et lengdefelt (LI). På samme måte som for SPDUene inkluderer lengdefeltene til parametergruppene og parametrene ikke lengden på PGI- eller PI-feltet, heller ikke på LI-feltet selv.



Figur 2.3 Oversikt over hvordan en SPDU-enhet er bygd opp.

Det blir lagt til en overhead i tillegg til selve brukerdataene i SPDUen, som vist i Figur 2.3. Denne overheaden er betydelig større enn den som blir lagt til på tilsvarende måte på presentasjonslaget. SI-, PGI-, og PI-enhetene består av bare 1 byte, mens de respektive lengdefeltene består av 1-3 byte. Parameterfeltene kan imidlertid være lange, f eks kan parameterfeltet på en Connect-SPDU være nesten 200 byte langt. Men, hvis man bare benytter seg av det minimum av parametre som er påkrevd, kan man klare seg med kun 2 byte. Vi er derfor avhengig av å vite mer nøyaktig hva slags parametre som blir valgt i Taktisk Meldingshånteringssystem (TMHS), for å kunne si noe bestemt om størrelsen på overheaden. I (2) vises et eksempel på en melding med hele overheaden (alle 7 lag i OSI-modellen), som viser at den totale overheaden her ligger på rundt 500 byte. Vi har også fått opplyst fra Alcatel at overheaden på en melding i det strategiske meldingssystemet har samme størrelse. Våre målinger er vist i kapittel 5.

Det er oppgitt i Systemspesifikasjonen til TMHS (11) at det er foretatt reduksjoner i presentasjonslaget og sesjonslaget for å minimere overheaden. Vi vet ikke hva disse reduksjonene går ut på.

3 DISTRIBUERTE DATABASER

En ubetinget fordel ved manøverkrigføring med desentralisert utførelse av ordre er et effektivt K2I-system der alle offiserer sitter med det samme oppdaterte situasjonsbilde. Det betyr at deling av informasjon er viktig, og det er da naturlig å se dette i sammenheng med distribuerte databaser. Dette er en mye brukt teknologi i dag, men det stilles allikevel litt andre krav på divisjonsnivå i hæren enn det gjør i det sivile. Hovedårsakene til dette er at sambandet er lite pålitelig, kravet til sikkerhet er stort og det eksisterer forskjellige graderingsnivå det må tas hensyn til. I dette kapittelet blir sentrale egenskaper til et slikt system presentert, samtidig som spesielle løsninger for divisjonen blir diskutert.

3.1 Krav til transaksjoner

Prosesser for å lese og oppdatere databaser blir kalt transaksjoner. I databaser har man satt visse krav til transaksjonene (10). Dette er følgende:

- Atomisitet (Atomicity): Enten blir alle handlingene i en transaksjon fullstendig utført, eller så blir alle kansellert.

- Konsistens (Consistency): Transaksjonene transformerer databasene fra en konsistent tilstand til en annen.

- Isolasjon (Isolation): En handling utført av en transaksjon mot en felles database kan ikke vises til andre transaksjoner før transaksjonen er fullstendig utført.

- Holdbarhet (Durability): Når en transaksjon har blitt fullstendig utført, må systemet garantere at resultatene av operasjonen aldri vil bli mistet, uavhengig av feil som måtte oppstå senere.

- Serieutførbarhet (serializability): Hvis flere transaksjoner blir utført samtidig, må resultatet bli det samme som om de ble utført i serie i en eller annen orden. Det å garantere transaksjonenes serieutførbarhet blir kalt samtidighetskontroll (Concurrency Control). Hvis systemet tilbyr samtidighetskontroll, kan programmereren skrive transaksjonen som om den blir utført alene.

Når det gjelder holdbarhet betyr dette at oppdateringer bør lagres på disk.

Atomisitet og konsistens er det mest vanlig å ivareta ved å bruke 2-fase-utførelsesprotokollen (2-phase-commitment-protocol). Denne protokollen har en agent som blir kalt koordinator, mens de andre som er med på transaksjonen blir kalt deltakere. Koordinatoren har ansvaret for å avgjøre om oppdateringen skal utføres eller aborteres. Deltakerne har ansvaret for å oppdatere i sine lokale databaser. Som det fremgår av navnet består protokollen av to faser. Målet med den første fasen er å nå en felles avgjørelse, mens målet med den andre fasen er å implementere avgjørelsen. Protokollen består av følgende (10):

Koordinator:
Skriv "prepare" inn i loggen.
Send PREPARE-melding og aktiver timeout.

Deltaker:
Vent på PREPARE-melding.
Hvis deltakeren vil utføre transaksjonen, så:
    Skriv deltransaksjonen inn i loggen.
    Skriv "ready" inn i loggen.
    Send READY-svarmelding til koordinator
Ellers:
    Skriv "abort" inn i loggen.
    Send ABORT-svarmelding til koordinator.

Koordinator:
Vent på svarmelding fra deltakerne eller timeout.
Hvis timeout-tiden utløper eller noen av svarmeldingene er ABORT, så:
    Skriv "global-abort" inn i loggen.
    Send ABORT-kommando til alle deltakerne.
Ellers (hvis alle svarmeldingene er READY):
    Skriv "global-utførelse" inn i loggen.
    Send UTFØR-kommando til alle deltakerne.

Deltaker:
Vent på kommando.
Skriv "abort" eller "utfør" inn i loggen.
Send BEKREFTELSE til koordinator.
Utfør kommando.

Koordinator:
Vent på BEKREFTELSE fra alle deltakerne.
Skriv "fullført" inn i loggen.

Vi ser at alle eller ingen av de distribuerte databasene blir oppdatert slik at atomisitet- og konsistenskravene blir oppfylt. Hvis også koordinator og deltakerne låser sine dataenheter under denne protokollen vil kravet til isolasjon også være oppfylt.

En sentral problemstilling i forbindelse med administrasjon av databaser er å gi flere samtidige brukere tilgang til en database. Dette krever løsning av mange problemer som ikke ville oppstått hvis alle transaksjoner ble tidsstyrt slik at en transaksjon var ferdig utført før den neste fikk anledning til å begynne. For å kunne støtte flere samtidige brukere er det nødvendig å kunne kontrollere lesing og skriving på flere steder i databasen samtidig, med andre ord en samtidighet eller parallellitet i systemet. Vi må med andre ord ha serieutførbarhet.

Hvis en lar transaksjoner bli utført samtidig uten at dette kontrolleres vil disse forstyrre hverandre. Interferens mellom samtidige transaksjoner kan føre til f eks:

- Problemer med tapte oppdateringer: Endring av data p g a at en transaksjon som tilsynelatende er ferdig overskrives av en annen transaksjon før den virkelig er ferdig.

- Overtredelse av integritetssperrer: Dette er en situasjon som kan oppstå hvis en tillater utførelse av to transaksjoner samtidig uten at disse er synkronisert. Transaksjoner kan ha avhengigheter selv om de oppdaterer forskjellige data. Et problem som oppstår hvis en oppdaterer data uten å se på helheten.

- Problem med inkonsistent lesing: Lesing av delresultater fra transaksjoner som ennå ikke er ferdige.

For å hindre disse problemene med interferens er det en modul i styreprogrammet som driver med det som kalles samtidighetskontroll. Denne modulen kalles "scheduler" og utfører en form for tidsstyring av transaksjonene. Som tidligere nevnt har en samtidighetskontroll hvis en kan garantere transaksjonens serieutførbarhet. Lese- og skriveoperasjonene for hver transaksjon må da være organisert slik at transaksjonene kan endre rekkefølge uten at sluttresultatet endres. Hvis dette er oppfylt vil sluttresultatet være det samme selv om noen av transaksjonene blir utført i parallell og noen i serie. Dette er det nødvendig å få til for å håndtere flere samtidige brukere i en sentralisert database eller flere samtidige endringer i en distribuert database.

For å realisere samtidighetskontroll er det 3 teknikker som danner grunnlaget:

- Metoder med låsing er mest brukt. Der må transaksjonen kreve en leselås som kan deles eller en skrivelås som denne transaksjonen er alene om. Størrelsen på dataenheten som låses er av stor betydning for ytelsen til algoritmen for samtidighetskontroll. Denne størrelsen bør tilpasses til den aktuelle databasen. Den mest vanlige metoden med låsing er to-fase-låsing (two-phase-locking 9. Metoden har mange fellestrekk med 2-fase-utførelsesprotokoller. Hovedproblemene med denne metoden i distribuerte databaser er stor mengde overhead og mulighet for å komme inn i en ond sirkel (deadlock) d v s at to transaksjoner venter på å kunne låse data som allerede er låst av den andre transaksjonen. Det kreves mekanismer for å detektere eller hindre at en kommer inn i en ond sirkel.

- Tidsstemplingsmetoder benytter en tidsstyring slik at en får serieutførbarhet. Alle transaksjoner får en tidsmerking slik at det er mulig å ordne transaksjonene globalt. Dette gjør det nødvendig å definere en global tid i distribuerte systemer. Gamle trans-

aksjoner får prioritet hvis det skulle oppstå en konflikt. Hvis denne prioritetshåndteringen ikke er nok startes transaksjonen opp på nytt med et nytt tidsmerke. Ved tidsstempling er det ingen venting eller låsing og derfor heller ingen deadlocks. Transaksjoner som kommer i konflikt blir startet på nytt.

- Optimistiske metoder bygger på at det er mulig å klare seg uten kompliserte synkroniseringsmetoder hvis en før en transaksjon utføres sjekker om det er konflikt. Hvis det er konflikt startes transaksjonen på nytt. For å sikre atomisitet ved denne metoden blir alle oppdateringene gjort på en kopi som er lokal for transaksjonen. Endringene blir bare overført til databasen ved "utfør" hvis ingen konflinkt har blitt detektert.

3.2 Pålitelighet

Det kan imidlertid oppstå feil både på maskiner der en database ligger og på selve nettet som binder databasene sammen under en transaksjon. Det er ønskelig at de fem kravene diskutert i avsnittene over også i slike tilfeller skal være oppfylt. Vi sier at en protokoll er "blokkerende" hvis noen former for feil tvinger de deltakende enhetene å vente med å fullføre transaksjonen til feilen er reparert, og at enhetene da må stå låst i mellomtiden.

3.2.1 3-fase-utførelsesprotokoll

Ved å utvide 2-fase-utførelsesprotokollen med en fase til, kan denne protokollen bli antiblokkerende med hensyn på feil med deltakende maskiner. Svakheten med 2-fase-utførelsesprotokollen er at de deltakende enhetene får problemer hvis koordinatoren går ned mellom 1. og 2. fase, og ingen av de deltakende enhetene har mottatt melding om å utføre eller abortere transaksjonen. De vet nemlig ikke om koordinatoren selv har utført eller abortert transaksjonen før han gikk ned. Det blir derfor umulig å garantere konsistens hvis ikke man skal blokkere de deltakende enhetene til koordinatoren er oppe igjen.

Med 3-fase-utførelsesprotokoll utfører ikke de deltakende enhetene transaksjonen i 2. fase. Istedenfor havner de i en ny tilstand kalt forberedt på utførelse (prepare-to-commit). Man trenger derfor en 3. fase for å utføre transaksjonen. Denne protokollen eliminerer blokkeringsproblemene til 2-fase-utførelsesprotokollen. I 2. fase (og også mellom 1. og 2. fase) vet man nå at koordinatoren hverken har utført transaksjonen eller abortert. Hvis i det minste en deltakende enhet har kommet i "forberedt på utførelse"-tilstanden, kan transaksjonen bli utført. Og hvis minst en deltakende enhet ikke har kommet i "forberedt på utførelse"-tilstanden, kan transaksjonen bli abortert. Det som skjer er at man velger en ny koordinator blant de deltakende enhetene som avgjør om transaksjonen skal aborteres eller utføres, og som deretter koordinerer dette. Det er heller ikke noe problem om koordinatoren skulle gå ned i 3. fase. Man kan på samme måte velge en ny koordinator og fullføre transaksjonen. Alle deltakende enheter er da i forberedt på utførelse-tilstanden. Den opprinnelige koordinatoren vil i tilfellene diskutert over fullføre eller abortere transaksjonen når den blir reparert.

3.2.2 Nettverksfeil

Det finnes ikke protokoller som er anti-blokkerende når det er feil med nettverket slik at det oppstår grupper som bare har kontakt internt og ikke gruppene seg i mellom. Det er vist at en slik anti-blokkerende protokoll bare eksisterer hvis systemet blir delt i kun to separate grupper, og alle uleverte meldinger blir levert tilbake til sender. Dette er imidlertid uoppnåelig i praksis, og vi må derfor godta metoder som er blokkerende om vi vil ha fullstendig konsistens.

De to vanligste suboptimale metodene ved nettverksdeling, er primærenhetsmetoden (Primary Site Approach) og flertallsmetoden (Majority Approach and Quorum-Based Protocols). Primærenhetsmetoden går ut på å velge seg en primærenhet, og så si at gruppen som inneholder denne enheten alltid skal fullføre transaksjonen. Ved 2-fase-utførelsesprotokollen må koordinator alltid være primærenhet for å få dette til. Men dette valget vil føre til at systemet blir sårbart ovenfor feil med primærenhet. En bedre protokoll å velge vil kanskje være 3-fase-utførelsesprotokollen. Her vil transaksjonen bli fullført i gruppa som inneholder primærenhet som om de andre enhetene i de andre gruppene var nede, og koordinator trenger ikke være primærenhet.

Hovedideen med flertallsmetoden er at bare gruppen som inneholder flertallet av enhetene kan fullføre transaksjonen. En svakhet med metoden er imidlertid at hvis ingen grupper oppnår flertall, vil alle gruppene bli blokkert. Metoden går ut på at en av gruppene må inneholde så mange enheter at over halvparten av enhetene må være enige om å abortere eller fullføre transaksjonen før transaksjonen blir utført eller abortert.

3.3 Tilgjengelighet kontra konsistens

Spørsmålet man må stille seg er om vi på taktisk nivå i hæren kan godta at enhetene er blokkert fordi sambandet er nede. At linker går ned må vi anta skjer relativt ofte, og i verste fall vil de da være nede i flere timer. Det ser ut som om det kan være fornuftig å øke tilgjengeligheten ved å redusere litt på kravene til at databasene skal være konsistente. Dette har de gjort i Norwgian Command Control and Information System (NORCCIS II), og der har de valgt en løsning hvor transaksjoner blir utført ved hjelp av databaseaksessprotokollen SQL*Net fra Oracle og Transmission Control Protocol/ Internet Protocol (TCP/IP)-protokollen. En annen mulighet er å bruke X.400-protokollen og meldingssystemet. Her vil overføringer av data være godt sikret, men prisen for dette er at vi får større overhead. Denne muligheten kan utvides til å bruke X.400 til å oppdatere data, og SQL*Net til å lese data fra eksterne databaser. En tredje mulighet er å bruke Web'en og intranett til å lese data. Her er det HTTP-protokollen (omtalt i kapittel 4) som blir benyttet.

For at ikke flere skal oppdatere samme enhet samtidig har man i NORCCIS II definert eierskap til dataene. De kommandoplassene som ikke har samband vil da ikke få oppdatert dataene sine så lenge sambandet er nede, men de vil ha tilgang på de "gamle" dataene sine. Vi forutsetter derfor at det er bedre å ha uoppdaterte data, enn å ikke ha tilgang på noe. I tillegg har det med eierskap en hensikt rent operasjonelt, ved at ansvar fordeles rundt i divisjonen.

Problemstillinger rundt eierskap blir diskutert videre i kapittel 7.1.1.

3.4 Deteksjon av inkonsistente data og kald omstart

En vanlig metode å bruke for å finne inkonsistente data er å definere versjonsnummer, $V(x,y)$. x kalles det originale versjonsnummeret, mens y kalles nåværende versjonsnummer. Sistnevnte starter på 1, og økes med 1 for hver oppdatering, mens det originale versjonsnummeret starter på 0 og blir satt lik det nåværende versjonsnummeret når feil oppstår. På denne måten vil det originale versjonsnummeret lagre det nåværende versjonsnummeret på de isolerte kopiene før noen oppdateringer er utført på dem. Det originale versjonsnummeret blir ikke endret før sambandet er reparert. Da kan sammenligning av det nåværende og det originale versjonsnummeret på kopier avsløre inkonsistensen.

Kald omstart blir krevd etter store feil som har ført til at loginformasjonen på diskene har gått tapt. Den daværende konsistente tilstanden på databasene kan dermed ikke bli rekonstruert, og man må derfor finne frem til en tidligere konsistent tilstand. En slik tidligere konsistent tilstand må være markert med et kontrollpunkt.

Å oppnå en tidligere konsistent tilstand kan være komplisert, spesielt i distribuerte databaser. Problemet er å sette kontrollpunkter. En måte å sette kontrollpunkter på kan være å bruke en protokoll tilsvarende 2-fase-utførelsesprotokollen, der man i stedet for å oppdatere data setter et kontrollpunkt. Denne metoden er imidlertid ikke brukbar i praksis, fordi den krever at hele databasen er inaktiv i en lengre periode. Dessuten får man ikke satt kontrollpunkter så lenge en maskin eller sambandet er nede. Det har heller ingen hensikt for oss at kontrollpunktene blir satt så likt i tid så lenge vi godtar inkonsistens under oppdatering og ved tekniske feil. Oppdateringene på samme dataenhet vil dermed ikke skje samtidig. Da vil en bedre metode være at hver distribuerte database lagrer separate kopier av de dataenhetene som de har eierskap på, og at dette skjer ved faste tidsintervaller. På denne måten vil det være enkelt å få rekonstruert en konsistent tilstand.

3.5 Metoder for reduksjon av trafikkbelastning ved sammenkobling av distribuerte data

Ved operasjoner hvor data fra flere distribuerte databaser skal sammenkobles, kan man spare mye datatransmisjon ved å bruke spesielle algoritmer. Sammenkoblingen skjer da ved hjelp av spesielle nøkler som f.eks. personnummer i databaser hvor data om forskjellige personer er lagret. En algoritme som blir brukt ved slike reduksjoner er SDD-1-algoritmen. Kort fortalt kan man si at denne algoritmen prøver å redusere tabellene som skal transmitteres over til det stedet hvor sammenkoblingen skal skje slik at ikke alle dataene som er registrert i tabellen må sendes over. Til dette brukes "semi-join" operasjoner (10), og reduksjonen skjer før de blir sendt over. Likeledes velges det stedet som det er mest optimalt å utføre sammenkoblingen på med hensyn på transmisjonskostnader. Når det gjelder informajon som skal lagres rundt i divisjonen, ser det ut til at informasjon med felles nøkler blir lagret på samme sted (12). Det vil derfor ikke være hensiktsmessig å implementere slike algoritmer her.

4 WEB-NETTVERK, FTP OG TELNET

For at sluttbrukere skal få tilgang til data på sin egen arbeidsplass vil det ofte være nødvendig med kommunikasjon med en annen datamaskin. Dette kan være en felles maskin med en database i kommandoplassen eller en tjenermaskin som dekker et behov hos brukere i flere kommandoplasser. Sluttbrukerens behov for henting av informasjon til egen arbeidsplass kan dekkes ved å hente filer med f eks FTP, fjerntilgang på en maskin v h a TELNET eller spørring etter informasjon v h a et Web-nettverk. Dette nettverket kan være innefor en kommandoplass eller omfatte et større område.

HTTP-protokollen, brukt i Web-nettverk, kan være en alternativ protokoll for å få tilgang på informasjon. Det er en protokoll som blir stadig mer populær, og dens styrke er at den fungerer som et enkelt men kraftig oppslagsverktøy. Den blir brukt til å utføre en rekke funksjoner i et intranett eller på Internet, og målsettingen med protokollen er at den skal være kompakt og påføre liten belastning på nettverket. I tillegg til et sett innebygde metoder har denne protokollen muligheter for metodeutvidelser.

HTTP-protokollen består av to typer pakker; forespørselspakker og responspakker. HTTP-klienten sender forespørselspakker til HTTP-tjeneren, mens HTTP-tjeneren sender responspakker tilbake. En og samme maskin kan inneha både tjener- og klientrolle.

For å overføre filer fra en annen maskin over et datanett kan man bruke FTP. Dette er en protokoll for overføring av komplette filer mellom datamaskiner, mens TELNET blir brukt når man ønsker å logge seg inn på en annen maskin. Dette er applikasjonsprotokoller som er tilgjengelig på Unix-maskiner og flere andre maskiner. Disse protokollene bygger på Advanced Research Projects Agency Network (ARPANET) (4) sine transport- og nettverksprotokoller. Begge protokollene gir forholdsvis mye overhead som eksemplene senere vil vise.

FTP benytter seg av to virtuelle forbindelser på transportlaget for å kontrollere overføring av filer mellom to maskiner. En av forbindelsene brukes for å overføre kommandoer og responser, og den andre brukes for å flytte filer. FTP benytter en klient/tjener modell. Kontrollforbindelsen brukes bl a til å beskrive filoverføringsformater på binær form, på American Standard Code for Information Interchange (ASCII)-format el l.

TELNET-protokollen tilbyr generell bidireksjonal 8-bit byteorientert kommunikasjon. Protokollen tilbyr en standard metode for grensesnitt mellom terminaler og terminalorienterte prosesser. TELNET opererer med noe som kalles Network Virtual Terminal (NVT) som er en tenkt utstyrsenhet som danner en slags standard for terminaltilknytning. I tillegg er det mulig for maskinene å forhandle om terminalparametre slik at terminalen kan opptre som mer sofistikert enn NVT.

5 NETTBELASTNING VED BRUK AV DE ULIKE PROTOKOLLENE

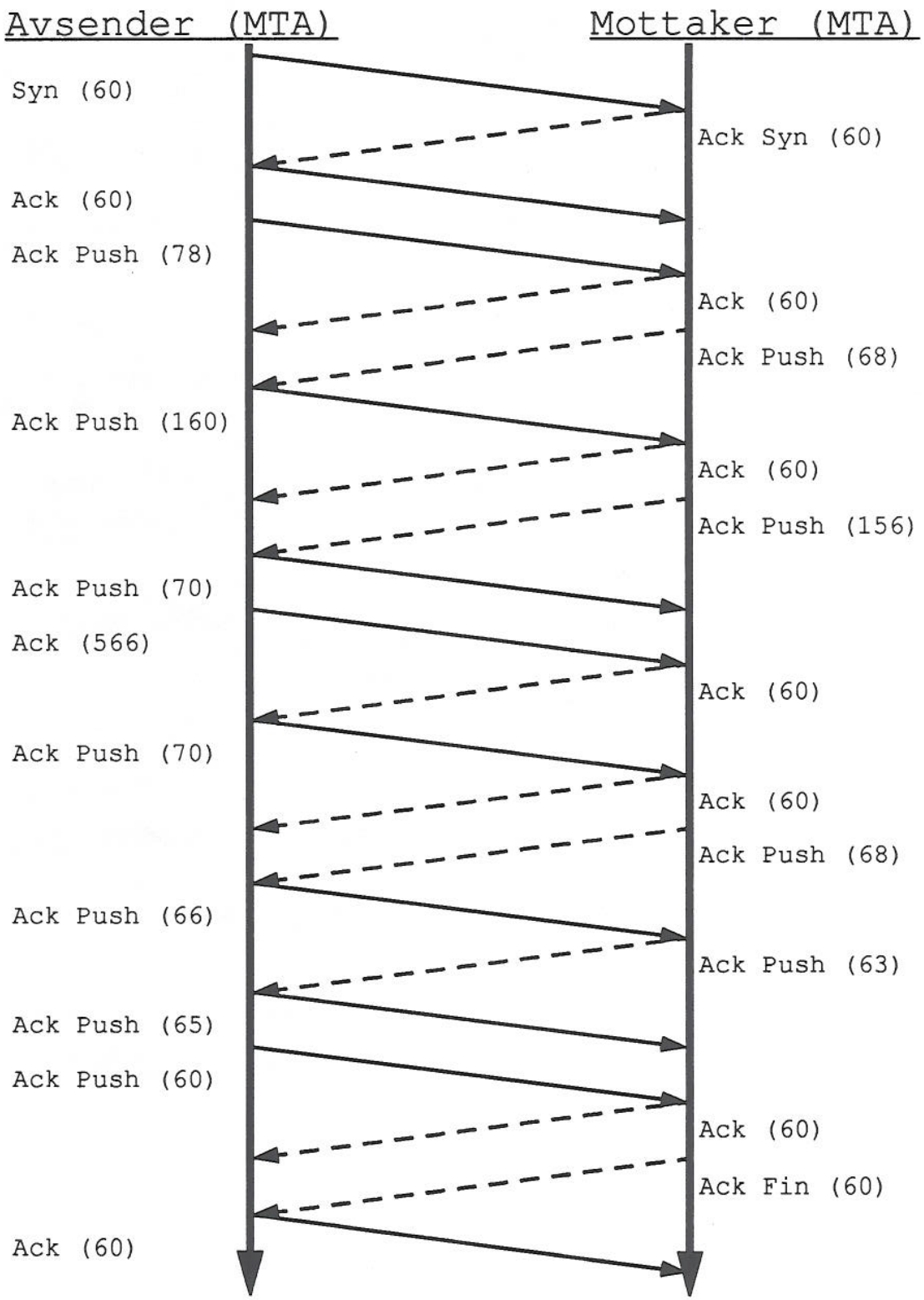
Siden sambandskapasiteten i divisjonen er begrenset, er en avgjørende faktor ved valg av informasjonsdistribusjonsløsning hvor mye trafikkpåtrykk som blir generert ved de ulike løsningene. Det har vært vanskelig å finne ut nøyaktig hvordan disse protokollene fungerer, og hvor stor overhead de vil gi. Vi har derfor valgt å undersøke dette ved å se på trafikken som blir generert ved bruk av de ulike protokollene og programmene på et lokalnett. På denne måten har vi fått en grov oversikt over hvor stor overhead de ulike måtene å overføre data på vil gi. Det ble brukt analyseprogramvare som viste påtrykk på ethernetnivå. Over nettet der vi målte ble det satt opp TCP/IP-forbindelse og ikke X.25-forbindelse som divisjonen skal benytte. Siden det ikke oppstod feil på kanalen eller ved ruting hadde vi oversiktlige forhold og antar derfor at resultatene fått ved målinger på TCP/IP-forbindelsen kan sammenlignes med dem vi ville fått med X.25. Forskjellen på protokollene TCP/IP og X.25 er beskrevet i avsnitt 5.4.

5.1 X.400

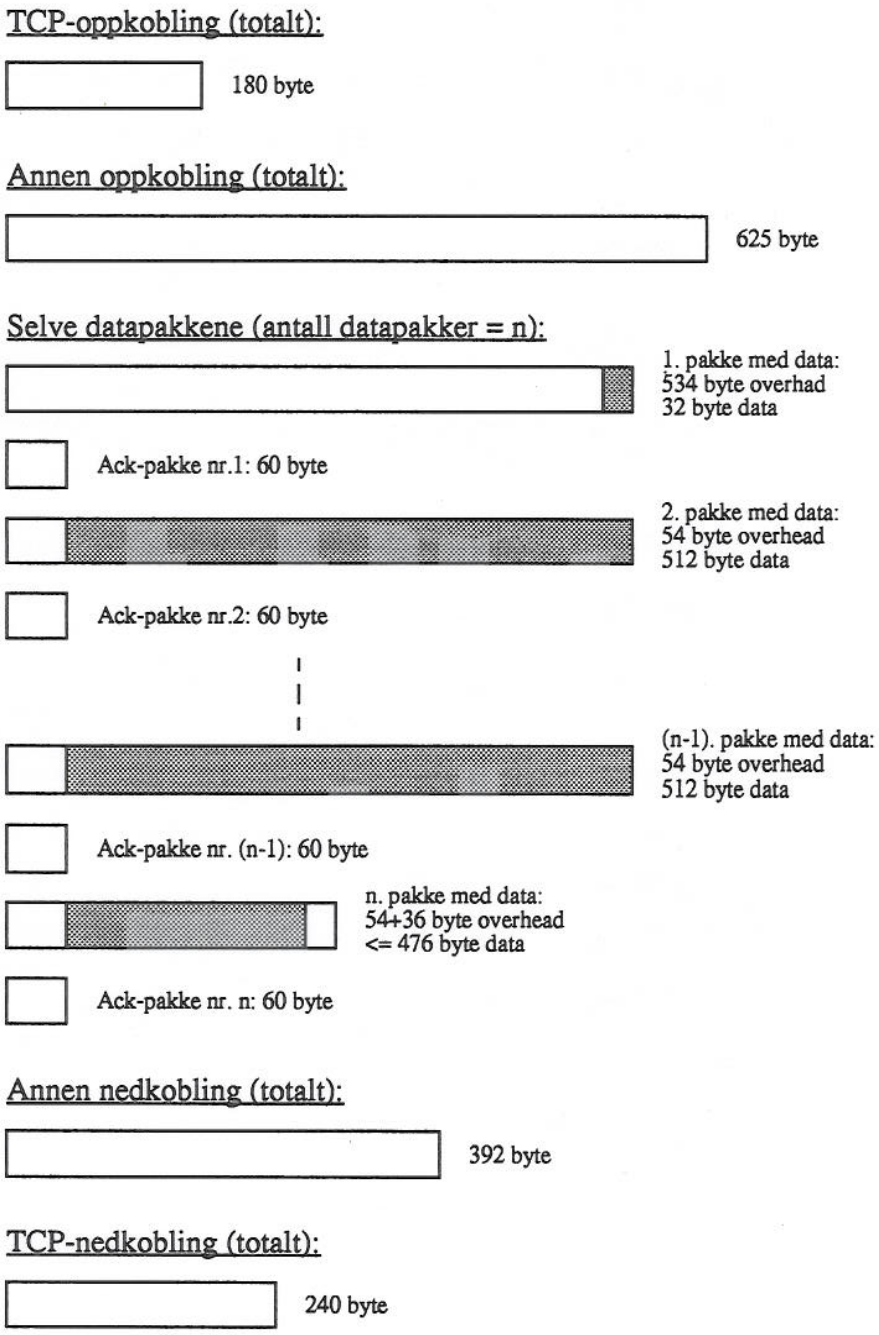
Her målte vi trafikkpåtrykket mellom to MTAer. Meldingens størrelse var neglisjerbar, da tittel og tekst var på tilsammen 13 tegn (13 byte). Trafikken som ble generert mellom avsenderens MTA og mottakerens MTA er vist i Figur 5.2.

Når det gjelder X.400-meldinger har vi undersøkt disse litt nærmere, og funnet ut at den totale overheaden kan beskrives som vist i Figur 5.2. TCP-oppkoblingen består av 3 pakker på 60 byte, mens TCP-nedkoblingen består av 4 pakker på 60 byte.

Ved oppkobling av en forbindelse blir maskinene enige om hvor store pakker de skal ha lov til å sende, og hvor stort "vindu" de skal bruke. Vindusstørrelsen sier noe om hvor mange byte som kan bli sendt før man må vente på en "Ack"-melding. Naturlig nok vil bestemmelsen av begge disse størrelsene innvirke på trafikken som blir sendt. På nettet hvor vi målte trafikken ble maksimalstørrelsen på pakkene satt til 566 byte, mens vindusstørrelsen ble satt til 4096 byte.



Figur 5.1 Eksempel på hva som blir sendt mellom to MTAer under en meldingsformidling med X.400, der en TCP/IP-forbindelse ble satt opp. Det er også angitt hva slags type pakke som blir sendt, og hvor stor den er (oppgitt i byte) på ethernetnivå.



Figur 5.2 Oversikt over hvordan overheaden er fordelt i en X.400-melding. Skravert område angir informasjon, mens hvitt område angir overhead. Størrelsen på pakkene er angitt på ethernetnivå.

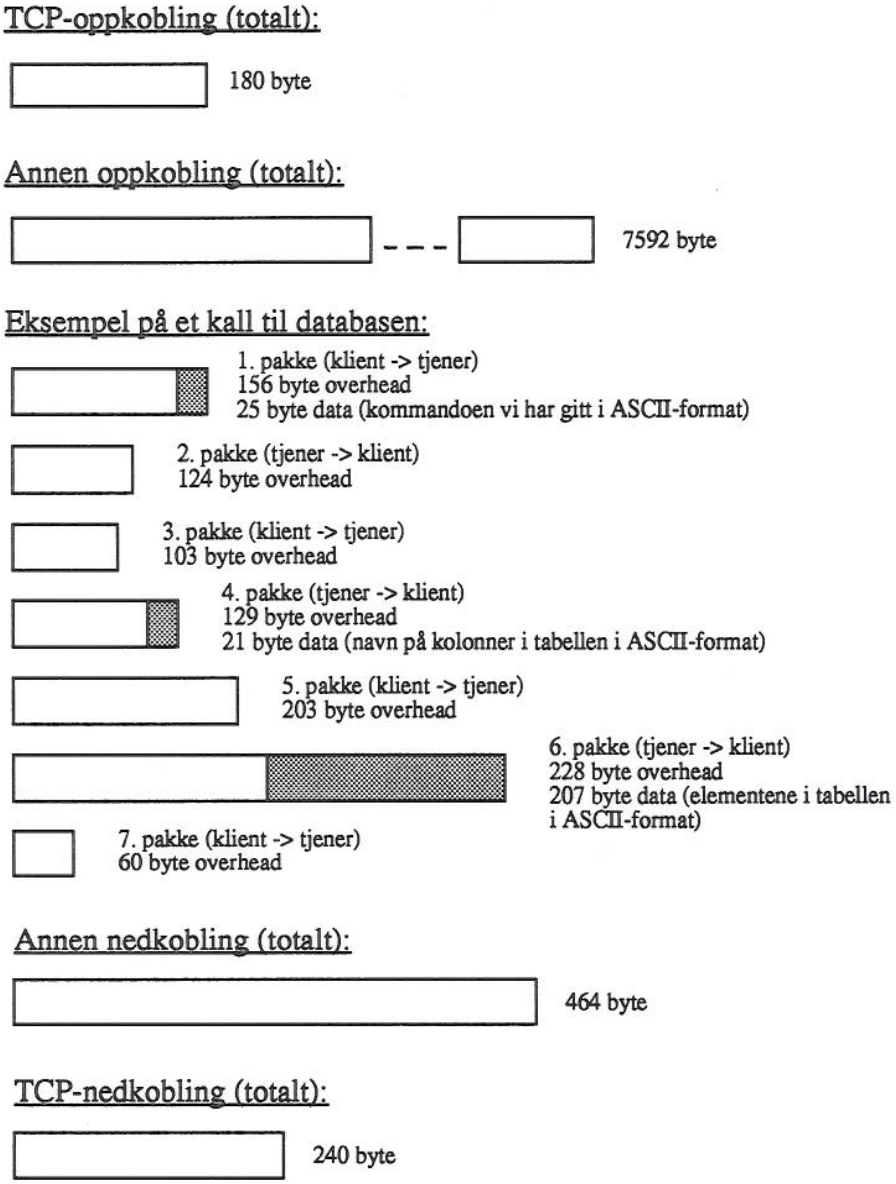
5.2 Databasekommunikasjon

Da vi skulle undersøkte trafikken som ble generert ved databasekommunikasjon lagde vi først en tabell i en Oracle-database på tjenermaskinen. Deretter hentet vi ut data fra denne tabellen med bruk av SQL*Net mens vi satt på klientmaskinen. Det viste seg at mange og tildels store pakker ble sendt over nettet når vi startet applikasjonen SQL*Plus, men når vi hadde startet programmet var forbindelsen oppe helt til vi gikk ut av det.

SQL*Net fungerte tydeligvis slik at kommandoen vi ga ble sendt i ASCII-format fra klient til tjener. Hvis vi hadde bedt om data, fikk vi en pakke tilbake som inneholdt kolonnenavnene til tabellen vi hadde bedt om, og en pakke som inneholdt elementene i tabellen. I tillegg ble det sendt noen få pakker i begge retninger som ikke inneholdt noen data, men bare overhead. Hvis vi bare oppdaterte databasen fikk vi ingen data-pakker tilbake. Figur 5.3 viser pakkene som ble sendt over nettet da vi hentet en liten tabell fra databasen. Tabellen inneholdt tre kolonner og ni rader. Størrelsen på overheaden på pakken som inneholder elementene i tabellen vil variere ettersom hvor mange elementer som er i tabellen. Mellom hvert element i en rad i tabellen blir det nemlig overført 2 byte. I tillegg blir det overført 1 byte i overhead for hver ny rad.

Hvis det overføres så mye data at det ikke blir plass i en pakke blir det imidlertid mer overhead (54 byte) for hver nye pakke. Maksimumsstørrelsen på pakkene var her 1514 byte på ethernetnivå, mens vindusstørrelsen var på 32768 byte. I tillegg til disse størrelsene kunne klienten sette bufferstørrelsen selv. Den sier noe om hvor mye data klienten kan ta i mot før den prosesserer dataene. Hvis dataene som skal overføres overstiger bufferstørrelsen blir det overført flere pakker med dataelementer av samme type som den som er vist i Figur 5.3.

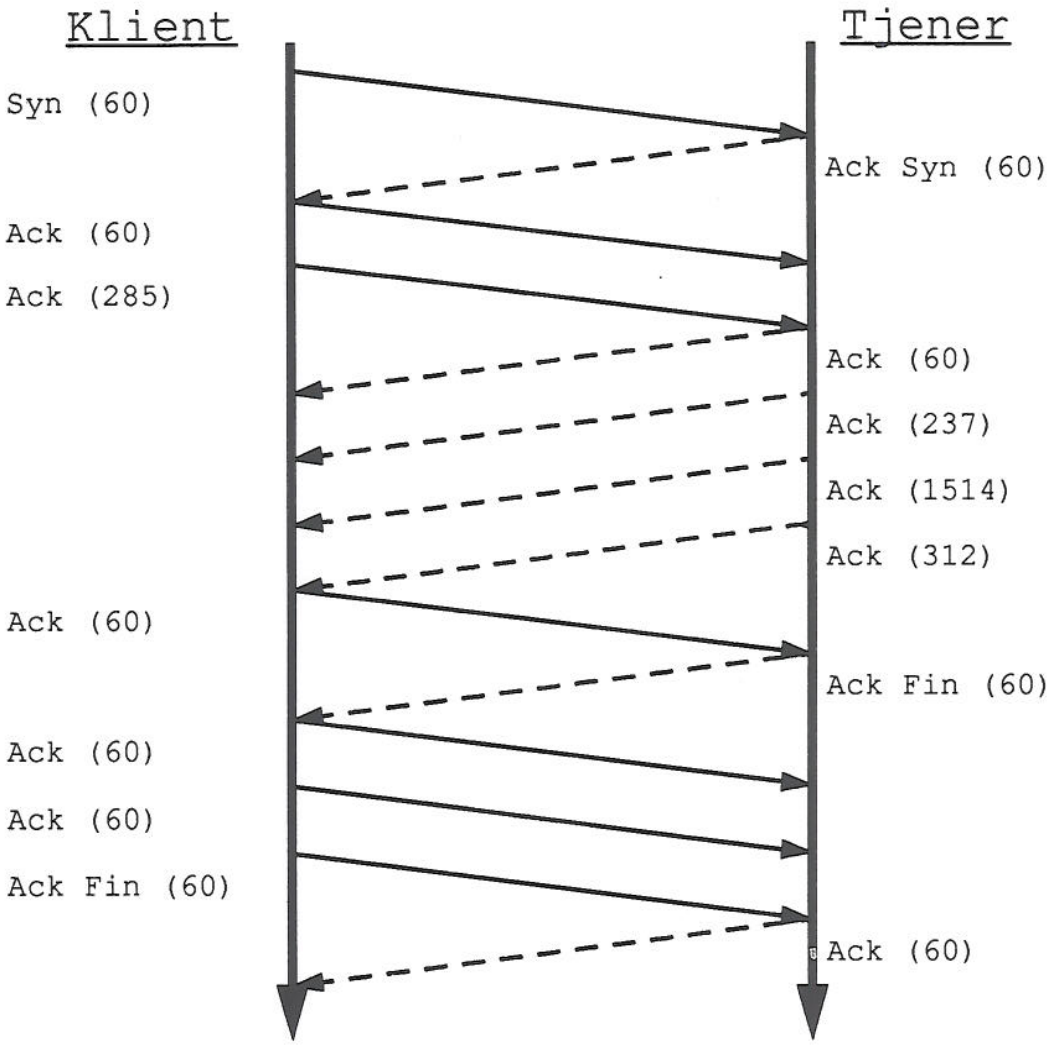
Pakkestørrelsene vi har fått i vårt eksempel kan være avhengig av applikasjonen vi brukte. Vi brukte applikasjonen SQL*Plus, og det kan hende vi ville fått litt andre tall ved bruk av andre applikasjoner. Vi antar likevel at trafikken i grove trekk vil være den samme, mens opp- og nedkoblingen ville vært litt annerledes.



Figur 5.3 Oversikt over pakkene som blir sendt over nettet ved et kall på en database som ligger på en annen maskin enn den vi sitter på. Eksemplet viser en forespørsel etter data lagret i en tabell. Skravert område angir informasjon, mens hvitt område angir overhead. Størrelsen på pakkene er angitt på ethernetnivå.

5.3 Web-nettverk, FTP og TELNET

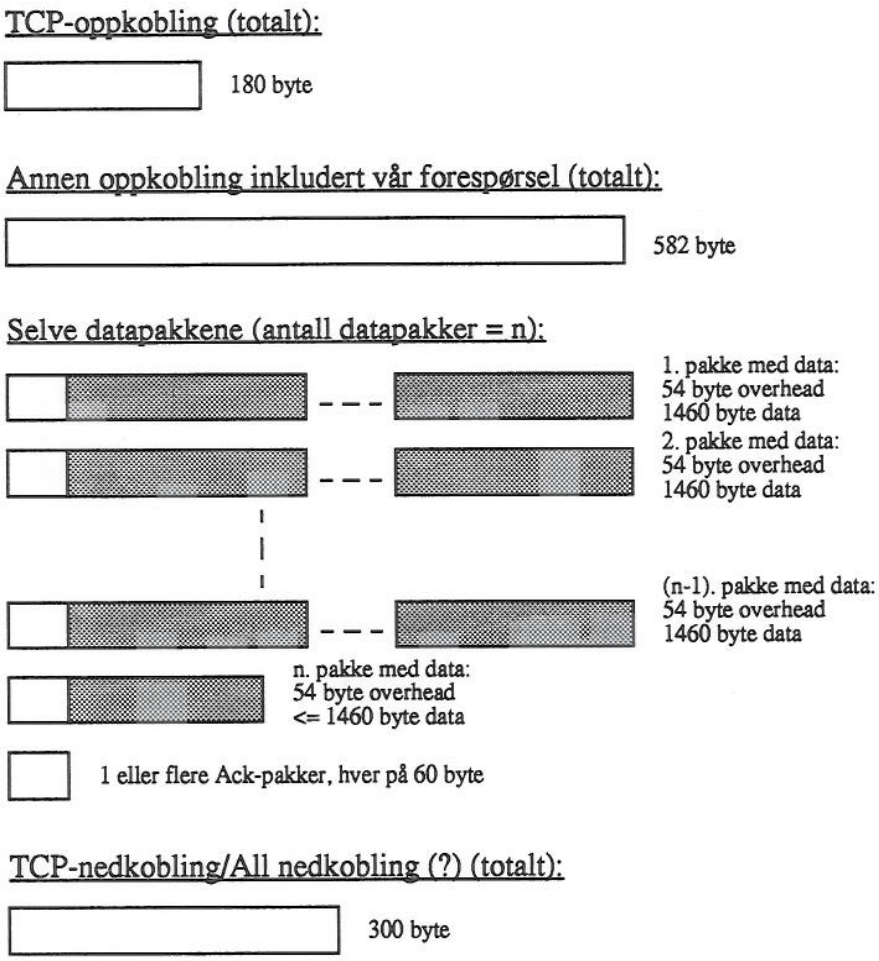
Vi undersøkte også trafikken på nettet som ble generert når vi hentet en fil ved hjelp av World Wide Web eller HTTP-protokollen. Vi satt da på klientmaskinen og hentet en fil fra tjenermaskinen. Filens størrelse var på 1718 byte. Figur 5.4 viser trafikken som gikk på nettet.



Figur 5.4 Eksempel på hva som blir sendt mellom to maskiner når en fil skal hentes, og HTTP-protokollen benyttes, og en TCP/IP-forbindelse blir satt opp. Det er også angitt hva slags type pakke som blir sendt, og hvor stor denne er på ethernetnivå.

Vi er her litt usikre på hva som overføres i de ulike pakkene, men etter å ha overført en del filer og undersøkt den respektive trafikken har vi fått en viss oversikt over hva som er data og hva som er overhead. Dette er vist i Figur 5.5. Størrelsen på datapakkene og vindusstørrelsen ble her satt til henholdsvis 1514 byte og 8192 byte. Dette kan variere fra nett til nett, og vil innvirke på bl a hvor mange "Ack"-pakker som blir sendt når dataene i filen overføres.

Ved bruk av TELNET blir det mye trafikk når brukeren sitter på sin maskin og skriver kommandoer til en annen maskin. For hver bokstav brukeren skriver blir det sendt 2 eller 3 pakker over nettet, avhengig av hvor fort han skriver. Disse pakkene er ikke store (60 byte på ethernetnivå), men det blir allikevel generert endel trafikk tilsammen på denne



Figur 5.5 Oversikt over hvordan overheaden er fordelt ved overføring av en fil med HTTP-protokollen. Skravert område angir informasjon, mens hvitt område angir overhead. Størrelsen på pakkene er angitt på ethernetnivå.

måten. Vi logget oss inn på maskinen "tulitt" fra en annen maskin, gjorde en enkel kommando, og logget oss ut igjen. Trafikken som ble generert er vist i Tabell 5.1.

Handling utført	Total trafikk [byte]
Logger inn ved å skrive "telnet tulitt"	984
Oppgir brukernavn og passord	2321
Skriver "ls" og får listet opp 3 directories	822
Logger ut ved å skrive "logout"	1560

Tabell 5.1 Oversikt over total trafikk som gikk på ethernetnivå ved bruk av protokollen TELNET.

Å logge seg inn på en annen maskin ved bruk av FTP er derimot ikke så belastende for nettverket. Her blir ikke alle bokstavene brukeren skriver sendt over nettet en og en. In-

genting blir sendt før brukeren har trykket <enter> etter en kommando. Vi gikk inn på maskinen "tulitt" fra en annen maskin og hentet en fil. Deretter logget vi oss ut. Trafikken som ble generert på nettet er vist i Tabell 5.2..

Handling utført	Total trafikk [byte]
Logger inn ved å skive "ftp tulitt"	336
Oppgir brukernavn og passord	477
Bytter directory 2 ganger	418
Henter ut en fil med størrelse 24576 byte	26921
Logger ut med kommandoen "bye"	368

Tabell 5.2 Oversikt over total trafikk som gikk på ethernetnivå ved bruk av protokollen FTP.

5.4 TCP/IP kontra X.25

Våre tester er utført på Unix-maskiner med TCP/IP protokoll mens datatrafikk i Taktisk Digitalt Kommunikasjonssystem (TADKOM) vil bruke X.25. TMHS vil bruke TP0 sammen med X.25. Den samlede protokollstakken opp til og med transportlaget vil måtte tilby en del tjenester for de øvre lag uavhengig av om det brukes ISO-protokoller eller andre protokoller. Fordelingen av funksjonalitet mellom de forskjellig lagene vil derimot bli ganske forskjellig.

5.4.1 Linklag.

Vårt testoppsett med TCP/IP bestod av Unix-maskiner i et helt vanlig lokalnett. Linklaget i et slikt nett tilbyr overføring av diskrete enheter av informasjon mellom stasjoner som er direkte tilknyttet lokalnettet. Disse dataenhetene kalles rammer. Linklaget tilbyr lite funksjonalitet utover levering av disse rammene. Dataoverføringen er beskyttet av en sjekksum for feildeteksjon. I lokalnett kan linklaget deles i Media Access Control (MAC) og Logical Link Control (LLC). I vårt lokalnett regulerer MAC tilgangen til det fysiske mediet vha en protokoll IEEE 802.3 (Ethernet) som benytter en transmisjonsteknikk som kalles Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Over dette sublaget ligger LLC som tar seg av adressering og kontroll av sjekksum. Data legges i rammer hvor Ethernet rammer har 46 - 1500 bytes data og 32 bit sjekksum for feildeteksjon. Dette gjør det mulig å detektere feil i en ramme men det kan også forekomme duplikater. CSMA/CD tilbyr ikke kvittering, slik at en e v t kvittering må komme som en egen pakke eller som et vedheng på en datapakke i riktig retning (piggybacking). Padding brukes fordi rammene ikke kan være under en minste lengde.

Linklaget i X.25 tilbyr en mer pålitelig overføring av rammer. Nettet blir her bygget opp som punkt til punkt forbindelser mellom pakkesvitsjede noder. Linklaget skal tilby en virtuell kanal som er feilfri og uten duplikater og skjule de feil som måtte oppstå under over-

føringen. Link Access Procedure B (LAPB) som er et subset av High-level Data Link Control (HDLC)- protokollen er laget for å gjøre dette. Det er da nødvendig med detektering av feil og retransmisjon hvis det oppstår feil. Linklaget tilbyr også flytkontroll for å hindre oversvømming av mottakeren. Data deles opp i rammer som avsluttes med en flaggsekvens i stedet for at lengden er angitt i et lengdefelt. Dataoverføringen er transparent. I hver ramme er det en 16 bits sjekksekvens. Linklagsprotokollen kontrollerer en flyt av rammer på en punkt til punkt forbindelse. Nettverksadresseringen kontrolleres av nettverks(pakke) laget. Sett fra linklaget er derfor nettverksadressen en del av datafeltet som overføres. Adressefeltet i rammen på linklaget brukes derfor bare for å vise retningen som rammen går. Dette kan en finne ut hvis en har oversikt over hva slags type ramme det er.

Linklaget i de 2 eksemplene er svært forskjellig da X.25 er designet for tilkobling til et nett der svitsjer er koblet i et maskenett som et klassisk telenett. Linklaget i Ethernet er derimot laget for et bussnett der alle hører alle. I det første tilfellet er det nødvendig med protokoller som kan håndtere høye feilrater som følge av støy på linjene. Dette er en naturlig følge av lange og ofte uskjermede linjer som utsettes for elektromagnetisk støy. Lokalnett (Ethernet) er derimot i hvert fall i opprinnelig form laget for å benytte seg av en koaksialkabel med terminering i begge ender og bestemte regler for lengde og avstand mellom tilkoblingene. Dette gir et mye mer kontrollert medium. Linklaget i lokalnettet kan derfor sende data til overliggende lag uten en rigid feilkontroll. Pga den nevnte oppbyggingen er derimot linklaget i X.25 avhengig av kompliserte protokoller for å kunne tilby en pålitelig overføring. Disse protokollene har en så stor grad av innebygget sikkerhet at det er mulig for de overliggende lag å stole på data som kommer gjennom også med hensyn til flytkontroll.

5.4.2 Nettlag.

I forbindelse med TCP brukes Internet Protocol (IP) på nettlaget. IP gir mulighet for adressering av både nettverk (nettverksnummer) og nodenummer. IP har også mulighet for fragmentering og sammensetting. Dette gjør det mulig å sende store IP datagrammer over transittnettverk som har små pakkestørrelser. Nettlaget gjør det mulig å lage uavhengige lokalnett og sende data fra en node i et lokalnett til en node i et annet lokalnett (router/ gateway-funksjon). Nett kan også forbindes v h a en telekommunikasjonslink. IP mapper internet adresser fra overliggende protokoller og over til datalinkadresser for det aktuelle nettverket. IP har et antall parametre for tjenestekvalitet bla valg mellom datagram eller forbindelsesorientert overføring.

X.25 Packet Layer Protocol (PLP) er X.25 familiens protokoll på lag 3, nettlaget som også kalles pakkelaget. PLP er en protokoll for forbindelsesorientert overføring. X.25 definerer grensesnittet mellom terminal, Data Terminal Equipment (DTE), og nettet, Data Circuit-terminating Equipment (DCE). Ruting håndteres derfor av andre protokoller som er interne i nettet. Det snakkes her om 2 former for forbindelse, virtuell krets og permanent virtuell krets. Den første er som et vanlig telefonoppkall og den siste formen tilsvarer en fast linje. Maks pakkelengde som kan overføres kan bestemmes ved forhandlinger ved oppsetting av forbindelsen. Det er mulig å sende avbruddspakker. Disse er uten sekvensnummer og sendes så fort de ankommer, foran e v t pakker som er ordnet i en kø av pakker med sekvensnummer. X.25 har etterhvert fått en datagrammulighet som kalles "fast se-

lect". Denne sendingen skjer ved at CALL REQUEST pakken er utvidet slik at den kan inneholde 128 byte med brukerdata. CLEAR REQUEST som sendes som svar er også utvidet slik at den kan inneholde 128 byte data. Sett fra nettverket blir dette som et forsøk på etablering av en virtuell krets. Dette gir en reduksjon i overhead i forhold til vanlig oppkobling. Det brukes flytkontroll med glidende vindu på nettlaget mens en satser på å bruke feilkontrollen fra linklaget (HDLC) dvs ingen egen feilkontroll på nettlaget. Hvis feilsituasjoner skulle oppstå er det 2 mekanismer på pakkelaget som kan brukes. Den ene er en "reset" prosedyre som starter opp igjen en virtuell kanal. Den andre mekanismen er en "restart" prosedyre som virker på alle kanalene som er i bruk.

Nettlaget i TCP/IP kontra X.25/TP0 har en svært forskjellig funksjonalitet da IP er innrettet mot å sende datagrammer mens X.25 PLP er en protokoll for forbindelsesorientert overføring. Det faktum at X.25 definerer grensesnittet mellom DTE og DCE gjør at en blir avhengig av andre protokoller for funksjoner som ruting. Selv om X.25 har en datagrammulighet er det lett å se at X.25 i utgangspunktet er bedre tilpasset telenett enn lokalnett.

5.4.3 Transportlag.

Transportlaget skal tilby en pålitelig og effektiv ende-til-ende overføring mellom prosesser, ikke bare mellom maskiner. Ved oppsett av virtuelle kretser skal transportlaget tilby tjenester til applikasjonene slik at disse kan utveksle data som om det var en fysisk punkt til punkt forbindelse mellom dem. Nettlagsprotokollen IP er laget for å støtte virtuell krets, Transmission Control Protocol (TCP) og datagram, User Datagram Protocol (UDP) på transportlaget. TCP er laget for å bygge på et relativt upålitelig nettlag sammenlignet med det som tilbys i X.25. Dette gjør at TCP blir en kompleks protokoll som må kunne håndtere deteksjon av tapte pakker, automatisk retransmisjon og håndtering av forsinkede duplikatpakker. TCP er designet for å operere over IP. Det settes opp en forbindelse mellom 2 "sockets". En socket er en sammenstilling av IP-adresse (fra pakkehodet til IP) og applikasjonens portnummer (fra pakkehodet til TCP). En TCP forbindelse som er åpnet kan bli stående åpen lenge fordi TCP har i utgangspunktet ingen mekanisme for å detektere tap av forbindelse.

Data fra overliggende protokoller behandles som en kontinuerlig "stream". TCP deler opp denne kontinuerlige datastrømmen i diskrete enheter som kalles segmenter (Opptil 65 K) Vanligvis deler TCP data opp i segmenter som er tilpasset nettverket som maskinen er tilkoblet. Ved retransmittering kan TCP pakke om data ved f eks å kombinere 2 små segmenter i et større for å redusere andelen av data som er pakkehoder. TCP har en flytkontroll basert på et vindu med lovlige sekvensnummer som også skal hindre metning internt i TCP. TCP har en handshakeprosedyre med klokkebaserte sekvensnummer slik at det skjer en synkronisering ved opprettelse av en forbindelse. TCP har 2 former for nedkobling "Graceful close" og "Abort". Ved "Abort " kan data gå tapt. TCP har mekanismer for positiv kvittering for å gjenopprette forbindelsen ved brudd på lavere lag, og en mekanisme for multipleksing. En øvrelagsprotokoll kan støtte forbindelse med flere fjernprotokoller og 2 prosesser kan ha flere forbindelser med hverandre. Internet protokollene TCP/IP og UDP/IP kan ikke lagdeles på helt samme måte som ISO's protokoller da f eks TCP er avhengig av IP for adressering.

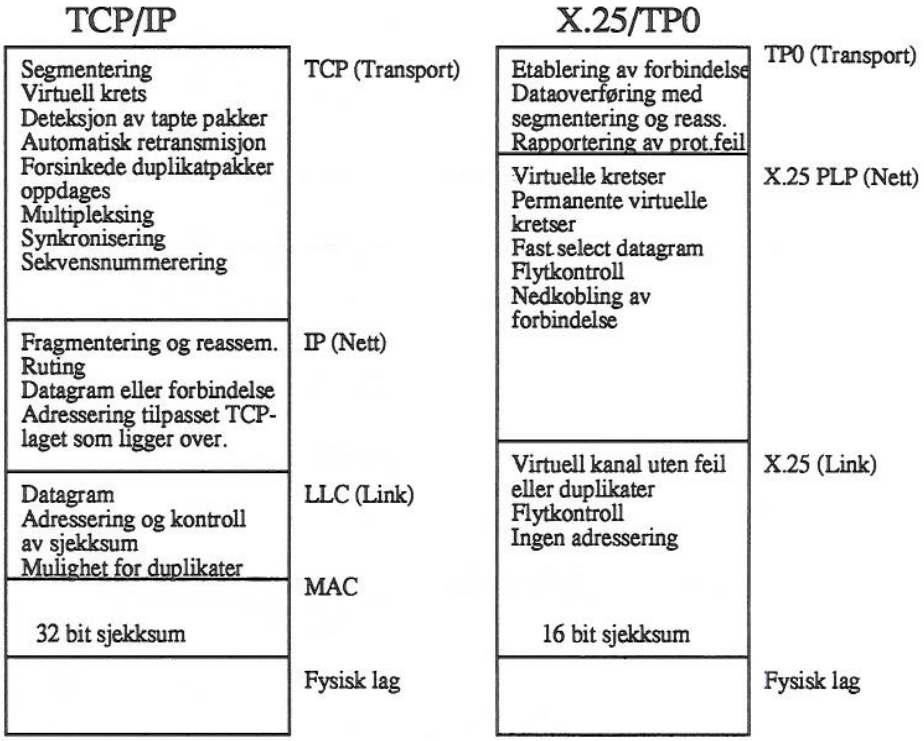
TP0 tilhører den laveste av ISO's 5 klasser av transportprotokoller. Dette er en enkel protokoll som forutsetter nettype A d v s et nett med at akseptabelt nivå av udetekterte og detekterte feil. Ved feil f eks N_RESET etablerer TP0 en ny forbindelse. Det må derfor være liten frekvens av N_RESET's fra nettprotokollen. Tjenestetilbudet er begrenset, og det tilbys bare funksjoner for etablering av forbindelse ved forhandling, dataoverføring med segmentering og rapportering av protokollfeil. Flytkontroll og nedkobling baseres på nettlaget. Alle ISO's transportklasser kobler transportforbindelser til nettverksforbindelser. Dette kan være nettverksforbindelser som allerede er opprettet. Ved feil sørger transportlaget for resynkronisering og reassignment. Alle transportklassene har segmentering og reassemblering for å tilpasse lengden av datafeltet til den lengden som det underliggende laget kan håndtere. Ved oppdeling av en pakke er det nødvendig å utstyre hver del med et hode med protokollinformasjon. Ved oppretting av en transportforbindelse er det mulig å forhandle om kommunikasjonsparametre. Enhver transportentitet kan nekte et forsøk på å opprette en transportforbindelse.

Transportprotokollene bærer tydelig preg av å være utformet for å bygge på forskjellige underliggende lag. TP0 er en typisk minimumsprotokoll der flytkontroll og nedkobling av forbindelser baseres på lavere lags protokoller. Hele protokollen baseres på at lagene under transportlaget tilbyr en stabil forbindelse.

TCP/IP er derimot laget for å operere over mange forskjellige slags nett så her må transportlaget (TCP) inneholde en høy grad av feilhåndtering og dermed mye mer av den totale funksjonaliteten. Den har bla egen håndtering av nedkobling og flytkontroll vha vinduer. ISO har 5 forskjellige klasser (TP0 - TP4) av transportprotokoller der TP0 tilhører den laveste klassen med minst funksjonalitet. Internet protokollene TCP/IP og UDP/IP gir ikke samme mulighet for forskjellige klasser. ISO's protokoll TP4 har en funksjonalitet på linje med TCP/IP. TP0 er valgt sammen med X.25 fordi X.25 gir en pålitelig forbindelse slik at en kan klare seg med en enklest mulig transportprotokoll. Forbindelsen X.25/TP0 benytter flytkontroll i X.25 mens TCP har en egen ende til ende flytkontroll.Normal nedkobling i TCP er hensynsfull "graceful close" men det er også mulig med en aborterende nedkobling. I X.25/TP0 er det en direkte sammenheng mellom levetiden på transportforbindelsen og levetiden på nettverksforbindelsen. Nedkoblingen skjer ved at nettlaget kobler ned. Ved bruk av TCP er det bare mulig med en forbindelse mellom et par Transport Service Access Point (TSAP). Tilsvarende gjelder for ISO-protokollene bortsett fra TP4 som har mulighet for splitting og rekombinering slik at 2 eller flere nettverksforbindelser kan brukes for å støtte en transportforbindelse.

Som vi ser av denne sammenligningen er funksjonaliteten på de forskjellige lagene svært forskjellig, men den totale funksjonaliteten når vi ser lagene under ett er ikke så forskjellig. Vi har derfor valgt å bruke resultater fra lokalnettet som en pekepinn på hva som vil skje i et nett basert på X.25 og TP0. Vi har brukt dette for b l a å finne overhead ved forskjellige typer overføring. Generelt har TADKOM mindre overhead enn TCP/IP. Ved overføring over TADKOM vil derfor overheaden bli mindre enn det som er vist i eksemplene i dette notatet. For vårt formål har det ikke vært nødvendig å beregne denne overheaden, selv om overheaden kan beregnes nøyaktig for situasjoner uten feil.

Forsøkene ble utført ved å analysere trafikk på linklaget mellom 2 maskiner. Data som ble lagt på i de forskjellige lagene kom mer eller mindre klart fram ved analysen. Egentlig var



Figur 5.6 Fordeling av funksjonalitet i TCP/IP og X.25/TP0.

vi interessert i å finne mengden av data som kommer ned fra applikasjonene og ned til transportlaget, for det er disse dataene som angir belastningen på sambandssystemet. Så lenge det ikke oppstår problemer på lokalnettet er det forholdsvis greit å finne hva som kommer ned til transportlaget ut i fra det som går på link/fysisk lag. I vårt tilfelle var det oversiktlige forhold fordi kommunikasjonen forgikk på en kanal uten feil eller ruting. Ved f eks feil på kanalen eller ruting hadde det vært mye vanskeligere å finne ut hvor mye ekstra trafikk som ble generert av de lavere lag som følge av feilen. Retransmisjon og liknende kan bli betydelig hvis det oppstår feil på kanalen.

6 ANALYSE AV PROTOKOLLENE

HTTP-protokollen som brukes i Web-nettverk og protokollene FTP og TELNET er alle protokoller for å overføre data fra et sted til et annet annet over en datalinje som hele tiden er tilgjengelig. De inneholder ikke funksjonalitet for evt lagring eller mellomlagring av data. De passer derfor best for sentraliserte løsninger der en kan gå ut i fra at datalinjene er tilgjengelige når en trenger informasjon. På grunn av større redundans hvis noen kommandoplasser skulle bli slått ut, og ønske om mindre tidskritisk avhengighet av sambandet, er vi i utgangspunktet interessert i at informasjonen skal ligge distribuert. Vi har derfor sett litt nærmere på hvor mye trafikk som vil bli generert ved oppdatering av situasjonsbildet ved bruk av SQL*Net og X.400.

6.1 Informasjonsrepresentasjon

Vi har konsentrert oss om oppdatering av situasjonsbildet og tatt utgangspunkt i informasjonsmengdene oppgitt i (12). Her er informasjonen delt inn i hovedemner som er oppgitt som en vektor i tre dimensjoner; antall forhold det ønskes informasjon om, antall parametre som skal bestemmes, og antall bit informasjon. Det er rimelig å tro at det kan bli problemer ved nedgradering når informasjonen er bitkodet på denne måten. Vi har derfor valgt å kode informasjonen med siffer (fra titallssystemet), fordi det er en naturlig og mye brukt representasjon. Vi har derfor antatt at parametrene bare kan anta verdier fra et endelig diskret utfallsrom. Tanken er da at det skal finnes tabeller lokalt på hver kommandoplass som konverterer sifrene over til beskrivende ord. Om det f eks gjelder en værmelding, finnes det tabeller som sier at hvis vindretning er 4, og vindstyrke er 6 betyr det nordøstlig stiv kuling. Å bruke siffer i steden for ord vil begrense størrelsen på informasjonen som må overføres.

I Standard Query Language (SQL) blir tekststrenger med variabel lengde som tidligere nevnt overført i ASCII-format hvor hvert tegn tilsvarer 1 byte. I tillegg blir det for hver tekststreng overført 1 byte som angir lengden på tekststrengen. Når det gjelder tall er representasjonen litt mer komplisert. Selve tallet blir beskrevet med 1 byte for hvert 2. siffer. Videre blir det lagt til 1 byte som angir eksponenten, 1 byte angir lengden på tallet, og hvis tallet er negativt blir det lagt til 1 byte ekstra for å angi dette. Siden vi ikke har brukt negative tall i vår koding, bortsett fra ved angivelsen av temperaturen i værmeldingene, har tall bestående av 1 og 2 siffer blitt representert med 3 byte, tall bestående av 3 og 4 siffer har blitt representert med 4 byte o s v. Både posisjoner (10 tegn) og tidspunkter (7 tegn) har vi valgt å representere som tekststrenger med variabel lengde slik at disse har fått en lengde på henholdsvis 11 byte og 8 byte iberegnet 1 byte som beskriver lengden på strengen.

Vi gikk igjennom noen av informasjonspakkene (12), og beregnet da følgende størrelser på tre pakker:

Emne	Beskrivelse	Mengde (byte)
Egne styrker	Posisjon, tap forsynings- og slitasjestatus hovedmateriell og personell, beredskap og aktivitet for underlagte og sideordnede avdelinger, minimum to hakk ned. Tilsvarer 30 avdelinger. Forsterkningsmuligheter.	3236
Motstanderen	Situasjon, luftsituasjon, handlemåte og forsterkningsmuligheter. Nivå, våpengren, posisjon, utstyr, slitningsgrad og virksomhet for 40 avdelinger.	2530
Været	Værmelding for en dag; temperatur, nedbør,vindforhold, tåke/skydekke og føre.	28

Tabell 6.1 Størrelse på tre typiske informasjonspakker.

Hvis vi lar y være størrelsen oppgitt i antall byte på en informasjonspakke bestående av heltall, og x være størrelsen på pakken oppgitt i antall bit i notatet (12) kan vi bruke følgende (konservative) omregningsformel eller tommelfingerregel:

y [byte]$= 0.7x$ [bit]

Ved å bruke denne formelen kan vi på en enkel måte konvertere informasjonsmengden på bitkodet informasjon (12) over til tallkodet informasjon hvor mengden er angitt i byte.

6.2 Generelle uttrykk for total trafikk

Vi har nå tatt for oss X.400 -protokollen og SQL*Plus og funnet generelle uttrykk for den totale trafikken som blir generert med disse protokollene. SQL*Plus er en Oracle-applikasjon som benytter SQL*Net. Størrelsene som vi har kommet frem til kan bare sees på som en indikasjon på den virkelige trafikken fordi vi ved våre målinger har brukt TCP/IP-protokollen mens vi på taktisk nivå i hæren skal bruke X.25. Vi antar likevel at den relative forskjellen blir omtrent den samme.

Vi har sett at disse protokollene fungerer forskjellig ved opp- og nedkobling. Ved bruk av databasekommunikasjon er det en tung oppkobling. Men når dette først er gjort, er det ingenting i veien for at forbindelsen kan stå oppe til neste transaksjon skal utføres. Med X.400 derimot har vi en lettere opp- og nedkobling, men så må den til gjengjeld utføres hver gang en melding skal formidles.

Vi har benevnt antall elementer som skal oppdateres/hentes fra en tabell med l og antall kolonner i tabellen med k. Videre har vi antatt at en oppdateringskommando i SQL har en størrelse på ca $50 + 12l$ byte. Disse tallene er delvis tatt fra erfaring og delvis fra at vi

antar at kolonnenavn, oppdatert verdi og et likhetstegn tilsvarer i gjennomsnitt 12 tegn. Vi vet fra før at kommandoen blir overført i ASCII-format.

Når det gjelder å oppdatere en database med X.400, antar vi at dette foregår ved at man sender en SQL-oppdateringskommando som en melding, og at trafikken som da går over nettet kan beskrives ved (størrelser hentet fra Figur 5.2):

$$T_{oppd}^{X,400} = 1725 + kommando$$
$$= 1775 + 12l$$

Her har vi sett bort fra selve TCP/IP-opp-og nedkobling, og antatt at kommandoen har en slik størrelse at to datapakker trengs. Trafikken er angitt på ethernetnivå og oppgitt i byte. Forespørsler kan vi anta at genererer minst dobbelt så mye trafikk, fordi man da må sende to meldinger, en forespørsel og et svar med data. Det vil da bli en overhead på over 3.5 kbyte.

Vi har videre antatt at et kolonnenavn i gjennomsnitt består av 8 bokstaver, slik at når de blir overført i ASCII-format blir dette 8 byte. I tillegg blir de overført med 1 byte mellom hvert kolonnenavn slik at vi får i gjennomsnitt 9 byte for hvert kolonnenavn som blir overført. Elementene som blir overført antar vi at i gjennomsnitt blir beskrevet med 4 byte, mens elementene i en tabell blir overført med 2 byte mellom hvert element. Dette gir oss i gjennomsnitt 6 byte for hvert element som blir overført i tillegg til annen overhead.

Hvis vi ser på en oppdatering med bruk av databasekommunikasjon v hj a SQL*Plus gir dette oss følgende totale trafikk på ethernetnivå:

$$T_{oppd}^{SQL} = 476 + kommando$$
$$= 526 + 12l$$

Vi har her sett bort fra all opp- og nedkobling fordi det antas at det ikke er nødvendig å koble opp og ned mellom hvert kall. Linkene i TADKOM kan ha opptil 250 oppkoblede forbindelser samtidig, så det burde være gjennomførbart om alle de 25 største kommandoplassene hadde en kontinuerlig oppkoblet forbindelse seg i mellom.

Denne oppdateringstrafikken kan reduseres til $526 + 3l$ hvis man ordner det slik at hver oppdatering består i å legge til en ny rad (med verdier i hver kolonne) i en tabell.

Hvis vi derimot ser på en forespørsel slik som i eksemplet vist i Figur 5.3, gir dette oss følgende trafikk på ethernetnivå når vi antar at en standard forespørsel består av 60 tegn:

$$T_{foresp}^{SQL} = 927 + kommando + kolonner + data$$
$$= 927 + 60 + 9k + 6l$$
$$= 987 + 9k + 6l$$

Ut fra dette ser en at det er mulig å redusere trafikken betraktelig ved å ikke benytte seg av X.400 ved databaseoppdateringer og forespørsler. Fordelen med X.400 er imidlertid at man ikke mister oppdateringen eller forespørselen hvis en link plutselig skulle gå ned. Man har derfor større kontroll med bruk av X.400, så lenge ikke trafikken blir så stor at større forsinkelser oppstår.

7 MULIGHETSSTUDIER

Ved hjelp av uttrykkene for trafikkpåtrykk og informasjonsrepresentasjon går det an å danne seg et bilde av hvor ofte det er mulig å oppdatere situasjonsbildet. Å ha et oppdatert situasjonsbilde er en viktig forutsetning for at divisjonen til enhver tid skal kunne ta de riktige beslutningene.

Videre har vi sett på gjennomførbarheten av å kunne ha forskjellige konferanseapplikasjoner. Det kan være til stor hjelp for offiserer om de slipper å reise til hverandre ved planutarbeidelse og ved andre diskusjoner.

7.1 Distribusjon av situasjonsbildet

I følge (12) skal fullstendig oppdatering av egne styrker foretas en gang i døgnet. Dette er den største pakken som består av 942 parametre. Med en SQL-oppdatering sendt som en X.400-melding (se avsnitt 6.2) vil denne få en størrelse på ca. 15 kbyte, iberegnet litt ekstra overhead fordi datamengden er så stor at vi vil få flere datapakker. Dette tilsvarer 120 kbit, og med linkkapasitet på 38.4 kbit/s vil dette ta 3.13 sekunder å klokke over en link i TADKOM. Reell forsinkelse for en melding vil bli betydelig større fordi vi har prosesseringstid i TADKOM-svitsjen og meldingssvitsjen, samt at meldingen som oftest må over flere linker. Allikevel viser dette at en slik oppdatering er realiserbart en gang i døgnet som forutsatt. De oppdateringene som i følge (12) skal gjøres med intervaller på 10-15 minutter er betydelig mindre. De består av 8 parametre, noe som tilsvarer å måtte overføre omtrent 1.9 kbyte, iberegnet X.400 overhead. Her er ikke X.25-oppkoblingen tatt med for noen av oppdateringene, men selv med et lite tillegg i overhead på grunn av dette ser det ut til at kapasiteten i TADKOM er stor nok til å håndtere dette.

Et aspekt som bør belyses er hvordan informasjonen skal distribueres rundt i divisjonen. TADKOM-nettverket kan ofte være langstrakt, og det kan godt hende at nettverket er delt i to deler som bare er forbundet med to linker. Det medfører at all trafikk som skal fra en node i den ene delen til en node som ligger i den andre delen av nettverket må gå over disse to linkene. Hvis alle skal oppdatere alle er det lett å tenke seg at disse to linkene vil bli flaskehalser i systemet. Distribusjonsmekanismer som hindrer dette vil derfor ha stor nytteverdi.

Likeså kan det være mulig å redusere den totale trafikken ved at man samler informasjonen fra "eierne" før man distribuerer ut et mer samlet situasjonsbilde, istedenfor at alle sender sitt bidrag til alle. Dette samt forskjellige distribusjonsmekanismer i nettverket har vi sett litt nærmere på, men først blir problemet knyttet til eierskap av data belyst.

7.1.1 Eierskap på data

Som nevnt i avsnitt 3.3, bør man ha eierskap på data for å sikre mest mulig konsistente data, selv om man prioriterer tilgjengelighet foran konsistens. Spørsmålet blir da hvem som skal ha eierskap på hva. En løsning kunne være å la en enhet eie alt, men dette blir ingen gunstig løsning med hensyn på sårbarhet og utjevning av sambandsbelastning. En

annen løsning kunne være å la enheter over bataljonsnivå ha eierskap på de dataene som angår dem selv og deres underordnede enheter, f eks posisjon og slitasjestatus, mens Div/ ES fikk eierskap på bearbeidede etterretninger og observasjoner.

Ved å definere eierskap til data vil vi få problemer med å få oppdatert data som eies av enheter som er nede, eller ikke har samband. En løsning på dette problemet kunne være å lage en prioritert rekkefølge på eierskap på data, slik at hvis en enhet skulle gå ned eller miste sambandet, er det en annen enhet som overtar dette eierskapet denne enheten hadde. Rekkefølgen bør da bestemmes utfra kompetansen på hver kommandoplass på det området som informasjonen omhandler. Det ville også være en fordel om rekkefølgen hadde noe med geografisk nærhet å gjøre, men dette vil som regel være oppfylt i praksis.

En annen løsning kunne også være å definere erstatningsdata som oppdateres lokalt eller av en enhet som en fortsatt har kontakt med. Et eksempel på dette kan være en lokal registrering av observasjoner som pågår så lenge den vanlige kilden til slike data ikke er tilgjengelig. Etter at sambandet er oppe igjen må de versjonene som har oppstått koordineres. Fordelen med denne metoden framfor prioritert rekkefølge på eierskap er at databasesystemet slipper å ta seg av bytte av eierskap. Applikasjonene må ta seg av dette etter at sambandet er kommet opp igjen. For observasjoner kan en etter rapportering til ES få tilbake et oppdatert og offisielt situasjonsbilde. Med den første metoden kan det være et problem å oppnå konsistente data etter at feilen har blitt reparert, hvis det har vært to eller flere grupper som ikke har hatt samband seg i mellom. Det har da oppstått to eller flere versjoner av dataene som må koordineres. Med den siste metoden vil det derimot bli lettere å oppnå konsistente data når eieren kommer opp eller får samband igjen.

7.1.2 Reduksjoner av samlet datamengde

Som nevnt er en mulighet at alle avdelingene distribuerer informasjon som de genererer eller har ansvaret for til alle de andre avdelingene. En annen mulighet er at man definerer en sentralisert enhet som har ansvaret for å motta informasjon fra de andre avdelingene og deretter distribuerer ut et samlet oppdatert situasjonsbilde. Dette er på en måte to ytterpunkter når det gjelder alternative måter å distribuere ut et oppdatert situasjonsbilde på, men vi har likevel sett litt nærmere på disse for å se om det blir noen forskjell på sambandsbelastningen.

For å forenkle situasjonen har vi antatt at for hvert oppdaterte situasjonsbilde som blir distribuert ut fra den sentraliserte enheten har hver av avdelingene sendt inn sin oppdaterte informasjon en gang. Med en sentralisert enhet blir det derfor sendt $(n - 1)$ små oppdateringer, og $(n - 1)$ store oppdateringer. n angir her antall avdelinger. Hvis alle skal sende til alle blir det derimot $n(n - 1)$ små oppdateringer. Med f eks 30 avdelinger blir dette 870 små oppdateringer som skal formidles, mens man med en sentralisert enhet bare trenger å formidle 29 små og 29 store oppdateringer. Enkel regning viser at hvis de store oppdateringene er mindre enn $(n - 1)$ ganger så store som de små oppdateringene vil dette føre til større total trafikkmengde. Her spiller derfor størrelsen på overheaden inn.

Hvis vi ser på informasjonen om egne styrker (12), så inneholder denne 942 parametre for 30 avdelinger. Grovt sett kan vi derfor si at hver og en avdeling har ansvaret for å oppda-

tere 31 parametre. Hvis vi antar at de bruker databaseoppdateringer vil dette si at 31 elementer skal overføres til den sentraliserte enheten, og 942 elementer skal sendes fra den sentraliserte enheten og tilbake til de andre avdelingene. Hvis vi bruker uttrykkene for den totale trafikk ved SQL-oppdateringer ved bruk av SQL*Net gjengitt i avsnitt 6.2, gir dette en trafikk på henholdsvis 898 byte og 11830 byte til og fra hver kommandoplass. Den siste oppdateringen er antakeligvis så stor at den vil bli delt opp på ethernetnivå, slik at hver av disse nye pakkene vil få en overhead på 54 byte. Vi kan anta at det vil bli omtrent 8 slike pakker, slik at den største oppdateringen vil få en størrelse på ca. 12300 byte. En fullstendig oppdatering av informasjonen knyttet til egne styrker vil derfor gi en total trafikk på 781 kbyte om alle skal sende oppdateringer til alle, mens man med en sentralisert enhet bare trenger å overføre 383 kbyte. Det vil si at hvis våre antakelser holder i praksis vil distribusjon ved hjelp av en sentralisert enhet medføre at den totale trafikken halveres ved oppdatering av egne styrker. Hvis vi hadde valgt å bruke X.400 istedenfor SQL*Net, ville reduksjonen blitt enda større fordi overheaden her er større.

En faktor som det må tas hensyn til er at det ved en slik løsning blir liten redundans i systemet. Man trenger prosedyrer for hva som skal skje hvis den sentraliserte enheten ikke lenger er operativ slik at man fremdeles får distribuert informasjonen rundt til avdelingene, og samtidig har kontroll med konsistensen til situasjonsbildet.

7.1.3 Distribusjonsmekanismer i nettverket

For å illustrere hvilken betydning ulike distribusjonsmekanismer har på trafikken i nettverket har vi tatt for oss et konkret eksempel. Vi har sett på hvor mange hopp som er nødvendig for å sende en melding fra en avdeling til alle de andre. Som scenario ble manøverfasen av kartspillet brukt, først med divisjonens avdelinger (25 stk) og så med noen underliggende bataljoner slik at det ble 31 avdelinger. Det ble sett på 5 ulike alternativer :

1. Avsender sender ut en melding til hver enkelt av de andre avdelingene.

2. Meldingen fordeles etter en organisasjonsmessig ruting med distribusjonslister.

3. Optimal ruting vha TADKOM's ruter (fullstendig multicast).

4. Deling i 2 regioner.

5. Deling i 4 regioner.

Først ble meldingene rutet slik at det ble færrest mulig hopp for hver enkelt melding. Dette førte til ekstra stor belastning på enkelte linker. Ved neste forsøk ble meldingene fordelt på flere linker. Dette førte til at de mest belastede linkene fikk mindre påtrykk, men antall hopp i nettet økte. De aller fleste knutepunktene har minst 2 linker mot resten av nettet.

Ved tredje forsøk ble det tatt med noen underliggende avdelinger som ikke hadde direkte tilknytning til noe TADKOM knutepunkt slik at det ble 31 avdelinger. Disse avdelingene

var tilknyttet via MRR og vil stå for sin del av trafikken. Det var nødvendig å ta med disse avdelingene for at organisasjonsmessig ruting skulle gi noen mening.

Ved alternativ 4 ble det ikke tatt hensyn til organisasjon, men geografisk plassering. Nettet ble delt opp i 2 regioner som hver hadde en svitsj med distribusjonsliste for den aktuelle regionen. Disse distribusjonslistene ble utformet ut i fra rutingmessige hensyn i stedet for organisasjonsmessig. Dette gjorde at det ikke var nødvendig å sende mer enn en kopi over kritiske deler av nettet. Disse 2 svitsjene og forbindelsen mellom dem kan betraktes som et "overordnet distribusjonsnett". Alternativ 5 ble gjennomført på tilsvarende måte som for alternativ 5, men denne gangen ble nettet delt opp i 4 regioner. Svitsjene ble plassert i knutepunkter med mange linker og flest mulig direkte tilknytninger for å få færrest mulig hopp.

	Forsøk 1 (25n)				Forsøk 2 (25n m/fordeling)				Forsøk 3 (31n m/fordeling)			
	Hopp	Snitt	Sa	Maks	Hopp	Snitt	Sa	Maks	Hopp	Snitt	Sa	Maks
Alt 1	269	2.15	3.30	16	273	2.18	2.94	12	413	3.20	3.96	17
Alt 2	238	1.90	2.96	17	227	1.82	2.65	13	334	2.61	3.34	11
Alt 3	48	0.38	0.48	1					55	0.45	0.49	1
Alt 4									153	1.19	1.62	8
Alt 5									95	0.74	1.05	5

Tabell 7.1 Antall hopp ved sending av melding til alle kommandoplasser. Tabellen viser hopp i nettet for forskjellig alternativer. Kolonnen "hopp" viser hvor mange hopp som er nødvendig for å fordele meldingen til alle. "Snitt" er det gjennomsnittlig antall ganger en link brukes hvis en tar med alle linkene som er i nettet. "Sa" er standardavviket på denne verdien. "Maks" er det maksimale antall meldinger som går over en link, d v s belastning på den mest belastede linken.

Resultatet over antall hopp er vist i Tabell 7.1. Vi ser av tallene at det er mye å spare på en optimal ruting som baserer seg på TADKOM's ruter. Det er her gått ut i fra at en bare bruker de linkene som er strengt nødvendig ved at meldingen kopieres og distribueres etter hvert som den propagerer gjennom nettet. Dette krever en mer eller mindre fullstendig oversikt over hvordan de forskjellige knutepunktene er koblet sammen. For å få til dette må en ha en tett kobling mellom TADKOM og meldingssystemet.

Ved optimal ruting blir trafikken på mest belastende link redusert til 1/17 = 0.06 i forhold til om man skulle sendt en pakke til alle. Dette tilsvarer en reduksjon på 94%.Totalt antall hopp blir redusert til 55/413 = 0.13, som er en reduksjon på 87%. Sammenlignet med en ruting som følger tjenestevei får vi redusert trafikken til 1/11 = 0.09 for mest belastende link, som tilsvarer en reduksjon på 91%. Totalt antall hopp reduseres tilsvarende til 55/334 = 0.16 som gir en reduksjon på 84%. Det er lett å se at gevinsten her er større enn ved reduksjon av samlet datamengde som diskutert i avsnittet over.

Alternativ 5 gir i størrelsesorden dobbelt så mange hopp totalt som optimal ruting, men er mye enklere å gjennomføre da systemet kan settes opp manuelt av en som har oversikt over nettet. En praktisk gjennomføring av en slik ruting vil by på enkelte problemer fordi nettet ikke er statisk, men det kreves ikke store forandringer på systemet slik som ved optimal ruting basert på TADKOM's ruter. Alternativ 4 og 5 er gjennomførbare med dagens system, da det i X.400 finnes muligheter for distribusjonslister (9).

Det ble for alternativ 4 og 5 sett på forskjellige plasseringer for svitsjene med distribusjonslister. En kom da fram til at svitsjene bør plasseres i knutepunkter med mange linker for å redusere maksimalbelastningen på enkelte av linkene. En bør velge knutepunkter med mange direkte tilknyninger for å redusere antall hopp totalt. Plassering i kommandoplasser eller midt i regionen ser ikke ut til å gi noen spesiell gevinst, mens plassering nær forbindelse til neste svitsj kan være gunstig.

7.1.4 Tidsforsinkelse ved oppdatering av situasjonsbilde

Det er i et regneeksempel under forsøkt å gi et inntrykk av tidsforbruket ved distribusjon av en melding fra en kommandoplass til alle de andre. Det forutsettes ingen distribusjonslister slik at det må sendes en egen melding til hver eneste kommandoplass. Videre har vi forutsatt at TADKOM har 38.4 kb/s forbindelse, mens tilsvarende tall for en 9.6 kb/s TADKOM-forbindelse er vist i parentes. Det er også her sett på manøverfasen av kartspillet. Det er ikke tatt hensyn til annen trafikk som måtte gå i nettet, bare den forsinkelsen som vil være i svitsjene og den tiden det tar å overføre dataene over linkene med en hastighet på 38.4 kb/s eller 9.6 kb/s.

Ut ifra målinger som er foretatt på lokalnett med X.400 meldinger med SQL-oppdateringer har vi anslått pakkeantall og størrelser. Vi har gått ut i fra at det går 10 pakker på tilsammen 805 byte før selve meldingen. Som eksempel på en melding har vi valgt en for oppdatering av fiendebildet (12). Denne meldingen er på 540 bit informasjon og 40 parametre. Ved andre tilsvarende konverteringer mellom antall parametre og størrelse på SQL-kommando, er det brukt 50+12*(antall parametre) som gir en størrelse på 530 byte. Dette gir en samlet størrelse på selve meldingen på 744 + 530 = 1274 byte fordelt på 4 pakker. Nedkoblingen er satt til 632 byte fordelt på 10 pakker. Dette gir samlet 24 pakker på tilsammen 2711 byte. Det er gått ut i fra en prosesseringstid på 11 ms i svitsjene. For 24 pakker blir samlet prosesseringstid 24 * 11 ms = 264 ms pr svitsj.

Utklokking over en link: (2711*8)/38400 s = 0.565 s (2.259 s)

Tidsforbruk ved et hopp: 0.565 s + 0.264 s = 0.829 s (3.316 s)

Gjennomsnittlig antall hopp er satt til 10 som gir følgende tidsforbruk: 8.3 s (33 s).

Totat for 31 avdelinger : 31*8 s =257 s dvs 4 min 17 s (17 min 10 s)

Her har vi ikke tatt med forsinkelse i selve meldingssvitsjen. Denne forsinkelsen vet vi ikke størrelsen på, men den avhenger av prosesseringskapasiteten, og vil få betydning.

7.1.5 Diskusjon

Vi har sett at det er flere måter å redusere den totale trafikken på ved oppdateringer. Den løsningen som virker mest interessant er likevel den som effektiviserer selve distribusjonen av meldingen, enten ved fullstendig multicast eller ved distribusjonslister. Her får vi mulighet til å redusere trafikken betraktelig på de mest belastede linkene, og det er opplagt disse som danner flaskehalsen i systemet. Å redusere trafikkpåtrykket her vil ha størst betydning for ytelsen til systemet.

7.2 Konferanseapplikasjoner

Et av målene med det nye ledelsessystemet er at det skal være så hurtig at man klarer å komme innenfor fiendens beslutningssyklus. Dette vil være en klar fordel ved manøverkrigføring. Det ville derfor være fordelaktig om offiserene slapp å reise til hverandre for å diskutere f eks fremtidige planer, fiendens forflytninger o s v, eller når de skulle holde briefinger. Dette tar lang tid, samtidig som det er forbundet med en viss risiko å reise mellom kommandoplassene. En måte å unngå dette problemet på ville være å ha en videokonferanse. Dette krever imidlertid så stor sambandskapasitet at det ikke er realiserbart over TADKOM, og ihvertfall ikke over Multi Rolle Radio (MRR).

En annen applikasjon som amerikanerne har stor tro på er at man har en felles markør på et felles kartgrunnlag hvor man kan tegne og vise på kart samtidig som man har taleforbindelse(14). Taleforbindelse eksisterer både i TADKOM og MRR, så spørsmålet er hvor mye trafikk som blir generert av en felles markør. Hvis markøren skal flytte seg kontinuerlig vil dette opplagt generere stor trafikk, men en bedre løsning ville antakelig være at man klikket på diskrete punkter på kartet for å angi posisjoner og bevegelsesretninger, og at disse posisjonene ble sendt over til de andre deltakerne på konferansen. Det er nemlig viktig at det ikke genereres for stor trafikk på sambandet. Markøren vil miste mye av hensikten sin hvis forsinkelsen blir for stor, fordi den da heller vil virke distraherende. Generelt er det også lite gunstig med stor trafikk, da den vil stjele sambandskapasitet fra andre brukere og applikasjoner. Man kunne også tenke seg at man utvidet funksjonen også til å gjelde markering av avdelinger, tegning av piler o s v. Hvor realiserbart dette blir, kommer da an på hvor effektivt man klarer å lage applikasjonen med hensyn på sambandsbelastningen.

Det finnes selvfølgelig muligheter for at man bruker en taleforbindelse til å overføre markørens bevegelser. Da vil man unngå problemet med forsinkelser, men hvis man velger en egen taleforbindelse til å gjøre dette vil man beslaglegge mye sambandsressurser i forhold til datamengden som skal overføres. Det er også mulig å overføre posisjonene på samme talekanal som konferansedaltakerne bruker til å snakke. Ulempen med denne løsningen er at kvaliteten på taleforbindelsen vil bli dårligere. Hver gang man klikker på en ny posisjon vil det da oppstå en liten forstyrrelse på talekanalen. Begge disse mulighetene er realiserbare, men det beste ville være om markørens bevegelser kunne overføres over pakkesvitsjen hvor datatrafikken går. Denne muligheten har derfor blitt undersøkt nærmere.

I UNIX finnes det en applikasjon som heter "chalktalk" som går ut på at to brukere sitter med samme tegnebrett. Applikasjonen fungerer slik at det den ene brukeren tegner hos seg

da også vil vises hos den andre brukeren. Vi undersøkte denne applikasjonen litt nærmere og fant da ut at det ble generert følgende trafikk på ethernetnivå:

strek med lengde på ca 1 cm (frihånd):	28 pakker, 2261 byte totalt
strek med lengde på ca 1 cm (rett):	18 pakker, 1610 byte totalt
tegn skrevet 3 forskjellige steder på tegnebrettet:	18 pakker, 1209 byte totalt

Å skrive tegn forskjellige steder på arket kan sammenlignes med å flytte pekeren med diskrete punkter. Disse testene viste at denne applikasjonen genererte trafikk som ikke er overkommelig med dagens taktiske sambandsløsninger. Med MRR og 1000 bit/s vil det ta over 13 sekunder å få overført en strek til en annen konferansedeltaker, selv om forbindelsen bare går over en link. Med flere konferansedeltakere og med forbindelser som går over flere linker blir det en helt uakseptabel lang ventetid

Å bruke databaseoppdateringer til å flytte pilen er heller ingen bedre løsning, da dette vil generere ca 550 byte for hver oppdatering (se avsnitt 6.2). Det er imidlertid klart at en slik oppdatering av en ny poisisjon på kartet ikke trenger være så stor. Strengt tatt trenger vi bare overføre en pakke med posisjonen til de andre konferansedeltakerne, og få en pakke i retur som sier at pakken er mottatt. Disse pakkene trenger ikke være store, da det er lite informasjon som skal overføres. Det er mulig det blir vanskelig å kjøpe en ferdig applikasjon som gjør dette slik vi ønsker det, så antakelig er dette noe som må utvikles spesielt til dette formålet.

Vi har sett litt nærmere på om det er gjennomførbart å ha en slik konferanse hvis vi utvikler applikasjonen selv. Vi forutsetter da at det som trengs av informsjon kan bli overført i en pakke (d v s max 128 byte som er maksimumspakkelengden i MRR). Med MRR blir ventetidene på pakker som skal sendes trukket i intervallet 25-125 ms. (forutsatt at dette blir sendt på prioritet 2, nest høyeste prioritet). Sannsynlighetstettheten for ventetiden, y, til den som trekker kortest ventetid av n som ønsker å sende blir da:

$$f(y) = \frac{n}{100^n}(125 - y)^{n-1} \qquad \text{for } y \in [25, 125)$$

Med denne sansynlighetstettheten blir forventet ventetid til første pakke:

$$E(y) = \frac{1}{n+1}(125 + 25n)$$

Hvis det er n MRR-radioer som ønsker å sende, benevner vi hvilket nummer du blir i rekkefølgen til de som får sende med m. Forutsetter nå at de som ønsker å sende har uendelig mange pakker, eller tilsvarende at det til enhver tid er n radioer som ønsker å sende i samme maskenett. Vi får da at forventet nummer i rekkefølgen blir:

$$E(m) = \sum_{m=1}^{\infty} m\left(\frac{n-1}{n}\right)^{m-1}\frac{1}{n}$$

$$= n$$

mens variansen til nummeret i rekkefølgen blir:

$$Var(m) = E(m^2) - E(m)^2$$
$$= n(n-1)$$

Vi ser at hvor lenge vi må vente på å få sende vil variere veldig fra gang til gang, fordi variansen her er stor.

Ved å se litt nærmere på hva slags ventetider dette blir i praksis, får vi et inntrykk av hvor realiserbart dette er. Forventet ventetid mellom hver pakke som sendes vil ligge på et titalls millisekunder (f eks 45 ms for n=4). Klokkeraten i MRR er på 2.4 kbit/s, noe som medfører en klokketid på 0.43 s for en pakke på 128 byte. Grovt sett kan vi si at det vil ta i størrelsesorden 0.5 s å sende en pakke, iberegnet ventetid og klokketid. Det er lett å se at hvis det er flere med på konferansen og dette i tillegg skal sendes over TADKOM, må konferansedeltakerne være omtrent de eneste brukerne av nettet med så høy prioritet for at dette skal være gjennomførbart. Vi antar at en forsinkelse over nettet på stort mer enn 2-3 s vil virke distraherende for deltakerne.

Den store forsinkelsen ved bruk av MRR gjør at direkte tilkobling til TADKOM er mest aktuelt. Forsinkelsen i TADKOM er sammensatt av mange små deler der summen hovedsaklig er bestemt av trafikkpåtrykk og det antall hopp som blir mellom avsender og mottaker. Hvis en ser på en enkelt svitsj vil forsinkelsen være sammensatt av køing av pakker med lik eller høyere prioritet, utklokkingstid og venting på pakker som behandles eller ligger i buffer i svitsjen.

Vi ser her for oss at TADKOM er belastet med en del trafikk med lavere prioritet f eks oppdatering av situasjonsbilde. Vi forutsetter at applikasjonen " felles markør på felles arbeidsflate" har høyere prioritet en dette. Så lenge det ikke går våpentrafikk går vi ut i fra at "vår" applikasjon er den eneste som generer trafikk med høy prioritet.
Ved lav totaltrafikk og ideelle forhold vil forsinkelsen for en pakke bestå av utklokkingstid og prosesseringstid.Antall bit i en pakke på 256 byte: 256 * 8 = 2048. Utklokkingstid 2048/38400 = 0.053 s. Prosesseringstid 11 ms. Forsinkelse 53 ms + 11 ms = 64 ms.

Ved stort påtrykk av pakker med lavere prioritet vil en måtte vente på inntil 3 pakker med lav prioritet som ligger i svisjen. Dette kan øke forsinkelsen med 3 * 53 ms = 159 ms. Dette gir en samlet forsinkelse på 159 ms+ 53 ms+11 ms= 223 ms. For en forbindelse som som går over 10 linker får vi da en forsinkelse på ca 2.2 s.

En annen anledning hvor offiserer møtes er ved briefinger. En kan tenke seg at også dette kunne foregå distribuert rundt på kommandoplassene. Offiserene kan da snakke sammen samtidig som de får opp samme bilde på skjermen. Disse bildene kan da distribueres ut under briefingen eller distribueres ut på forhånd slik at alle deltakerne har fått foilene/ plansjene før briefingen starter. Størrelsen på slike foiler kan variere veldig ettersom hvilket program som har laget dem. Selvsagt er størrelsen også avhengig av hvor kompliserte figurer man har, og hvor mye tekst de inneholder. Tekstbehandlingsprogrammer lager generelt "store" illustrasjoner. Vi prøvde derfor et tegneprogram kalt "xfig". Her fant vi at

en middels stor plansje (med noen sirkler, streker og firkanter samt litt tekst) hadde en størrelse på ca 2000 byte. Dette er en overkommelig størrelse også for MRR, da plansjene kan distribueres ut litt i forkant av briefingen.

Ved planutarbeidelse og annen dokumentbehandling ville det lette arbeidet og kreve mindre tid om flere kommandoplasser kunne arbeide med samme dokument samtidig. Man må da finne en bedre metode enn å utveksle word-dokumenter, da disse er veldig store og vil kreve enormt med sambandskapasitet å overføre. En mulighet kunne være å lagre planen i en database, hvor hvert punkt i planen er et element i en tabell av typen "long". Det vil si at elementene i tabellen kan være nærmest ubegrenset store (opptil 2 Gbyte). Disse elementene kunne så være skrevet i Hypertext Markup Language (HTML)-format, som er et format som ikke lager mye overhead. (Her kan størrelsen regnes tilnærmet lik ASCII-koding.)

8 KONKLUSJON

Vi har tatt for oss noen av de viktigste bitene i et informasjonssystem, nemlig meldingssystemet og databasen. Deretter har vi sett dette opp mot formidling av situasjonsbilde. Ved å benytte X.400-protokollen for meldingsformidling og Oracle databaser har vi sett at det er gjennomførbart å få oppdatert situasjonsbilde så ofte som ønskelig.

Ved oppdatering av situasjonsbilde vil vi forslå å bruke X.400 med SQL-oppdateringer og ikke bare SQL*Net. X.400 kan tilby større grad av pålitelighet, samt at det blir enklere å gjennomføre en mer hensiktsmessig form for ruting på grunn av mulighetene for distribusjonslister som finnes her. Det blir da mulig å redusere antall ganger samme pakke må sendes over samme link ved en en-til-alle oppdatering. Fordi flaskehalsen i systemet er de linkene hvor pakkeantallet er stort, vil en reduksjon i antall pakker her ha større betydning enn den økningen vi får i overhead. For linker med lite trafikk vil gevinsten være liten eller ingen, men her er det heller ikke så kritisk med kapasiteten. Helt optimal ruting med multicast finnes det ikke muligheter for med distribusjonslister i X.400, men dette er noe vi mener har stor nytteverdi. Det bør derfor vurderes om dette kan være gjennomførbart på sikt. Hvis man implementerer multicast slik at databasen blir i stand til å benytte seg av dette vil mye av fordelene med X.400 falle bort.

Vi kan velge X.400 til oppdatering fordi vi har prioritert tilgjengelighet fremfor konsistens. Det er da ikke så viktig at oppdateringene av databasen skjer samtidig i tid, som å hindre at databasene skal låses hvis det oppstår feil med nettet. Nettverksfeil er noe som skjer relativt ofte ved bruk av mobile nett som i divisjonen, så vi ønsker ikke et system som er sårbart ovenfor slike feil.

Likeledes har vi funnet at det er gjennomførbart med visse typer konferanseapplikasjoner om disse tilpasses divisjonens begrensede sambandskapasitet slik at forsinkelsene blir minimale. Hyllevareprodukter egner seg stort sett ikke her p g a stort trafikkpåtrykk og dermed stor forsinkelse.

Litteratur

(1) Rapport etter studie for Taktisk Meldingshåndteringssystem, Hærens fosyningskommando 7. Februar 1995, Begrenset.

(2) Betanov Cemil, An Introduction to X.400, Artech House, 1993, Ugradert

(3) Litlere Arne, Elektronisk post - X.400, OSI standarder for verdiøkende tjenester, Artikkelsamling, rapport 840, Norsk Regnesentral, 1990, Ugradert.

(4) Tanenbaum Andrew S, Computer Networks, Prentice-Hall International Editions, 1989, Ugradert.

(5) Specification of Basic Encoding Rules for ASN.1, CCITT Recommendation X.209, Melbourne 1988, Ugradert.

(6) Presentation Service Definition for OSI for CCITT Applications, CCITT Recommendation X.216, Melbourne 1988, Ugradert.

(7) Association Control Service Definition for OSI for CCITT Applications, CCITT Recommendation X.217, Melbourne 1988, Ugradert.

(8) Remote Operations, Model, Notation and Service Definition, CCITT Recommendation X.219, Melbourne 1988, Ugradert.

(9) Message handling system and service overview, CCITT, Recommendation X.400, Melbourne 1988, Ugredert.

(10) Ceri Stefano and Pelagatti Giuseppe, Distributed Databases Principles and Systems, McGraw-Hill Company, 1984, Ugradert.

(11) Systemspesifikasjon for Taktisk Meldingshåndteringssystem, Hærens forsyningskommando, Begrenset.

(12) Bergene T, Sundfør H O, Sørheim J: Beskrivelse av stabs- og ledelsesprosesser for DIV 2000 som analysegrunnlag for prosjekt 671, KKI-Hær, FFI/NOTAT-97/00635 Forsvarets forskningsinstitutt, Begrenset.

(13) Message Handling in MRR, Study Report, Alcatel Telecom Norway A/S, 28. Sept. 1992, Ugradert.

(14) ARMY, Col. Rosenberger J D: A year in the EXFOR, November 1996, Ugradert.

Forkortelser

ACSE	Association Control Service Element
ADMD	Administrative Management Domain
ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation one
AU	Access Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DCE	Data Circuit Terminating Equipment
DTE	Data Terminal Equipment
FDN	Forsvarets Digitale Nett
FTP	File Transfer Protocol
HDLC	High Level Data Link Control
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
K2IS	Kommando Kontroll og Informasjonssystem
LAPB	Link Access Procedure B
LI	Lengdeidentifikator
LLC	Logical Link Control
MAC	Media Access Control
MASE	Message Administration Service Element
MD	Management Domain
MDSE	Message Delivery Service Element

MRR	Multi Rolle Radio
MRSE	Message Retrieval Service Element
MS	Message Store
MSSE	Message Submission Service Element
MTA	Message Transfer Agent
MTS	Message Transfer System
MTSE	Message Transfer Service Element
NORCCIS II	Norwegian Command Control and Information System II
NVT	Network Virtual Terminal
PDAU	Physical Delivery Access Unit
PGI	Parametergruppeidentifikator
PI	Parameteridentifikator
PLP	Packet Layer Protocol
PRMD	Private Management Domain
ROSE	Remote Operations Service Element
RTSE	Reliable Transfer Service Element
SI	SPDU Identifikasjon
SPDU	Session Protocol Data Units
SQL	Standard Query Language
TADKOM	Taktisk Digitalt Kommunikasjonssystem
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/ Internet Protocol
TELNET	Telecommunications Network
TMHS	Taktisk Meldingshåndteringssystem

TSAP	Transport Service Access Point
UA	User Agent
UDP	User Datagram Protocol

FORDELINGSLISTE

FFIE **Dato:** 6 august 1997

RAPPORT TYPE (KRYSS AV)	RAPPORT NR	REFERANSE:	RAPPORTENS DATO:
X RAPP ☐ NOTAT ☐ RR	97/03389	FFIE/671/161.2	6 august 1997
RAPPORTENS BESKYTTELSESGRAD: UGARDERT		ANTALL EKS UTSTEDT: 80	ANTALL SIDER: 49
RAPPORTENS TITTEL: DATADISTRIBUSJON I DIVISJONENS K2IS		FORFATTER(E): FARSUND Bodil Hvesser, JENSVOLL Audun, SANDER Jostein	
FORDELING GODKJENT AV FORSKNINGSSJEF:		FORDELING GODKJENT AV ADM DIREKTØR:	

EKSTERN FORDELING

ANTALL	EKS NR	TIL
3		FO/HST
3		HFK
1		FKN
2		INFINSP
2		KAVINSP
2		ARTINSP
1		ARTINSP/Stedfortredende
2		INGINSP
3		SBINSP
1		TRENINSP
1		SANINSP
1		DKN
2		TLF/DIV6
2		FSTS/Hæravd
1		FTD

INTERN FORDELING

ANTALL	EKS NR	TIL
14		FFI-BIBL
1		Adm dir/Stabssjef
1		FFIE
1		FFISYS
1		FFITOX
1		FFIU
1		FFIVM
1		Audun Jensvoll, FFIE
1		Bodil Farsund, FFIE
1		Bjørn Skeie, FFIE
1		Bjørn Solberg
1		Einar Meek, FFIE
1		Frode-Johan Lillevold, FFIE
1		Geir Egeland, FFIE
1		Georg Anuglen, FFIE
1		Jostein Sander, FFIE
1		Kjell Kristiansen, FFIE
1		Mona Ofigsbø, FFIE
1		Olav Berg, FFIE
1		Ove K Grønnerud, FFIE
1		Snorre Prytz, FFIE
1		Sten Amundsen, FFIE
1		Svein Haavik, FFIE
1		Tor Neple, FFIE
1		Vivianne Jodalen, FFIE
1		Asbjørn Taugbøl, FFISYS
1		Bjørn T Bakken, FFISYS
1		Hans-Olav Sundfør, FFISYS
1		Ingar Moen, FFISYS
1		Ivar Farup, FFISYS
1		Reidar Skaug, FFISYS
1		Sverre Braathen, FFISYS
1		Tone Hafstad Dale, FFISYS

FFI-K1 Retningslinjer for fordeling og forsendelse er gitt i Oraklet, Bind 1, Bestemmelser om publikasjoner for Forsvarets forskjningsinstitutt, pkt 2 og 5. Benytt ny side om nødvendig.

EKSTERN FORDELING

ANTALL	EKS NR	TIL

INTERN FORDELING

ANTALL	EKS NR	TIL
1		Tor Langsæter, FFISYS
1		Trond Bergene, FFISYS
5		Arkiv FFIE