Samlede notater for

Datamatikeruddannelsen

1. – 3. semester

Netværk

Udgave nr. 1

Skrevet af

Alice Raunsbæk

(alice@m-r-a.dk)

Senest opdateret

Lørdag d. 10. juni 2006

1. SAMLEDE NOTATER FOR DMU (1-3. SEMESTER)

1.1. Forord

1.1.1.1. Opdeling i nogle udgaver

Indenfor uddannelsen er der som sådan fire emner man kan dele fagene op i: virksomhedsforståelse, systemudvikling, programmering og netværk. Grundet størrelsen af dokumentet findes disse typisk i forskellige dokumenter – du kan på forsiden (og forhåbentlig på filens navn) se hvilken udgivelse du har fat i.

Typisk forefindes udgivelser kun i PDF. Ønskes de i et andet format kontakt da redaktøren.

1.1.1.2. Tegn og tekst forklaring

Forskellige ting har forskellige formateringer.

```
Hvis du ser tekst af denne type er der tale om java-kode
(eller kode i det hele taget).
```

For forståelsens skyld er der iblandt notaterne også opgaveløsninger. Dem kan du kende på den grå farve denne tekst også har.

Over den grå tekst eller inden i den ser du måske noget med denne lysere farve – det er opgave formuleringer.

Forklaringer

> Når noget skal uddybes så har det typisk en normal linje (eller en med fed) hvorefter der er en indrykket tekst – det er forklaringen på/beskrivelsen af linjen.

1.1.1.3. Opbygning

Da der er en del notater og dette er et meget langt dokument har jeg valgt at have to indholdsfortegnelser, en overordnet – så du lettere kan finde et fag eller et emne. Den indeholder kun to niveauer: Fag og Emner.

Den mere detaljerede indeholder alle niveauer (der er ned til 4. niveau).

1.1.1.4. Forbehold

Disse notater er skrevet på baggrund af min egen forståelse af hvad lærerne har forkyndt i timerne. Jeg kan derfor ikke garanterer for at de er 100% korrekte.

Der kan endvidere være 'huller' i notaterne, dette kan enten skyldes at jeg synes at det virkede himmelråbende logisk da jeg skrev notaterne eller at jeg fattede hat af emnet.

Finder du ellers fejl (eller mangler) er du velkommen til at skrive til mig med rettelser.

Mvh. Alice Raunsbæk

alice@m-r-a.dk

1.2. Overordnet indholdsfortegnelse

1.3. Detaljeret indholdsfortegnelse

2. NETVÆRK

Netværk dækker over Computer Arkitektur & Operativ Systemer (CAOS) og Computer Netværk og Distribuerede Systemer (CNDS).

3. CAOS, 2. SEMESTER

3.1. Database [Rama]

Målet med databaser er at vi selv skal kunne gøre en masse og have en forståelse for hvordan det fungerer inde bagved.

- Gemme oplysninger
- Let at læse (for programmer)
- Gem på en bestemt måde
- Strukturerede data
- På elektronisk form

3.1.1.1. DBMS

= DataBase Management System

Holder styr på en eller flere Databaser (f.eks. SQL Server 2000).

Hvorfor DBMS?

I EDB'ens barndom:

Program -> Fil

Udvikles til:

Program -> DBMS -> Fil

- DBMS = alle generelle faciliteter (f.eks. til at hente og gemme i fil samt tage backup)

3.1.1.2. Faciliteter i DBMS

- Et forespørgselssprog (SQL)
- Effektiv datafremfinding (indeks)
- Backup/restore*
- Samtidighedskontrol*
- Alt eller intet – logik*
- Sikkerhed (Hvem må se hvad)
- Dataintegritet (Regler om data)
- …

* = Transaktioner

3.1.1.3. Nyere faciliteter

- Programmeringssprog (f.eks. Transact SQL, PL SQL, P SQL)
- Automatisering
- Diverse utilities

3.1.1.4. DBMS'er

- DB 2 (mainframe) af IBM *
- Oracel (UNIX, Windows, …) den ældste *
- MS – SQL Server (Windows) *
- MySQL (UNIX, Windows) gratis
- Firebird (Windows) gratis
- Paradox (Windows)
- Access(Windows) Database & udviklingsværktøj
- …

* = Toppen af poppen

3.1.1.5. Den Relationelle model

- Hierarkisk *
- Netværks *
- Relationel <- Totalt dominerende
- Object <- Måske engang i fremtiden

* = Historiske – effektive men langsomme at kode op imod

Object-databaserne har været svære at få ind, derfor er der i nyere tid blevet lavet Relationelle databaser med strejf af Object som et forsøg på at få det objektorienterede databaser ind på markedet.

3.1.1.6. Regler i relationel model

- Attributter skal være atomare (der må kun være en oplysning i hver attribut)
- Nøgler i alle tabeller

3.1.2. SQL

Structured Query Language

DDL = Data Definition Language

Hvordan tabellen ser ud

- create create table tabelnavn (felt1 type, felt2 type)
- drop
- alter kun til ekstra felter, kan sætte dem på I enden af den eksisterende tabel

DML = Data Manipulation Language

Hvordan man ændrer I tabellen/oplysningerne

- insert insert into tabelnavn values(, , ,) // i rigtig rækkefølge
  insert into tabelnavn (felt1, felt2) values ('Hej',13)
- update update tabelnavn set felt2 = 17 where felt1 = 'Hej'
- delete delete from tabelnavn where felt2=13
  delete from tabelnavn // sletter alt fra tabelnavn
- select

Create/drop table

- Ret let at ændre i tabeller under udviklingsfasen
- Når "systemet" er taget i brug er der brug for en konvertering
    - Flyt data fra tabellerne til fil/midlertidig tabel
    - Drop table
    - Create table
    - Lav et konverteringsprogram, der kan "flytte" data fra gammelt til nyt format

3.1.2.1. Datatyper

Tekst

CHAR(10)

Holder altid 10 pladser Til små felter (under 18 karakterer), eller har man et godt bud på længden,

VARCHAR(40)

Bruger kun den plads der er fyldt i men max den angivne længde

Heltal

TINYINT

(1 byte)

SMALLINT

(2 bytes)

INT

(4 bytes)

BIGINT

(8 bytes)

Decimaler

DECIMAL

decimal tal (foretrukne)

REAL / FLOAT

flydende tal

Dato

DATETIME

Andet

BIT

Boolean (0 = false, 1 = true)

TEXT

Uendelige mængder af karakterer (alle andre op til 8.000 karakterer)

BLOB

Billeder

3.1.2.2. Constraints

Kan skrives dirrekte efter et felts definition. Skrives noget mellem constraint og constraint-typen er dette navnet på den pågældende constraint.

Primary key

Nøgle, der kan kun være en primary key pr. tabel.

Unique

Værdierne skal være unikke. Der må gerne være flere unikke felter pr. tabel.

Foreign key

Checker fremmednøgler. Pågældende tabel skal være oprettet først og feltet skal være primary key i tabellen.

Check

Undersøger værdier. (feltnavn between x and y)

Hvad nu hvis jeg gerne vil have kontrolleret noget som constraints ikke kan klare.

Betingelse

En Person må højest være ansat i 3 Firmaer

Løsning

Anvende triggere (beskrives sernere), består reelt af en lille smule kode

Andet

Identity(startværdi,forskydning)

Autonummerering af indsættelser. Skrives typisk efter primary key.

Not null

Skrives efter felt-definitionen. Tillader ikke at der indsættes null i feltet.
Dvs. at bestemme at et felt skal have en værdi.
Felter, der er primary key eller uniqe er altid NOT NULL.

Cascade

Skrives på en forgein key.

```
Create table Ansati (
Cpr char(10) foreign key references Person(cpr) ON DELETE
CASCADE,
Firmanr int foreign key references Firma(Firmanr) ON DELETE
CASCADE,
Primary key(Cpr,Firmanr),
)
```

”Denne har ingen mening hvis personen er væk” (øverst)

“Denne har ingen mening hvis firmaet er væk” (nederste)

3.1.2.3. Select

```
SELECT attributnavn eller *
FROM tabelnavne
```

```
WHERE betingelse (sammensættes med AND, OR, NOT samt () )
evt. JOIN-betingelse, der er en join-betingelse mindre end
antallet af tabelnavne
```

Eks

```
Select * from Firma
Select * form Firma where postnr = '8000'
```

Find navne på alle Personer der bor I Højbjerg

```
Select Navn from Person p, Postnummer po where
po.postdistrikt = 'Højbjerg' and p.postnr = po.postnr
```

Ny mulighed for at lave join

```
Select Navn from Person p JOIN Postnummer po ON
p.postnr=po.postnr where po.postdistrikt = 'Højbjerg'
```

Join med tre tabeller: find navne på alle personer, der arbejder på LEC

```
SELECT Navn FROM Person p, Firma f, Ansati a WHERE Firmanavn
= 'LEC' and p.cpr = a.cpr and f.firmanr = a.firmanr
```

Ny join

```
SELECT Navn FROM (Person p JOIN Ansati a ON p.cpr = a.cpr)
JOIN Firma f ON f.firmanr = a.firmanr  WHERE Firmanavn =
'LEC'
```

Distinct

Fjerner dubletter I et resultat.

Eks. Hvilke stillinger er repræsenteret i Person?

```
Select distinct stilling from Person
```

Giver "systemudvikler" en gang I stedet for flere

As

Gør det lidt hurtigere at skrive queries

```
SELECT Navn AS 'Personnavn' FROM Person AS p, Firma AS f,
Ansati AS a WHERE Firmanavn = 'LEC' and p.cpr = a.cpr and
f.firmanr = a.firmanr
```

Dog behøver man ikke at skrive AS idet der er lavet en konvention om at et mellemrum er lige så dækkende. I stedet for 'navn' kan man skrive [navn].

Like

Bruges til streng-match.
% = delstreng (ingen til mange tegn)
_ = et enkelt tegn

Find navne på alle firmaer hvis navn indeholder m

```
Select firmanavn from firma where firmanavn like '%m%'
```

Find navne på alle firmaer hvis navn indeholder m, der ikke er forrest

```
Select firmanavn from firma where firmanavn like '_%m%'
```

Union

Foreningsmængden

```
(select ......) UNION (select ......)
```

Krav: de to select-sætninger skal give strukturelt ens resultater

Den øjeblikkelige dato

```
Select getdate()
```

IN

Eks.

```
Select * from Person where postnr IN ('8000', '8240', '8270')
```

NOT IN kan bruges I stedet for IN, blot med den modsatte effekt.

Find de Personer, der bor i et postnummer, hvor der ikke er Firmaer.

```
Select * from Person where postnr NOT IN ( select postnr from
Firma)
```

Generelt: brug kun subselect, hvis det er nødvendigt!

Aggregates

COUNT()

MAX()

MIN()

SUM() (kun tal)

AVG() (kun tal)

GROUP BY

Når man laver en group by på en attribut a får man lige så mange records i resultatet, som a har forskellige værdier.

NULL

Fælles værdi for alle typer.

Alternativ til null kan være at vælge en neutral værdi.

Når man sammenligner med NULL-værdier bruges IS.

```
Select * from Person where loen IS NULL / IS NOT NULL
```

Frivillig deltagelse

Ved mange joins vil records, der ikke har en fremmednøgle i en anden tabel, "forsvinde".

Problemet kan løses på tre måder:

1. Opret en fup-record
2. UNION-løsning

```
Select navn, firmanavn
from person p, ansati a, firma f
where p.cpr = a.cpr and a.firmanr=f.firmanr
union
```

```
select navn, 'Intet firma'
form person
where cpr NOT IN (select cpr from ansati)
```

3. LEFT JOIN (de på venstre side kommer med uanset om de har en refferance på højre side eller ej. Dvs. den første tabel kommer altid med)

```
Select navn, isnull(firmanavn,'intet firma')
from person p
LEFT JOIN ansati a ON p.cpr=a.cpr
LEFT JOIN firma f ON a.firmanr=f.firmanr
```

Der findes ligeledes RIGHT JOIN.

3.1.2.4. View

Virtuel tabel

```
create view xxxx
as
"select-sætning"
```

Views kan bruges hvor man normalt ville bruge tabeller

Eks:

```
create view Person2
as
select navn, firmanavn
from Person p, Ansati a, Firma f
where p.cpr = a.cpr and a.firmanr = f.firmanr
```

Anvendelse:

```
select navn
from Person2
where firmanavn like '%CSC%'
```

Med et view kan man kun indsætte i en enkelt tabel. Ellers skal man bruge de oprindelige tabelnavne. Views kan bruges til at slette entrys samt lave select.

View er udelukkende for at gøre det nemmere at bruge SQL, der er ingen performans gevinst.

Data bliver ikke gemt i et view, men et view anvendes på samme måde som en tabel.

Fordele

Typiske sammensætninger og beregninger kan laves i views

Sikkerhed kan baseres på views

Ulemper

Ingen performansgevinst ved views

3.1.2.5. Stored procedures

En prekompileret stump SQL, der bor på serveren.

```
create proc xxxx
as
"select-sætning"
```

Eks

```
create proc AllePersoner
as
select * from Person
```

Anvendelse:

```
exec AllePersoner
```

Med parametre

```
create proc HojLoen
@graense int, @navn carchar(30)
as
select *
from person
where loen > @graense and navn = @navn
```

Anvendelse:

```
exec HojLoen 300000, 'Anna%'
```

Med returparameter

Returner gennemsnitsløn for alle, der bor i et bestemt postnr.

```
create proc GnsLoen
@postnr char(4)
@gns int output
as
select @gns=avg(Loen)
from Person
where postnr = @postnr
```

At bruge stored procedures som inputs til hinanden.

```
exec GnsLoen '8210', @xxxx output
exec HojLoen @xxxx, 'navn'
```

Fordele

Performansgevinst

Ulemper

-

3.1.2.6. Trigger

En stored procedure der kaldes automatisk ved INSERT, UPDATE eller DELETE

Eksempel

Lav en trigger, der forbyder sletning af en konto, hvis saldoen <> 0

```
create trigger xxxx
on Konto
for delete
as
if (select count(*) from deleted where saldo <> 0) > 0
begin
raiserror('Kan ikke slette konto',16,1)
rollback tran
```

```
end
```

`if exists()` kan bruges som alternative til if `select > 0`

tabellerne deleted, inserted bruges til at finde de data der vil blive slettet/tilføjet

3.1.3. Database teori

Databasen vil modtage ting ens, uanset hvem der sender.

Programmet vil gerne kunne arbejde på samme måde mod alle DBMSer.

Derfor:

Drivere ligger mellem programmerne og databaserne. Driverne opfattes som en del af programmet.

Driverne har to dele, den der vender mod programmet hedder JDBC (ved Java). Den del der vender mod databasen har to forskellige muligheder, ODBC (og en SQL-server spicifik driver/ADO/) eller en Native driver

JDBC er en standard databasegrænseflade for Java-programmer. Består basalt det af 3 klasser:

- Connection (opretter forbindelse)
- Statement (sender noget til DB)
- ResultSet (det der kommer tilbage fra DB)

ODBC

Generelt begreb – skal konfigureres,

Native driver

Kan forstå JDBC i den ene ende, den anden afhænger helt af hvilken server der skal arbejdes op imod. Native drivers er derfor lavet til at arbejde med en bestemt type af database-server.

Programmeringsforskel mellem Native og ODBC er to linjer.

SQL er ikke en del af programmeringssprogene.

Pre-kompilering (SQLJ, COBOLT, PL1)

```
EXEC-SQL
select navn INTO :navnvar from Person
where cpr = :crpvar
END-EXEC
```

Meget brugt i gamle dage, bruges stadig f.eks. på MAINFRAME.

Uden pre-kompilering (JDBC)

SQL gemmes i tekststrenge

Navngivning af constraints således der kan gives korrekte fejlmeddelelser:

```
constraint cprforeign foreign key
```

3.1.3.1. Opbygning af BD server

I/O = Disk input/output

En blok er 8k.

[[tegning]]

En isoleret disk I/O (en blok) tager 10ms.

100Mb data = 100*1000/8 = 12.500 blokke. Dvs. 125 sekunder (Dette er worst-case)

- Normalt vil nogle blokke ligge samlet og derfor tage kortere tid
- Normalt vil nogle blokke blive fundet i bufferen, og skal derfor ikke hentes på harddisken

3.1.3.2. Indeks

Et indeks er en "indholds-fortegnelse". Index er organiseret som et søgetræ.

[[tegning]]

Et indeks kan oprettes og nedlægges efter for godt befindende. Indekset opdateres og vedligeholdes hver gang der indsættes eller slettes data.

```
create index xxxx on Person(stilling)
drop index xxxx
```

Fordelen ved et indeks er at søgningen i DB bliver væsentlig forbedret.

3.1.3.3. Hvad sker der når DB modtager en SELECT-sætning

- Parser SQL (checke at den er OK)
- Omformer til relationel algebra
- Optimerer forespørgsel

3.1.3.4. Relationel algebra

Select σ: udvælger de records der opfylder en betingelse

$\sigma_{\text{stilling = 'IT-chef'}}$ Person

Project π: udvælger attributter fra en tabel

$\pi_{\text{cpr, navn}}$ Person

Join □□/$\Delta\Delta_b$: sammensætter alle records i den ene tabel med alle i den anden, hvor betingelsen b er opfyldt.

Person $\Delta\Delta_{\text{postnr}}$ Postnummer

Union U: svarer til foreningsmængden

3.1.4. Transaktioner

3.1.4.1. Hvad er en transaktion?

- Et 'hele' set fra brugeren
- Et antal SQL-kald set fra databasen

Bruger	Programmet	Database
Opfatter GUI som et hele	Modtager tastetryk og evt.	Modtager select og updates

museklik	men kan ikke se evt. sammenhænge

```
begin tran
     select
     select
     update
     update
commit / rollback tran
```

3.1.4.2. Hvad lover databasen om en transaktion

A: atomicy / *alt eller intet*

C: consistency / konsistensbevarelse (ikke kommer fejl pga. *samtidighed*)

I: isolation / mellemresultater usynlige for andre

D: durability / bevarelse el. holdbarhed (*intet tab af data*)

C + I = Samtidighedskontrol sikres ved låsning af records

A + D gør brug af log – muliggør backup

D kan der tages backup af

3.1.4.3. Valg af nøgler

Informationsbærende

En attribut(ter) fra genstandsområdet.

Ulempen kan være at man ikke kan komme til at ændre nøglen på en fornuftig måde.

Ikke informationsbærende (dumme nøgler)

Typisk en tæller.

Identity gør ikke brug af genbrug

3.1.4.4. Normalisering

Gøres der ikke meget ud af mere, idet folk har lært at tænke på en sådan måde at opbygning allerede er normaliseret.

3.1.4.5. Historik

Skal jeg huske selve hændelsen eller kun resultatet af hændelsen.

Hvis det kun er resultatet, der skal bruges er der ikke behov for historik.

3.1.4.6. Optimalisering

Performancemæssige forbedringer

F.eks. saldo på konto selv om transaktionerne også er der

Man sætter ekstra felter på som egentlig ikke behøvede at være der, men som man smider på for at gøre systemet hurtigere til daglig.

3.1.4.7. Type af sletninger

Hvis man ønsker at slette en record, kan dens nøgle optræde som fremmednøgle i andre tabeller.

- KASKADE sletning

- Hvis man ønsker det, kan logisk sletning være en mulighed. Implementeres typisk ved hjælp af en boolean på tabellen.

3.1.4.8. Samtidighedskontrol

Hvis man ikke har samtidighedskontrol kan der opstå

Lost update (begge skal opdatere WW)

T1	T2
læs x	
	læs x
x = x + 1	
	x = x +2
commit	
	commit

T1's resultat forsvinder da T2 ikke læser den nye værdi inden sin beregning.

Inkonsistent retrieval (en skal kun læse, den anden skal skrive RW)

T1: finder det samlede indestående på alle konti

T2: flyt penge fra en konto til en anden

Problemet opstår hvis T2 flytter til og fra en konto T1 har læst og en der ikke er læst endnu.

3.1.4.9. Låsning

(Se opgave 9)

To slags låse:

- Shared lås, bruges ved SELECT
- Eksklusiv lås, bruges ved INSERT, UPDATE, DELETE

(forestil badeværelse med ET toilet og FIRE håndvaske)

3.1.4.10. Låseregel

Streng 2-faset låsning:

Låse frigives først når transaktionen kommer til commit/rollback.

3.1.4.11. Granulitet

Hvor meget låses der?

- record
- blok (8k) (SQL-server default)
- tabel

3.1.4.12. Isolation levels

[Rama]p. 539

- READ UNCOMMITTED ← stort det ingen samtidighedskontrol
- READ COMITTED ← beskytter mod Dirty Read, men ikke mere
- REPEATABLE READ ← beskytter mod "almindelige" samtidighedsproblemer, men ikke Phantom-problemet

- SERIALIZABLE ← fuld samtidighedskontrol

3.1.4.13. Deadlock/livelock

Hvad sker der hvis en transaktion skal bruge et dataelement, der er låst af andre.

- Vent → fare for deadlock (T1 Wx1, Wx2 + T2 Wx2, Wx1 = de låser for hinanden)
- Giv op → fare for livelock (stopper hvis noget er låst = kan tage lang tid før det er færdig)

3.1.4.14. Håndtering af Deadlock

Se også Deadlock på side 28.

Prevention

Programmerer så deadlock undgås eller at risiko for deadlock nedsættes.

Detection

Opdage deadlock, der er opstået

- Time-out

- Wait-for-grafer (Tegning af hvem, der venter på hvem – er der en cirkel er der DL)

3.1.5. Crash Recovery

Alt eller intet + beskyttelse mod datatab.

3.1.5.1. Problemer DBMS selv kan klare

- Rollback af en enkelt transaktion
- DB-procesen går ned (f.eks. serveren slukkes)
- Disk-fejl (hele eller dele af disk ulæselig)

Dirty Page = en diskblok, hvor der er sket ændringer, men hvor ændringerne endnu ikke er på datadisken.

3.1.5.2. WAL (Write-Ahead-Log)

Der skrives til loggen inden der skrives til datadisken. Der skrives til loggen ved:

- begin, commit eller rollback af transaction
- alle opdateringer (også sletning)

Hver log record indeholder:

- transaktions nummer
- type (begin, commit, write, abort)
- hvilket dataelement (bloknummer og recordnummer)
- before/after image

3.1.5.3. Checkpoint/statusrapportering

Fungerer som et bogmærke i loggen

- Synkront checkpoint:
  Stop alle nye transaktioner, vent på at alle igangværende transaktioner bliver færdige.
  Skriv alle dirty pages til disk.
  I praksis tages synkront checkpoint hvis server stoppes pænt.

- Asynkront checkpoint:
  Skriv listen over igangværende transaktioner i loggen. Skriv alle dirty pages til datadisken.
- Fuzzy checkpoint:
  Skriv i loggen hvilke transaktioner, der er i gang og hvilke pages, der er dirty.

3.1.5.4. Recovery-algoritme

Køres, når DB-server kommer op efter nedbrud.

1. Find ud af hvilke transaktioner der skal gennemføres og hvilke der skal have rollback.
   Interessant del af loggen = fra sidste checkpoint til enden af loggen
   Undo-liste = dem, der skal have rollback
   Redo-liste = dem, der skal have commit (dem der er fundet en commit på)
2. (arbejder nedefra)
   Fjerner alt det, dem på undo-listen har lavet.
3. (arbejder oppefra)
   Gør alt det, som dem på redo-listen har lavet.

Ikke recovery på TEXT eller billeder!

3.1.5.5. Backup

Seneste backup samt log fra seneste backup og frem.

Mirroring/spejling og replikering er måder at ekstra sikre data på. Dette bruges blandt andet for at beskytte mod fysiske hændelser.

3.2. Operativ Systemer

Målet med operativsystemer er at forstå hvordan maskinen virker. Eneste programmering handler om 'tråde'.

3.2.1. Maskinopbygning

DMA = harddisken sørger for at aflevere og hente data i Memory og giver besked til CPU'en om hvornår den er færdig (interrupt). I modsætning til tidligere, hvor alt skulle om CPU både når det hentes og gemmes fra memory.

Processor

CPU, den udførende enhed/kontrolenhed, afvikler maskininstruktioner.

16-32 registrer (hver 4 bytes). Nogle bruges til OS, resten er 'user registers'

Forbindelse til memory

Program counter (PC), register der indeholder adresse på den instruktion, der skal udføres næste gang

(IR) den ordre, der eksekveres nu.

Memory

Antal bytes. Nummereret fra 0 til n, n = størrelsen. Nummeret er lig adressen.

Maskininstruktion

En maskininstruktion kan være: (dog har hver maskintype sine måder)

- Flytter ting mellem registre og memory-adresser

- ADD, SUB, MUL, DIV
- Jump
- Compare
- I/O-kald

Proces (exec)

Består af Instruktion (kode), data (variabler) og et dynamisk område (stack, heap..)

Fx

```
x = x * y

flyt x fra memory til register1
flyt y fra memory til register2
MUL register1,register2
Flyt resultat fra register1 til memory
```

Cache

Bruges til at hjælpe med at opbevare ting fra user register således at det går hurtigere.

3.2.2. Operativsystemer

- Z/OS – IBMS til mainframes (performance og sikkerhed)
- UNIX-familien
- Windows-familien
- Andre (fx MAC-OS)

3.2.2.1. OS's formål

- Gør det let at bruge maskinen (i forhold til hvis der ikke var et)
- Effektiv udnyttelse af resurser (fx CPU og RAM)
- Fremtidssikring/Udviklingsmuligheder

3.2.2.2. Udvikling i OS

1. En proces af gangen, manuelt skift (intet OS)
2. En proces af gangen, automatisk skift (kaldet monitor = forløber for OS)
3. Multiprogrammeret Batch-OS.
   Ønske: at udnytte ventetiden ved I/O
   Løsning: Flere processer af gangen skiftes til at få CPU'en
4. Online Brugere ønsker: hurtigt (små svartider), forudsigelige svartider, "retfærdighed"
   Time-sharing OS: Tvungen tidsdeling = en proces fratages CPU, hvis den har brugt CPU'en længere end en "time-slice"
   På PC-platform kom det først med Win95.
5. Virtuel memory: hele processen behøver ikke være i memory på en gang (går dog langsommere end hvis det hele lå i memory). Placeres i swop-fil

Ovenstående er 'gamle' opfindelser (60'erne-70'erne senest).

6. Tråde: opdele en proces i mindre dele

7. Fler-CPU maskiner (mange (fx 1000 CPU) er primært til meget store regne opgaver)

Adgang til CPU/skedulering kan ske efter:

- Prioritet
- SRT (shortest time remaining)
- FIFO (first in first out)
- LIFO (last in first out)

3.2.3. Processer

3.2.3.1. Processer/jobs på en server

- Online (Bruger venter aktivt)
- Asynkron (Baggrundsjob, kortfristet)
- Batch (Andre jobs, hvor ingen direkte venter)

3.2.3.2. Hvad er en proces?

- En kørende udgave af et program
- OS beskytter processer mod hinanden
- Processer kan kommunikere med hinanden
    - OS tilbyder faciliteter, kræver at processerne kører på samme maskine
    - Via netværket

3.2.3.3. Tilstand

Se figurer i bog: s.115, 118, 121, 123.

3.2.3.4. Kontrolblok

En proces består af

___
|___| PCB (Proces Kontol Blok)
|___| Instruktioner
|___| Data
|___| Stack/Heap

PCB (s 131)

Proces identifikation

- navn
- id

Processor state information

- her gemmes indhold af alle registre (fx Program Counter) ved running slut

Process control information

- proces tilstand (running, ready, waiting, halted)

- prioritet

3.2.3.5. Processkift

- Den gamle proces' registre skrives til PCB
- Den gamle proces' procestilstand ændres
- Dispatcher vælger en ny proces
- Den nye proces' registre kopieres ind fra PCB
- Den nye proces' procestilstand ændres

3.2.4. Tråde

Processer kan ikke dele variable. Kun sende beskeder til hinanden.

- En tråd er en del af en proces.
- En tråd tilhører altid én proces.
- Tråde i samme proces kan dele variable.
- Tråde har delvis deres egen tilstand. Dog ikke suspend- og slut-tilstand, idet hvis processen går i suspend eller termination vil alle tråde gøre det sammen med processen.

Java har indbygget support til tråde.

Hvad er en tråd

En del af én proces.

Visse programmer kan effektivt opdeles i tråde.

Hvad vinder man ved at opdele et program i tråde?

- Udnyttelse af flere CPU'er
- En tråd kan arbejde mens andre venter
- God programstruktur

En tråd har

- Egen tilstand (bortset fra suspend)
- Egne variabler
- Egen prioritet

3.2.4.1. Tråde fra Java

- En tråd-klasse skal extend Thread
- Der skal være en run-metode i klassen (svarer sig til main i almindelig Java)
- En tråd startes med xxx.start()

3.2.5. Kernel vs. User-level tråde

Kernel-level tråde = tråd support i operativsystemet

Skal understøttes af operativsystemet.

User-level tråde = Tråd support i programmerne

Problem: Blokerende systemkald.

3.2.6. Synkronisering

Styring af flere trådes adgang til fælles ressourcer (variabler).

Hvorfor er der problemer?

- Fler-CPU maskiner
- Tvungen tidsdeling (dvs. CPU'en kan blive frataget)

Problemer er karakteriseret ved uforudsigelighed.

Mulighed for lost update → Behov for gensidig udelukkelse.

Kritisk sektion = de kodelinjer som kun én må udføre af gangen.

Synkronisering medfører

- Fare for deadlock
- Fare for starvation (udsultning: at nogle aldrig kommer til)
- Ønske om retfærdighed

Alle synkroniserings metoder arbejder efter samme princip

<Enter kode>

Kritisk sektion (KS)

<Exit kode>

3.2.6.1. Fire løsnings kategorier

- Software: Petersons algoritme
  Se kodeeksemlper på kopier
- Hardware ← stiller krav til maskininstruktioner
  - Disable interrupt (virker kun på 1-CPU maskiner)
  - Specielle kraftige maskininstruktioner
    - Exchange operation
    - TestAndSet operation
- Semafor ← laves typisk i OS
- Monitor ← laves typisk i udviklingsmiljø/sprog

3.2.6.2. Krav til en metode til gensidig udelukkelse

- Sikrer at der kun er en i KS af gangen
- Hvis en tråd går ned udenfor KS ødelægger det ikke synkroniseringen
- Der er ikke starvation
- Hvis KS er ledig skal en tråd kunne komme ind
- Ingen antagelse om hastighed
- En tråd bliver ikke i KS for evigt

3.2.7. Optimering af CPU udnyttelse

Busy waiting betyder at man står og bruger CPU'en på at vente. Dvs. man blokerer CPU'en for det man venter på. Dette kan i Java løses ved hjælp af en sleep-funktion, der sender pågældende metode i 'blocked', efter sleep perioden er udløbet bliver metoden sat til ready.

3.2.8. Semafor

Facilitet stillet til rådighed af OS

En semafor har:

- en tæller
- en kø
- to operationer: semWait og semSignal, der kører udeleligt (dvs. to af samme type kan ikke køre samtidig, kun én wait og kun én signal)
    - semWait: tæller--, hvis tæller<0 så sæt tråden bagerst i køen (hos semaforen)
    - semSignal: tæller++, hvis <= 0 så væk tråden forrest i køen (hos semaforen)

Hvis en semafor skal bruges til gensidig udelukkelse, så initialiseres den med tæller=1 og en tom kø.

```
semWait(s)
<Kritisk sektion>
semSignal(s)
```

Mulige tæller værdier:

- 1 = ledig
- 0 = optaget, ingen kø
- -x = optaget, x i kø

3.2.8.1. Producer/Consumer

Producere leverer opgaver til en buffer (f.eks. en printkø med printjobs) hvor til der er tilsluttet en eller flere consumere (printere).

Ønsker:

- Gensidig udelukkelse: kun en må pille i køen af gangen.
- Ønske om at blive vækket hvis køen har været tom/fuld

3.2.8.2. Monitor

En klasse, hvor der kun kan kaldes én metode af gangen.

Monitor indeholder:

To ekstra: cwait og csignal (cond // condition)

3.2.8.3. Monitor I java

- Man skriver synchronizaed ud for metoden
- Cwait → wait()
- Csignal → notify() ← vækker en tilfældig
  og notifyAll() ← vækker alle

3.2.9. Deadlock

3.2.9.1. Betingelser for deadlock

- Gensidig udelukkelse
- Hold og vent

- Ressourcer fratages ikke
- Cirkulær venten

3.2.9.2. Deadlock håndtering

Prevention

Forhindre én af de fire betingelser fra at opstå.

- Opgiv i stedet for at vente
- Spørg altid efter ressourcer i en bestemt rækkefølge (fx alfabetisk, gaffel før kniv). Kræver at programmet på forhånd ved hvilke ressourcer den får behov for.

Avoidance

Hver gang inden der uddeles en ressource regnes der igennem om udleveringen kan betyde at der kan forekomme en deadlock.

- Bankers algoritme. Skal køres hver gang en ressource tildeles. (s. 269)

Detektion

Køres f.eks. hver 30sek og checker blot om der er deadlock. Hvis der er deadlock 'dræbes' en af processerne så dennes ressource(r) kan gives til en af de andre processer.

3.2.10. Memory administration

Ønske: Flere processer ad gangen i samme memory.

3.2.10.1. Partitionering

Princip: Processen ligger samlet i RAM.

Partitionering kræver at hele processen er i RAM på en gang.

To slags partitionering:

1. Statisk:
    a. RAM opdeles i partitioner på forhånd
        i. Intern fragmentering (pladsspild)
2. Dynamisk
    a. Processen tildeles sammenhængende RAM, når den starter.
        i. Ekstern fragmentering (pladsspild)

Address translation

Alle processer bruger adresse kald der baserer sig på at processen starter i 0, der for er det nødvendigt at 'oversætte' adresserne for de processer, der ikke starter i 0.

Fysisk adresse = start-adresse (offset) + logisk adresse

3.2.10.2. Paging

Ide: Både RAM og processen opdeles i små bidder kaldet frames i RAM og pages i processen. Frame-size = page-size = en fast størrelse (typiske ½k-8k).

For at holde styr på hvor hvad ligger bruges en page-tabel, der skal ses fra processen, dvs. 'min page 0 ligger i frame 7'.

Minimalt pladsspild idet der i gennemsnit 'spildes' en halv frame pr process.

Address-translation

Eksempel: (frame-size = 2k) [0-2047, 2048-4095, 4096-6143, …]

P1: Logisk adresse = 4200

Lav heltalsdivisionen 4200 div. 2048 = 2 = page nummer

Tag modulus af ovenstående 4200 mod 2048 = 104 = offset

Dvs. Logisk adresse 4200 er i page 2 offset 104.

Slår op i page-tabel og ser at page 2 er i frame 3.

Fysisk adresse: frame 3 offset 104 (= 3*2048 + 104 = 6144 + 104 = 6248)

3.2.10.3. Virtuelt lager (Virtual Storage)

Hele processen behøver IKKE at være i RAM. Resten befinder sig på harddisken. Styringen af hvor det ligges styres af OS.

I praksis hænger virtuelt lager altid sammen med paging.

Grunden til at virtuelt storage virker er lokalitetsprincippet. Indenfor et kort tidsrum vil et program anvende få pages.

3.2.11. Algoritmer (politikker) i virtual storage

3.2.11.1. Fetch-politik

Hvem skal ind i RAM?

- Demand-paging
  Tag en page ind i RAM, når den skal bruges.
- Prepaging
  Forsøg at forudse, hvad der er brug for lige om lidt.

3.2.11.2. Placement

Hvor skal en page placeres? → Helt lige gyldigt.

3.2.11.3. Peplacement-politik

Hvis der ikke er ledige frames, hvem skal så ud af RAM?

- Den, der er længst tid til, nogen skal bruge. ← Umuligt.
- Kigge på fortiden, 3 muligheder:
  - FIFO = First In First Out
  - Least Recently Used ← god, men tung
  - Clock ← baseret på en used bit, nem og hurtig at implementere

3.2.11.4. Cleaning-policy

Hvornår smides indholdet af en frame ud af RAM?

Checker en modifyed bit – hvis den er 'sand' skal page gemmes på harddisken ellers smides den bare væk.

- Demand cleaning
  Når en anden skal bruge pladsen.
- Precleaning
  Holder et lille antal frames ledige.

3.2.12. Kø teori

Ankomst frekvens → Kø → Behandler

Der ankommer λ pr tidsenhed. Tilfældig ankomst (lamda)

Behandlingstid. (my)
Der behandles μ pr tidsenhed

Middel svartid = 1 / (μ- λ), μ > λ

Belastning = λ /μ

3.2.13. Filsystemer

Filsystemets formål:

- Skjule harddisk detaljer for brugeren (f.eks. hvor hvad ligger)
- Navngivning af filer + directory
- Deling af filer
- Access control

3.2.13.1. Fil (set fra filsystemet)

- et antal bytes med et navn

Enkelte filsystemer kan arbejde med strukturerede filer, dvs. en fil består af records.

3.2.13.2. Fil-organiseringer

- flad fil (ingen søgemuligheder ud over gennemløb)
- indekssekventielle filer
  (filsystemet har opbygget et indeks over filen)

3.2.13.3. Hvordan styrer filsystemet harddisken

- Samlet/konseduktivt
  - Performance godt
  - Svært at udnytte pladsen = fare for fragmentering
- I klumper (så store som muligt)
  -
- Kædet liste af diskblokke
  - Performance skidt
  - Udnytter harddisken godt

3.2.14. ???

Middel svartid kurve (se opgave 19) er god at huske.

3.2.15. Distribuerede OS

[OS]-\ /-[OS]

(Net)

[OS]-/ \-[OS]

Når et OS har for meget at lave kan den give processer til en ledig maskine. Et problem ved denne løsning er dog at mange processer har en del referencer til den lokale maskine, afvikling

på en anden maskine vil derfor kræve en del kommunikation mellem de to maskiner for at processen får oplysningerne med.

3.2.16. Computerarkitektur

Mængden af maskininstruktioner som en CPU tilbyder kaldes maskinens instruktionssæt.

Jo flere og jo kraftigere maskininstruktioner jo sværere og dyrere er det at lave CPU'en til gengæld vil programafviklingen være hurtigere.

Høj-niveau sprog (FORTRAN (55), ALGOL, COBAL, PL-1, PASCAL, C, C++, Java, C#) blev indført for at man ikke behøvede at kode i maskinkode.

Denne teknik kræver en oversætter/fortolker per kombination af sprog og instruktionssæt. Og disse var besværlige at lave. = Dårlig flytbarhed.

Der indførtes mellem-kode, som høj-niveau sproget kunne oversættes til. Denne Mellem-kode var generel for sproget.

Samtidig indførtes arkitektur-instruktionssæt, som lå over instruktionssættet. Afhængig af hvilken CPU i serien man havde, blev arkitektur instruktionssættet oversat direkte eller fortolket (idet instruktionerne fra arkitektur instruktionssættet ikke var kendt af CPU).

Arkitektur-instruktionssæt (S/360) er fra 1964.

Hermed skulle der blot oversættes/fortolkes fra mellem-kode til arkitektur instruktionssæt. Derfor blev flytningen af programmer gjort meget lettet, i processen er der dog tabt noget performance.

3.2.17. Linkning

Et spørgsmål om at sætte nogle referencer rigtigt på plads.

Moduler kan compiles til OBJ-filer, disse linkes så sammen (linkning) til en EXE-fil (eksekverbart modul). (statisk linkning)

3.2.17.1. DLL: Dynamic Link Libraries

Indeholder/består af kode/kodestumper.

- Egentlig opfundet til fælles kode, dvs. kode som flere processer bruger.
- I Windows er en del af OS placeret i DLL'er.
- DLL'er kan bruges til dynamisk at lave programændringer

3.2.18. Java Virtual Machine (JVM)

For JAVA er flytbarhed et _vigtigt_ mål.

JAVA-program oversættes til byte-kode (.class filer). Byte-kode 'fortolkes' ned imod instruktionssættet = JVM.

JVM er en slags fortolker af byte-kode. Men da byte-koden ligger et godt stykke fra maskinkode tager 'fortolkningen' med JVM en del tid. Derfor er Java ikke specielt performance venligt. Maskinerne i dag har dog så stor CPU-kraft at det ikke er et problem.

3.2.18.1. Just In Time – kompilering (JIT)

Teknik til at undgå 'fortolkning'. Første gang en class fil køres laves en oversættelse af filen, der så kan bruges når filen skal køres efterfølgende, dvs. det bliver hurtigere ved gentagende kørsler.

3.2.18.2. Hot Spot

Forsøger at regne ud hvilke brugeren vil bruge tit (oversættes optimeret) og hvilke brugeren vil bruge sjældent (oversættes uoptimeret eller slet ikke).

3.3. Samlet DB og OS

3.3.1. Database arkitektur

Program --- Tabeller = ikke OO DB struktur, svært at vedligeholde/opdatere ved større systemer.

Bruger --- Controller --- Model --- DB = OO struktur, let at vedligeholde og genbruge, dog en begrænsning på 4 Gb. Model fx JPOX.

Bruger --- Controller --- DB = OO, hurtig udvikling, ikke komplekse systemer, - vedligeholdelse, - genbrug

Anbefalet til projekt at opdateringer laves gennem JPOX. Aflæsninger må gerne laves udenom JPOX, hvilket er praktisk til f.eks. statistik.

3.3.1.1. Flerbrugersystemer

Hvis man skal lave et flerbrugersystem vil man typisk ligge databasen på en server og modellen (JPOX) på en server således at hver bruger kun har controlleren til at køre på sin egen maskine. For at kommunikere mellem C og Model bruges RMI, hvorved det teknisk fungerer som om det ligger på samme maskine.

3.3.2. Projektet på 2. semester

- Spørg i tide!
- Gør jer klart hvad I vil vide!

Torben er der de fleste formiddage – normalt på sit kontor. Send en mail, hvor der står hvor vi sidder, så kommer han. Er ikke inde i Swing eller generel programmering af GUI.

I første omgang får vi kravene for ITO og SD, efterfølgende kommer det fra CAOS og SK (nok efter 2 uger).

Husk at have alle fag med, også i rapporten. Maksimum sidetal (muligvis på 60 sider) foruden bilag.

Når klassediagrammet er færdigt og vi skal lave databasen vil Torben gerne 'snakke' med os for at sikre at klassediagrammet ser fornuftigt ud. (Send en mail)

3.4. Repetition til Eksamen på 2. semester

3.5. Operativsystemer

Bog: William Stallings, Operating Systems, Fifth edition, 2005, Prentice Hall, ISBN 0-13-127837-1

3.5.1. Kapitel 2.2

- Multiprogrammeret OS ← Udnytte ventetider ved I/O (Frivillig tidsdeling)
- Time-sharing OS ← Online brugere ønsker forudsigelige svartider (Tvungen tidsdeling)

3.5.2. Kapitel 3

- Proces-begrebet
    - Hvad er en proces
    - Tilstandsdiagram (Ready -> Running -> Blocked)
    - Procesbeskrivelse (s. 131)
        - Hvordan den gemmer registre, når der skiftes proces
            - Registre gemmes over i proces kontrolblokken

3.5.3. Kapitel 4.1

- Tråde
    - Tråde kontra Proces
        - En tråd er en del af en proces
        - En tråd tilhører én proces
        - Delvis sin egen tilstand (ikke suspend og slut)
        - Kan dele variabler med moder processen og andre tråde

3.5.4. Kapitel 5

- Synkronisering & gensidig udelukkelse
    - Hvorfor
        - Når programmet er trådet
        - Tvungen tidsdeling
        - Flere CPU-maskiner
    - Metoder
        - Monitor
        - Petersons algoritme (Software)
        - Semafor
        - Hardware-løsninger

3.5.5. Kapitel 6

- Deadlock
    - Had er det?
    - Teknikker til at undgå
        - Prevention (altid tage i bestemt rækkefølge)
        - Avoidance (checke om ressource kan skabe DL)
        - Detektion (checke om der er DL)

3.5.6. Kapitel 7 & 8

- Hvad er problemet = flere processer deler samme RAM

- Løsninger
    - Partitionering ← En proces ligger samlet
    - Paging ← En proces ligger spredt (i frames)
- Virtuel Storage
    - Kun en del af processen er i RAM

3.5.7. Kapitel 12

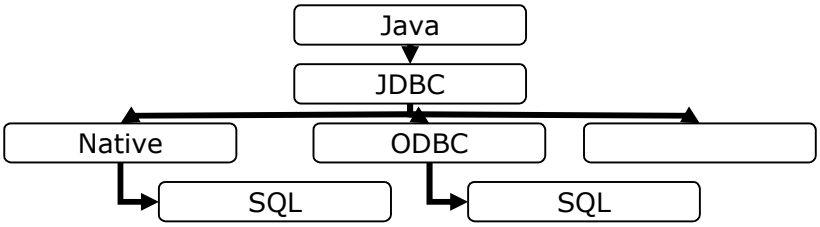
- Sandsynligvis ikke nødvendigt

3.6. Database

Ramakrishnan et Gehrke, Database Management Systems, Third Edition, 2003, McGraw-Hill, ISBN 0-07-246563-8

3.6.1. Kapitel 1

- Ikke så væsentlig
- Generel introduktion



3.6.2. Kapitel 3.1 – 3.4

- create table
- constraints
    - on delete cascade kan kun bruges ved ubetinget sletninger (dvs. ingen undtagelser fra slette-reglen)
    - primary kay (samme som unique, men der kan kun være én)
    - forigen key
    - unique
    - check (bruges ikke så ofte i den virkelige verden)
- insert, update, delete

3.6.3. Kapitel 5

- select i SQL ← VIGTIGT

3.6.4. Kapitel 6

- JDBC (kun hvis det er brugt i projekt)
    - Prepared statement

3.6.5. Kapitel 16 & 17 & 18

- Transaktioner ← VIGTIGT
    - Tre aktører
        - Bruger: Har en opfattelse af ”Et hele”
            - Fx en pengeoverførsel

- Program (lavet af programmør): "Transaktion start, indhold, transaktion slut"
  - Fx begin tran, select, select, update, update, commit
- Databasen: "En transaktion bestående af et antal SQL kald"

o Hvad lover DB om en transaktion
- Alt eller intet
- Samtidighedskontrol
- Bevarelse (intet tab af data, hvis f.eks. en harddisk går og loggen ligger på en anden eller strømsvigt)

3.6.6. Andet

- View
- Stored procedure
- Trigger

4. CNDS, 3. SEMESTER

4.1. Introduktion til faget

4.1.1. Computer Historie / IT udvikling

50'erne

Regnekraft, talknuseri/evnen til at regne

60'erne

Administrativ EDB, evne til at huske oplysninger og finde dem hurtigt frem igen,

70'erne

Netværk

4.1.1.1. Udvikling

I 80'erne kommer mindre og billigere maskiner på markedet (afdelingsmaskiner)

- UNIX
- AS/400
- PC'er i et LAN

4.1.2. Driftsmodeller

4.1.2.1. Central drift

(Mainframe-form, dvs. en maskine med al processorkraft og data, online forbindelse.) Stadig udbredt. Stabilitet og evne til store systemer. Historie (dyrt at omlægge systemer)

Fordele

- Stordriftsfordel
  - Nødstrømsanlæg, brandsikring

    - Effektiv udnyttelse af specialudstyr
    - Stor ekspertise samlet
    - Mulighed for døgnbemandning
- Let integration til andre systemer på samme maskine
- Hurtig, kraftig maskinel
- Stor dataaktualitet
- Ingen EDB-folk hos brugerne
- Høj udnyttelse af CPU-kraft
- Let at installere og udskifte programmer

Ulemper

- Relativ dyr hardware (den gang)
- Meget personalekrævende (specialister)
- Sårbar (både linie og maskinnedbrud)
- Forøget svartid på grund af netværk
- GUI ikke muligt (den gang)

4.1.2.2. Opgave 1

Til X-køb ing kommune er der oprettet en 9600 bit per sekund linie. Kommunen har i øjeblikket 20 skærme og 5 almindelige printere, der anvender denne linie. Der anvendes til alle systemer tegnsættet EBCDIC (8-bit karaktersæt).

Spørgsmål a

> Kommunen har lige hjemkøbt en hurtigprinter (1200 linier per minut) til udprintning af breve i A4-format. Kommunen havde tænkt sig, at også denne printer skulle kobles på den samme linie. Kommenter om denne løsning er holdbar.

Idet linjen er 1200 tegn pr. sekund og printeren kan tage 1200 linjer per minut hver bestående af 60 tegn (i alt 72000 tegn i minuttet eller 1200 tegn i sekundet).
Det betyder at hurtig-printeren vil kunne bruge hele forbindelsen.

Spørgsmål b

> Via skærmeme kan man køre en række systemer, der alle er fuldskærmsorienterede (dvs man udfylder et helt skærmbillede inden dette sendes over linien lige som man altid får et helt skærmbillede retur). Et typisk skærmbillede fylder 1000 bytes. Hvor meget af brugerens svartid skyldes netværket?

Da et skærmbillede er 1000 bytes, svarende til tegn, skal der 2000 tegn igennem før svartiden er overstået.
Dvs. svartiden er på 1 2/3 sekund såfremt ingen andre bruger forbindelsen.

Spørgsmål c

> Du har sikkert under de to første spørgsmål set bort fra en række faktorer, der ville kunne gøre transmissionen langsommere. Nævn disse faktorer. Der findes faktisk også faktorer, der kunne gøre den hurtigere. Kunne du forestille dig nogle muligheder.

Andre brugere på linjen, ventetid på mainframe, ustabilitet på forbindelsen. Retransmissioner (eller støj), lysets hastighed, headere + trailere.

Hurtigere forbindelse: komprimering, brutto/netto (kun sende forskellen på de to skærmbilleder)

Spørgsmål d

Hvad vil der ske, hvis alle 20 skærme sender et skærmbillede samtidigt?

Hvis alle 20 sender samtidig vil der skulle 1000*2 *20 tegn igennem forbindelsen før alle har fået svar. Dvs. 40.000 tegn.
Det kan betyde at alle må side og vente i 33 1/3 sekund.

4.1.2.3. Decentral drift

(En maskine ved hver location, dvs. delt processorkraft og data, ingen online forbindelse.) Stadig en mulighed. FTP har afløst magnetbåndene

En lokal "maskine" uden online forbindelse til andre.

Fordele

- Billige maskiner
- Mulighed for GUI
- Stor selvbestemmelse hos brugerne (autonomi)
- WAN-trafik undgås (bedre svartid, mindre sårbarhed)

Ulemper

- Dårlig dataaktualitet for fælles data
- Kræver EDB-personale hos brugerne
- Vanskelig integration med virksomhedens andre systemer
- Fare for dataanarki
- Lidt vanskeligere installation og udskiftning af programmer

4.1.2.4. Distribueret drift

(Online forbindelse.) Lettere (billigere) at lave end førhen. Mere distribution af processing end data! Sikkerhed er en udfordring

Taget det bedste fra den centrale og decentrale.

Fordele

- Billige maskiner
- Mulighed for GUI
- Nogen selvbestemmelse hos brugerne (autonomi)
- God dataaktualitet
- Mulighed for integration med andre systemer

Ulemper

- Kræver WAN
- Kræver EBD-personale hos brugerne
- Udviklingsmæssig dyr (kompleks software)
- Vanskeligere installation og udskiftning af programmer

4.1.2.5. Opgave 2

Denne opgave handler om at analysere mulighederne for at omlægge et system til afregning af lægeregninger fra central til decentral eller distribueret drift. Den centrale drift afvikles i udgangspunktet for opgaven fælles for hele landet på een mainframe. Fakta omkring opgaven præsenteres her kort.

Når en patient er hos lægen udfyldes en seddel med patientens cpr-nummer, lægens ydernummer (et nummer, der entydigt identificerer lægen) og en række koder, der angiver hvad lægen har foretaget på/ved patienten. Disse koder er nødvendige, idet lægen får betaling efter hvor meget han faktisk har udført (dvs der er koder for øreskylning, receptudskrivning, fjernelse af fremmedlegemer fra næseregionen osv.). Hver uge sender lægen lægeregningerne til det amt, hvor lægen bor. Hos amtet tastes regningerne ind. I forbindelse med indtastningen kontrollerer systemet at cpr-nummer, ydernummer og behandlingskoder er valide. En gang per måned køres en afregningskørsel på de indtastede regninger. Denne afregning resulterer i, at der via PBS tilsendes de enkelte læger en betaling svarende til de indsendte regninger. Dette betyder, at et amt altid afregner med de læger, der bor i amtet. Amterne skal altid betale for patienterne i eget amt men naturligvis ikke for patienter fra andre amter.

Hvis en læge i amtet således har behandlet en patient fra et andet amt, betaler lægens hjemamt i første omgang regningen, men sammentæller under afregningskørslen hvilke beløb den således har lagt ud for andre amter. Disse udlæg amterne imellem opgøres ved en efterfølgende kørsel og evt. tilgodehavende/skyldige beløb afregnes via et specielt offentligt refusionssystem.

Spørgsmål a

> Lav en simpel datamodel (E/R-model eller klassediagram) for dette system og omform efterfølgende til et antal relationelle tabeller.

Omformning til relationel tabel

Amt	amtsnr, navn
Afregning	afregningsnr, beløb, *skyldneramt*, *tilgodehavendeamt*
Læge	ydernr, navn, pbsnr, *amt*
Patient	cpr, navn, *amt*
Behandlingstype	behandlingskode, tekst, pris
Behandling	behandlingsnr , *ydernr*, *cpr*, dato
Behandling_type	*nr, kode*

Spørgsmål b

> Hvilke fordele ville man kunne opnå ved at gå fra central til decentral/distribueret drift.

Billigere maskiner, mulighed for GUI, kortere svartider
Bedre svartider
Mere robust system, da data er flere steder
Ikke så meget alt eller intet
Mulighed for GUI
Bedre økonomi

Spørgsmål c

> Diskuter mulighederne for rene decentrale løsninger på dette område. Løsningsforslagene skal beskrive hvor hvilke data placeres og hvordan et evt. samspil mellem de decentrale anlæg kunne håndteres. I dette spørgsmål menes der med decentral en maskine per amt.

I hvert amt:
amtets egne læger (delt ud)
amtets/hele landets patienter (ofte replikeret data, der opdateres ofte)
behandlingskoder (replikeret data, der ikke opdateres så ofte)
lægeregninger fra amtets læger (delt ud)
amts-tabel (replikeret data, ændres meget sjældent)

Spørgsmål d

> Beskriv en eller flere muligheder for distribuerede løsninger. Løsningerne skal igen indeholde angivelse af hvilke data, der placeres hvor. Løsningens konsekvenser skal angives.

Distribueret løsning:
Placer alle landets patienter på en fælles server!
Evt. amtets egne patienter lokalt.

4.1.3. Valg af driftsmodel

Ofte kan reelt valg ved eget udviklet software. De fleste virksomheder vælger overordnet driftsmodeller.

4.1.3.1. Funktions-data-skema

Godt til at give overblik over et større system. Skemaet laves ved at have Tabellerne (data) hen af og funktionerne i systemet nedad. Der krydses så af hvilke tabeller hver funktion trækker på.

Når dette er gjort kan man flytte rundt på funktionerne og de tilhørende krydser i håb om at finde et 'område' af skemaet hvor der er en høj tæthed af krydser. Et sådan område til typisk være et delsystem, der evt. vil kunne distribueres.

SOA = Service Orienteret Arkitektur. Går ud på at få systemet delt op i kasser, der så kan yde services for/til hinanden. (bruges ofte i forbindelse med MainFrame).

4.1.3.2. Valg af driftsmodel

- Krav, der påvirker driftsmodellen (næste li med ikke funktionelle krav)
  - Speciel hardware
  - Styresystem
  - GUI
  - Svartider
  - Tilgængelighed
- Forslag til maskinel og distribution
- Konsekvensberegning
  - Hvor godt opfyldes krav
  - Hvilke ressourcer skal der bruges (→ penge)

4.1.3.3. Udviklingstendenserne

Klient/Server-systemer:

> At der er udviklet en klient del og en server del.

+ GUI
+ Integration med PC-værktøjer
- Tilgængelighed (hvis serveren går ned vil man typisk hænge fast)
- Opdateringsproblemer (løst med automatisk udrulning)

Tynde klienter:

Central drift med GUI

De tynde klienter lader til at være på vej ud igen idet Klient/Server-systemerne nu har automatisk udrulning. Dog er browser-baserede systemer også meget brugt, men primært til lejlighedsvise brugere.

4.2. Netværk

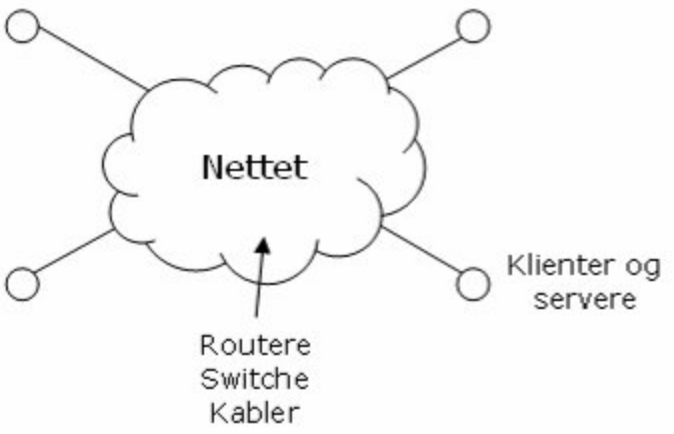
(se evt. [K&R] s. 21 figur 1.1)

Tilsluttet/i udkanten af nettet er

Værter (hosts), klienter, servere

I (midten af) nettet er

Kabler, switche, routere, trådløse enheder



Figur 1: Internettets opbygning

Typisk har de lokale enheder adgang til et LAN (Local Area Network), der så har adgang til en ISP (Internet Service Provider), der reelt er del af selve Internettet. ISP'erne har dog forbindelse til hinanden via Internettets 'backbone'.

4.2.1. Kommunikations former

4.2.1.1. Set fra "brugeren"

- forbindelses orienteret
- forbindelses løs
  - det forventes ikke at der kommer svar retur
  - (fx at sende et postkort hjem når man er på ferie)
  - bruges f.eks. når man 'browser'

4.2.1.2. Nettets opfattelse

I nettet har man to former for kommunikation (og en blandet)

kredsløbskoblet

- en uafbrudt forbindelse
- man reserverer båndbrede (x antal bit/sek.)
- hvis båndbredden ikke kan reserveres vil der blive meldt 'optaget'
- (fx som telefonforbindelser)

Pakkekoblet

- pakkerne er uafhængige
- hver pakke skal indeholde adressen på modtageren
- nettet holder ikke styr på/har ingen fornemmelse af sammenhæng mellem pakker
- (fx som postvæsnet med breve)
- benyttes på Internettet (IP)
- Quality of Service
    - noget nyt der er ved at blive implementeret på nettet, der gør det muligt at sende 'ekspres' pakker

Virtual Curcut

- Implementeres på et pakkekoblet net men ligner ud mod brugeren en kredsløbskoblet

4.2.2. Multipleksing (deling af forbindelser)

Det er ikke muligt at overtage andre forbindelsers ledige kapacitet.

4.2.2.1. Kredsløbskobling (statisk/konstant)

- Frekvens multipleksing
    - Placerer forbindelserne på hver sit frekvensområde
    - (fx radio, lysbølger, TV)
- Tidsdelt multipleksing (split modem)

Fordel: reservere båndbrede

4.2.2.2. Pakkekobling

Forbindelserne ryger i en kø og når pakkerne kommer frem bliver de fordelt efter den 'adresse' der står som modtager på pakken. Der kan dog kun sende én pakke af gangen. Ofte kombinerer pakkekoblet multipleksing med segmentering, dvs. store pakker sendes i mindre dele.

Fordel: effektivitet

Ulempe: der kommer en 'kø-tid'.

4.2.3. Opgave 4

Denne opgave handler om at undersøge muligheder og problemer ved at kommunikere bits på et bestemt fysisk medium.

Jeres fysiske medium er lys via en lommelygte og I skal lave kommunikation af bits mellem to personer, der begge har en lommelygte.

- Hvad skal de to personer være enige om (giv et faktisk forslag til de ting de skal være enige om.)?
- Hvordan startes og afsluttes kommunikationen?

- Hvilken dataoverførselshastighed har I?
- Hvilke fejlkilder vil der være?
- Hvilke begrænsninger har jeres kommunikationsmedium?
- Kunne I give et forslag til kompression af data?

Kommunikation via lys fra lommelygter i bit (0 og 1).

1. Hvad skal man være enige om?
   Hvordan man kommunikerer hhv. et 0 og et 1 – lige som man morser, kort (½ sek. = 0) og langt (1 sek. = 1) og (1/4 sek. Pause mellem hver bit) = protokol.
   Enighed om hvordan handshaking foregår.
2. Hvordan startes og sluttes?
   Afsender lyser i 3 sekunder hvorpå modtager lyser i 3 sekunder ofr at bekræfte
3. Overførsels hastighed?
   ca. 1 bit/sek.
4. Fejlkilder?
   Afsender: trykker forkert, kan tabe lommelygten
   Modtager: 'læser' forkert, lukker øjnene, kigger væk
   Begge: pæren sprænger, batteriet løber tør, løs forbindelse i lygten, 'støj' i form af lys, blokering (fx noget placeres imellem dem), vejret
5. Hvilke begrænsninger er der?
   Rækkevidde/afstand, kræver tilstedeværelse på begge poster, tidsmæssig begrænsning, jordens krumning
6. Forslag til kompression af data?
   Øvelse bør mester (hastighed), en enighed om hvordan det kommunikeres at 'nu kommer der x antal hhv. 0'er eller 1'er'.

4.2.4. Medier

4.2.4.1. Fysiske medier

"Non-return to zero"

Bruges f.eks. ved overførsel med strøm. Et skift i stømstyrken = 0 og en vedvarende strømstyrke = 1.

Bit stuffing

Det kunne dog være svært at 'læse' mange 1-taller i træk, derfor blev der indført følgende regel:

Hver gang man har sendt fem 1-taller skal man sende et 0. Hver gang man modtager fem 1-taller fjerner man det efterfølgende 0.

Dette betød også at man kunne bruge følgende som start og slutsignal: *01111110* (et 0, seks 1-taller og et 0)

4.2.4.2. Medier generelt

Man kan ofte sende hurtigere end man ville kunne modtage. Derfor vil man ofte forsøge at øge hastigheden så meget som muligt, dog ikke mere ned at modtageren kan følge med.

Alle medier har dæmpning (dvs. et signal bliver svagere).

Modtræk = mellemstationer, der aflæser + gensender = repeater (i gamle dage).

Kobberkabler

Er billige, begrænset fysisk længde (100-500 meter), billige netkort

Typer af kobberkabler

- COAX (=antenne kabel)
  - Kerne, der er beskyttet imod indstråling
  - Bedre (længer)
- twisted pair ('snoet par')
  - Billigere (knap så langt)

Al strøm giver udstråling (magnetfelt), dermed er der et problem.

Indstråling: et magnetfelt giver "ekstra" strøm i ledningen = signalet forvrænges.

Lysleder kabler

Er hurtige, kan klare lange afstande (lille dæmpning) (op til ca. 40km), ingen ud/indstråling, dyre netkort, træls at reparere.

4.2.5. Forsinkelse

- Ekspeditionstid i routere og lignende
  - Fleste routere kører med "store-and-forward"
  - Det er væsentligt hvor mange routere man skal igennem samt hvor meget 'kø' der er
- Kø-tider
  - Sammenhængen mellem svartid og belastning
- Transmissionstid (hvor lang (i tid) en bit er)
  - Faldet de senere år
- Udbredelses hastighed (lysets hastighed (300.000 km/sek.))
  - Kan ikke umilddelbart ændres
  - Er ikke tænkt ind i f.eks. opbygningen af internettet

4.2.6. Trafiktyper

Datatrafik

- Almindelig datatrafik
    - Data skal være korrekte
    - Ventetider accepteres
  - Online (hvor brugeren sidder og venter)
  - Batch
- Realtid
  - Video etc.
    - Hellere ukorrekte data frem for ingen
  - Produktionsstyring
    - Data skal være korrekte og der skal være korte svartider

4.3. Lagdeling

4.3.1. Lagdeling som softwarearkitektur

+ Et lag kan udskiftes (så længe det nye lag har samme interface som det tidligere lag)
+ Giver overskuelighed/struktur
- Langsomt (afviklingsmæssigt)
    - Der kan være tendens til at det midterste lag i en 3-lagsmodel med database udelukkende bruges til 'gennemstilling', dog bør laget være med til brug for evt. fremtidige funktioner.
- "Lodrette" ændringer er dyre

Første net-arkitektur havde 7 lag: (OSI-modellen)

1) Applikation
2) Præsentation
3) Session
4) Transport
5) Netværk
6) Datalink
7) Fysisk

De tre øverste lag blev siden slået sammen til et lag, som det ses i TCP-modellen idet der ikke kunne findes indhold til alle 3 lag samt indholdet af de tre lag ofte hang sammen.

Udvikling til 5-lags-model: (TCP-IP)

1) Applikation
2) Transport
3) Netværk
4) Datalink
5) Fysisk

Det er kommet på tale at slå de to nederste lag sammen idet disse to stort set hænger sammen/er samlet i netkortet.

Fremtidig 4-lagsmodel:

1) Applikation
2) Transport
3) Netværk
4) MAC-lag

ATM forsøgte at gøre op med lagdelingen, men er aldrig kommet ordentligt ind. Tanken var at have få men større lag, der forøger performance. ARP-protokollen mellem netværk og datalink lagene synes især at være dum idet en maskine har to adresser – det sås mere fornuftigt at en maskine blot får en IP-adresse oppefra.

4.3.2. Protokoller

Findes på alle niveauer i et netværk, alle bruger headere.

- HTTP (internet)
- TCP (data)
- IP (routing)
- Ethernet – bruger også trailere

Message-formater: hvordan beskederne ser ud når de skal sendes

Message-flow: hvornår man må sende

4.3.2.1. Lagdelte protokoller

Første lagdelte protokol var SNA (1974) og bestod af 7 lag og var lavet udelukkende af IBM til langsomme WAN. Bruges ikke rigtig mere.

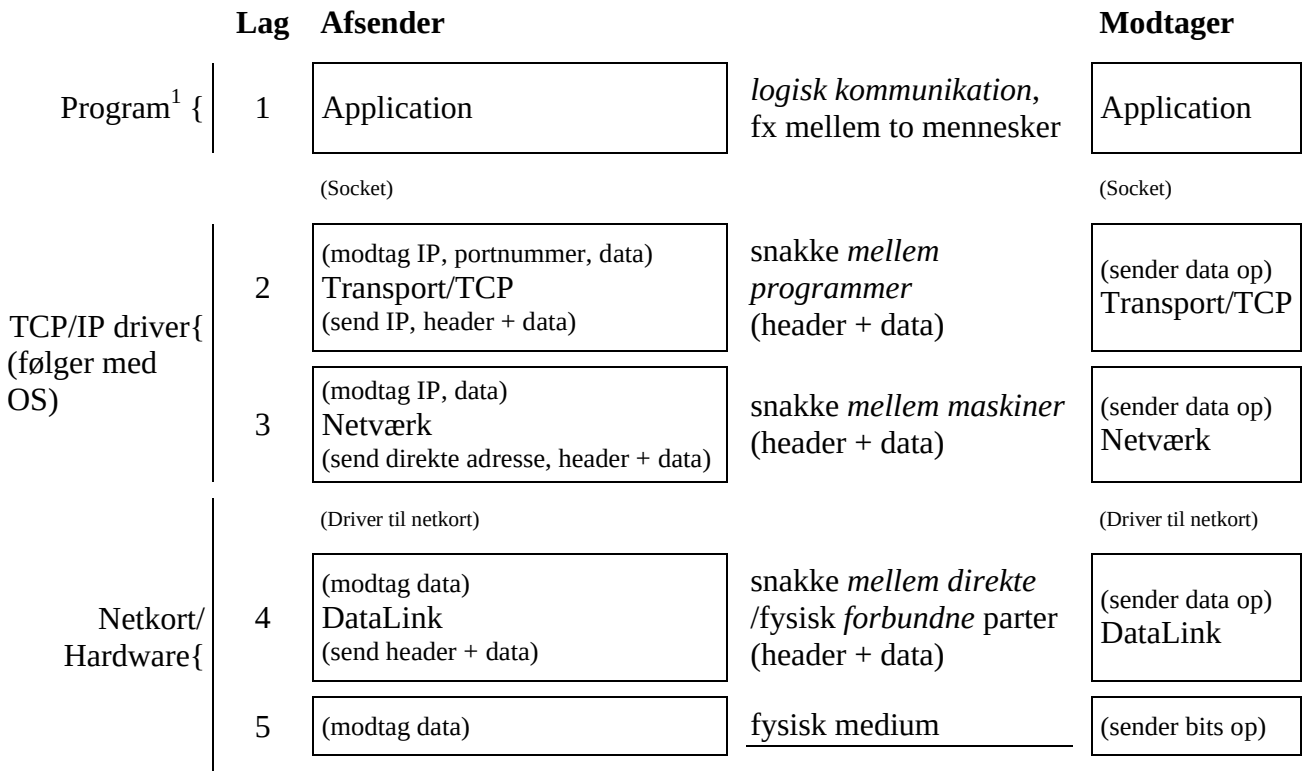
OSI (1975) bestod også på 7 lag og var lavet som standardisering til alt. Er helt død.

TCP-modellen (udviklet ved knop-skydning, ca. 80'erne) 5 lag udviklet af en del forskellige primært til Internettet. Udgangspunkt for undervisning. Svarer til OSI-modellen, dog er lag 5-7 slået sammen til lag 5. Nogle snakker om 4 lag TCP, her er det lag 1 og 2 der er smeltet sammen.

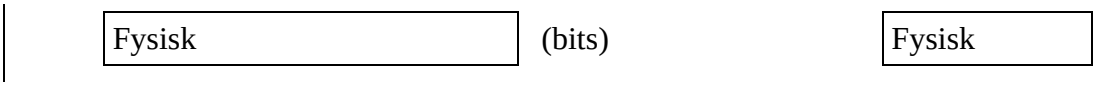
IPX/SPX (ca. 80'erne), 4 lag lavet af Novell til LAN. Godt til ren LAN, men ikke videre udbredt. Var med i Win 95 og 98 (også derfor nogle spil fra den periode har problemer med netværk), Win XP har dog mulighed for at installere den.

4.3.3. TCP lagdeling

(Følgende kan vi regne med at komme til eksamen i idet det påregnes at være med i alle spørgsmål – se repetition for forsimplet opstilling)



[1] Kan være helt selvkodet, noget selvkodet der bygger på et komponent fx HTTP eller SMTP



Headere er ment til være ens/tilsvarende til modtagersiden.

Pålidelighed: sørge for at pakker når frem. Dvs. sender pakken igen hvis det ikke bliver bekræftet at den er nået frem.

Flow-kontrol: sikre at en modtager ikke oversvømmes af data.

Congestion-kontrol: undgå trafikpropper

Fejl-kontrol: check om bit er blevet "vendt" (kan gøres med CSE)

4.3.3.1. Lag 1: Applikationslaget

Kendte protokoller: HTTP, FTP, SMTP, (DNS)

Standard protokoller: fx til lagring af data

Egne protokoller (til et system): Talk05V

Protokollerne ligger oven på socket, der ligger lige under applikationslaget

4.3.3.2. Lag 2: Transportlaget

TCP og UDP (discount udgave, kun port numre)

- Port numre
- Kan sikre pålidelighed (kun TCP)
- Congestion-kontrol (valgfrit) (kun TCP)
    - Prøver dynamisk at sende mere og mere, når det så fornemmes at det er ved at gå galt, sænker den farten.
- Flow-kontrol (valgfrit) (kun TCP)
    - På samme måde som ovenstående
- Kan være forbindelsesorienteret (kun TCP)
    - Opretter en session. Den er dog kun kendt på dette lag og ikke resten af vejen gennem nettet.
- Opdeling i segmenter (kun TCP)
    - Store pakker deles automatisk op i mindre pakker og samles igen i det tilsvarende lag hos modtageren.

UDP er hurtigere end TCP idet den ikke gør brug af handshaking. UDP bruges ofte til streaming.

4.3.3.3. Lag 3: Netværkslaget

Rutning (sende pakker af de rigtige veje/ruter) via en Router

Kan fragmenterer (dele store pakker op i mindre)

4.3.3.4. Lag 4: Datalink-laget

- Kan laves pålideligt
- Flow-kontrol
- Fejl-kontrol
- Tilgang til mediet (sørger for ikke at sende samtidig med andre, i det tilfælde at mediet ikke kan klare dette, fx halfduplex eller ethernet)

(Det er på dette lag en switch hører til. En switch arbejder på mac-adresser (og kender så at sige alle mac-adresser den er forbundet til), dvs. den kun kan bruges på et lokalnetværk)

4.3.3.5. Lag 5: Fysiske lag

Sender bits (0 og 1-taller).

4.3.4. Opgave 5

Denne opgave handler om at undersøge en bestemt netværkstopologi kaldet hyperterningen.

ningen er defineret rekursivt således:

1. Hyperterningen af dimension 0 består af en enkelt knude.
2. Hyperterningen af dimension N fremkommer ved at tage to hyperterninger af dimension N - 1 og for hver knude i den ene af disse mindre hyperterninger at tilføje een ekstra linie, som føres til en knude i den anden mindre hyperterning, så enhver knude i den anden hyperterning bliver ramt af præcis en linie.

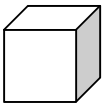
Dimension 0: en prik

Dimension 1: to prikker med en linje imellem

Dimension 2: to hold af to prikker der hver et forbundet og derefter forbundet med hinanden.

Spørgsmål a

Tegn hyperterningen af dimension 3.



Spørgsmål b

Hvor mange knuder henholdsvis linier vil der være i hyperterningen af dimension K?

Et netværks diameter defineres som den længste afstand man kan risikere at finde mellem to knuder, hvis man vælger den korteste vej.

Tal for hyperterning med dimension K:

Knuder: (K-1)knuder * 2 // 2^K

Linjer: (K-1)linjer * 2 + (K-1)knuder // K*2^(K-1)

Diameter: K // log(antal knuder)

Spørgsmål c

Hvad er diameteren for hyperterningen af dimension K?

Diameteren for en hyperterning med dimension K er K

Spørgsmål d

Beregn antal knuder, antal linier og diameter for hyperterningen af dimension 10.

Dimension	knuder	linjer	diameter
0	1	0	0
1	2	1	1
2	4	4	2
3	8	12	3
4	16	32	4

5	32	80	5
6	64	192	6
7	128	448	7
8	256	1024	8
9	512	2304	9
10	**1024**	**5120**	**10**

4.4. Applikationslag

4.4.1. HTTP

Historie

Statiske hjemmesider uden formatering (rå tekst)

→ Formatering vis HTML

→ Programmer der genererer hjemmesider (1 gang per time/døgn)

→ Dynamiske hjemmesider (CGI-filer)

→ Dynamiske hjemmesider (script-sprog (ASP, PHP, JSP))

Princippet i en stateless-server (http-server) er at den glemmer alt fra forespørgsel til forespørgsel.

4.4.1.1. HTTP-protokol

Består af to dele, browser og webserver. Kommunikation er request→respons baseret.

- Web-server er stateless (dvs. den husker intet om en klient fra gang til gang)
  - Cookie-begrebet er lavet for at lappe lidt på dette
  - I script-sprog er der ligeledes Session-objekter
- I starten bruges (HTTP 1.0) ikke-vedvarende forbindelser i TCP (ikke-persistent)
- Senere (HTTP 1.1) kom muligheden for vedvarende forbindelser i TCP (persistent)
- Bruger port 80
- Headere er i ren tekst

Senere udvidelser

- Autorisation
  - ”Primitiv logon”
    - Browser → request
    - Web-server → auth. required
    - Browser → request + bruger & pass
- Cookie
  - En fil (typisk et tal) der gemmes hos klienten
    - Browser → request
    - Web-server → respons + cookie
    - Browser → request + cookie
  - Session bygger også på cookies

- Caching
    - Man kan i Proxy-servere lave caching, dvs. indholdet af web-sider gemmes.
        - Gør det muligt at lave en request der hedder 'er denne side opdateret siden sidst jeg fik den' (baseres på Last-Modified).

4.4.1.2. HTML/XML

HTML er godt til at kommunikere mellem en bruger og en maskine. Det kan dog også være rart hvis man kan lave kommunikation fra maskine til maskine, det er her XML kommer ind i billedet.

Forskellen er at HTML har formateringsoplysninger blandet sammen med data. XML har udelukkende strukturoplysninger.

4.4.2. FTP

Udviklet til overførsel af filer. Typisk kræves der brugerid og password.

- Baseret på TCP
- Laver to forbindelser
    - Kontrolforbindelse
    - Dataforbindelse (en per fil)
- Bruger port 21
- Husker tilstand om brugeren

4.4.3. SMTP

Bruger agent → (SMTP) → Mail-server → (SMTP) → Mail-server → (???) → Bruger agent

Der bruges kun SMTP mellem afsender og afsenders mail-server hvis der bruges et mail-program som f.eks. Outlook, ellers bruges der f.eks. HTTP.

Mellem modtagerens mailserver og brugeren selv afhænger protokollen af hvordan brugeren kigger på sin mail, det være sig POP3 (udelukkende en indbakke, sletter mails når de er afhentet), IMAP (har mappestrukturer), HTTP (alle afhentnings/pull protokoller).

- Fra 1982
- Baseret på 7 bits ASCII (MIME[2]-encoding)
- Ovenpå TCP (direkte forbindelser mellem mail-servere)
- Bruger port 25
- Push-protokol (sender data frem uden der er lavet en request fra modtageren)
- Asynkron service (sender og modtager behøver ikke være tilstede samtidig)

4.4.4. DNS

Veksler navne til IP-adresser.

- Bruger UDP
- Bruger port 53
- Distribueret oplysningsservice

[2] Multipurpose Internet Mail Extention

Support for alias (dvs. flere navne til samme IP)

4.4.4.1. DNS-server opbygning

- Lokale DNS-servere
  - Kender lokale adresser
  - Cache
- Root-servere
  - Kender flere navne
  - Kender andre autoritative servere
  - Cache
- Autoritative-servere
  - Kender typisk er område (fx ".dk")

4.5. Socket-programmering

En tynd skal ovenpå enten UDP eller TCP, som forbinder dette og applikationen.

Definitioner:

- Klient ← den der tager initiativ (den, der ringer op)
  - Klient skal kende servers domænenavn eller IP-adresse samt port nummer
- Server ← den, der bliver kontaktet (den, der ringes op)
  - Server skal være startet op først

Socket = den ene ende af en forbindelse

For TCP, se [K&R] s. 139 og s. 142. For UDP, se [K&R] s. 144 og s. 148.

I TCP åbnes der en socket per forbindelse (dvs. pr klient) hvilket betyder at flere sockets kan bruge samme portnummer, sockets identificeres så ud fra IP og port på både server og klient.

I UDP åbnes der en socket pr portnummer. Der er ikke behov for at identificerer 'hvem' der har forespurgt andet end når svaret skal sendes retur.

4.6. Transportlaget

4.6.1.1. Portnumre

2 bytes (dvs. 16 bit eller op til ca. 65.000)

De laveste 1024 var reserverede, men grænsen er i dag reelt højere.

Normalt er kun server-portnummer interessant.

4.6.1.2. UDP

Discount transportlag. Simpelt, nemt og hurtigt.

To faciliteter

- portnumre
- simpelt check for transmissionsfejl

4.6.1.3. Introduktion til sliding window

Pålidelighed er hovedformålet. Med i mekanismen er også flow- og congestion-kontrol.

Ide

Tager en kopi af pakken inden den sendes og starter en timer (til time-out). (Time-out værdier skal være større end man tror)

Når man sender en pakke får man en 'ACK'/acknowledgement tilbage når den er modtaget. En ny pakke sendes først når man har fået ACK.

Dette kaldes 'Stop & vent' og er forholdsvis ineffektivt.

En løsning kan være at må sende flere pakker af sted og først få ACK senere. En gruppe pakker afsendes indenfor hvad man kan kalde et window.

4.6.2. Fejlkilder og pålidelighed

Protekoller til pålidelighed = stabilitet (at være sikker på at pakker når korrekt frem)

Der er to fejlkilder:

1. pakker går tabt
2. pakken får en bit-fejl

4.6.2.1. Den første løsning (løse bit-fejl)

Afsender gemmer en kopi af pakken.

Afsender → pakke → modtager

Afsender →← ACK / NACK ← modtager

Afsender → pakke (igen) → modtager

Afsender ← ACK / NACK ← modtager

4.6.2.2. Den anden løsning (løser bit-fejl og tabte pakker)

Pakker skal udstyres med et sekvensnummer så modtager kan se hvad der er gensendelser (grundet tabt/forvansket ACK/NACK)

Afsender → pakke + sekvensnummer → modtager

Afsender →← ACK + sekvensnummer ← modtager

Afsender → pakke + sekvensnummer (igen) → modtager

Afsender ← ACK + sekvensnummer (igen) ← modtager

Afsender beholder også her en kopi af data til ACK er modtaget.

Pakker med bit-fejl behandles på samme måde som tabte pakker, dvs. der intet svares hvorved der vil opstå en time-out hos afsenderen.

Ovenstående kaldes "Stop og vent" hvilket giver en meget dårlig performance idet der er for meget venten.

4.6.2.3. Den tredje løsning: Sliding Window (løser bit-fejl og tabte pakker)

Ide: Afsender må sende op til X pakker inden der ventes på ACK. X kaldes sendevinduet.

Der er to udgaver af sliding window: "go back n" og "selektiv repeat".

Go back n

- Hvis modtager ser pakker ude af nummerorden smides de væk. Evt. gensendes ACK på den seneste modtagne pakke
- Uden NACK

- Ofte kumulative ACK (fx betyder ACK 17 at 15 og 16 er modtaget)
- Nemt at kode modtagersiden
- Der opstår nemt spild hvis fx en af de første pakker går tabt smides resten væk

Selektiv repeat

- Modtager gemmer pakker ude af nummerorden
- ACK sendes enkeltvis for pakkerne
- Mere besværligt at kode modtagersiden
- Mere besparende ved gensendelser

Fælles

Et problem i alle sliding window er gamle pakker (især hvis sekvensnumrene er valgt indenfor et snævert interval eller hvis de bliver genanvendt for hurtigt).

Hvis man "genbruger" sekvensnumre meget hurtigt kan man tro at det er en ny pakke.

"Gamle pakker" opstår hvis en pakke hænger fast i en router hvor der er kø (og for meget ram!) og så bliver sendt videre når køen opløses. Hvis der er sket time-out i mellemtiden så er pakken blevet 'gammel'

4.6.3. TCP

Se [K&R] s. 217 (figur 3.29) for struktur.

- Sessionsopsætning via 3-way handshake
- Fuld duplex
- Altid punkt til punkt (dvs. ikke et broadcast medium)
- Ingen NACK
- ACK kan sendes sammen med en ny gruppe af data ("piggybag"), eller alene
- Kumulativ ACK
- Portnumre
- Pålidelig (stabil) (sliding window)
- Finder selv time-out grænse ud fra målt RTT
- Flow-kontrol (tilladte mængde bytes beregnes ud fra header)
- Segmentering (bruger numre på bytes)

Sekvensnummer = nummeret på 1. byte i pakken (i forhold til forbindelsen)

Bekræftelsesnummer = nummeret på den næste byte, der forventes (bekræftelse på at alle bytes op til bkr.nr-1)

Ingen regler i TCP om hvad der sker med pakker der er ude af nummerorden

Estimering af time-out værdi

RTT = gennemsnitlig svartid på nettet

(Gennemsnit) Est_{RTT} = 7/8 * Est_{RTT} + 1/8 * nyeste svartid

(Varians) Dev_{RTT} = (1-0,25) * Dev_{RTT} + 0,25 * (Nyeste svartid – Est_{RTT})

Time-out = Est_{RTT} + 4 * Dev_{RTT}

Ved første gensendelse bruges 2* time-out, anden gang bruges 4* timeout

4.6.3.1. Flow-kontrol

Modtager skriver i "Modtagervindue"-feltet hvor mange bytes denne har plads til lige nu.

Hvis tallet er 0 sender man 1 byte af gangen indtil tallet stiger. Dette gøres for at se hvornår der er plads og for ikke at sende en masse til en applikation der ikke tager det til sig.

4.6.3.2. Congestion-kontrol (ikke del af pensum)

Se [K&R] s. 246 (figur 3.50)

Afsender presser så meget af sted af gangen. Når man opnår time-out halveres afsendelses frekvensen.

4.6.4. Navneserver protokol til workshop (Talk05V)

Udarbejd I fællesskab en protokol for kommunikation til brug for et simpelt chat-program

IP-adressen til denne må gerne hardcodes.

Skal kunne huske navn og en IP-adresse.

Funktioner: Log ind, modtage liste med online, log af.

Tilmeld:

"#Registrer navn" (fra klient til navneserver)

"#OK …" (fra navneserver til klient)

"#NOTOK …" (fra navneserver til klient)

Afmeld:

"#Afmeld" (fra klient til navneserver

Liste:

"#Liste" (fra klient til navneserver)

"#Listesvar#navn1,ip1#navn2,ip2 (fra navneserver til klient)

- Alt afsluttes med "\n"
- TCP
- Port nummer: 23457

4.7. Netværkslaget

4.7.1.1. Netværkslagets opgaver

- Routning (at finde vej)
- Fragmenterer (dele op i mindre stykker)
    - Udgås ofte idet transportlaget også segmenterer filerne, denne sørger normalt for at stykkerne er små nok til at kunne komme direkte med datalink laget.

4.7.1.2. Netværkslagets servicemodel

- Kredsløbskoblet (fysisk forbindelse gennem hele netværket)
  Dyr og udnytter ikke nettet særlig godt.
- Pakkekoblet
    - Virtual Curcuit
      Der oprettes en rute ved opstart. Alle efterfølgende pakker følger ruten.

- Datagram (IP)
  Hver pakke løber uafhængigt af hinanden. Lover intet om hvorvidt eller hvornår en pakke når frem.

4.7.2. Routning

En router behøver ikke kende hele vejen for en pakke, bare den ved hvor den selv skal sende pakken.

Hvad er en god rute

- korteste
- hurtigste
- billigst

Skal valg af rute være afhængig af

- nedbrud af linier
- aktuel belastning

4.7.2.1. Global rutningsalgoritme

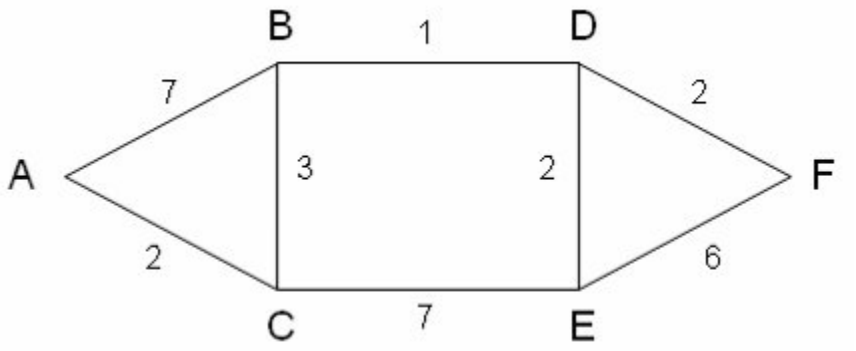
(kræver hele nettet er kendt)

Decentral rutning (modtager informationer om veje fra naboer)

4.7.2.2. Dijkstras find korteste vej (global)

- Kender afstande og veje i hele nettet
- Finder korteste vej til alle andre
- En "grådig" algoritme = når den har lagt sig fast på en rute, så vil den fortsat bruge den (nægter at skifte)

Finder den korteste vej fra en knude (router) til alle andre knuder.



Initialisering (af A)

Klaret = {A}

Knude	A	B	C	D	E	F
Dist/afstand	0	7	2	-	-	-

Indsætter alle kendte veje fra A

Gentag

Find den korteste vej fra A til en knude, som ikke er klaret

Sætter knuden ind i klaret

Sæt alle den nyes veje ind i Dist, men kun hvis de er bedre end de der er i forvejen

Klaret = {A, C}

Knude	A	B	C	D	E	F
Dist/afstand	0	5	2	-	9	-

Klaret = {A, C, B}

Knude	A	B	C	D	E	F
Dist/afstand	0	5	2	6	9	-

Klaret = {A, C, B, D}

Knude	A	B	C	D	E	F
Dist/afstand	0	5	2	6	8	8

Klaret = {A, C, B, D, E, F}

Knude	A	B	C	D	E	F
Dist/afstand	0	5	2	6	8	8

4.7.2.3. Distance vektor (decentral)

- Modtager information om ruter fra naboer
- Kender naboer og afstande til disse
- Gode rygter sendes videre, dårlige rygter gør ikke
- Hvis der nedlægges linier eller linier bliver dårligere, så kan vi få en 'kedelig' situation

Princip

Man hører om gode veje via rygter fra naboer.

Gode rygter = der er fundet en bedre vej end den der var kendt før. Gode rygter huskes og sendes videre til naboerne.

Dårlige rygter glemmes bare og sendes ikke videre.

Eksempel – set fra E

[K&R] s. 283 figur 4.5.

		Direkte forbundet til/ omkostninger til modtager via		
	$D^E()$	A	B	D
Alle i hele nettet/ modtager	A	**1**	15 / 14	5
	B	8 / 7	8	**5**
	C	9 / 6	9	**4**
	D	4	11	**2**

Initialisering

Fortæl alle naboer om de direkte forbindelser du selv har.

Undervejs

Hver gang jeg modtager en besked opdateres skemaet.

Hvis den vej, jeg får besked om, er bedre end den hidtil bedste, så informerer jeg alle naboer.

Hvis der kommer nye veje til eller eksisterende veje bliver 'billigere'

→ så send bare det gode rygte videre

Hvis der forsvinder veje eller eksisterende veje bliver dyrere

[K&R] s. 285 figur 4.7.

Hvis der *slettes* en vej opfordres knuderne til at:

→ sige knuden ikke længere har forbindelse til pågældende knude – heller ikke gennem andre knuder.

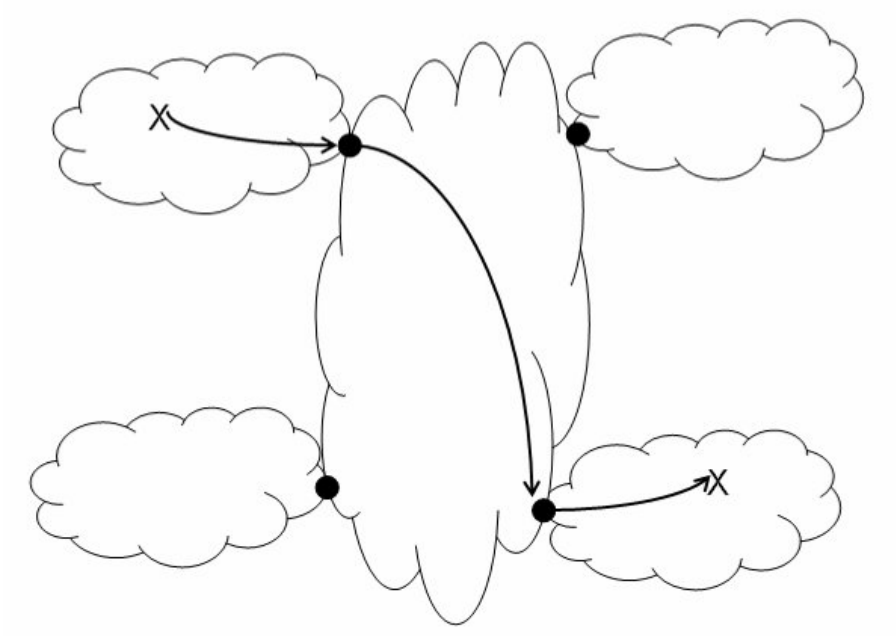
→ initialisere knuden igen

Hvis forbindelsen Y-X *opjusteres* fortæller Y til Z at der er en ny 'korteste vej'.

→ Korteste vej hos Z til X gennem Y opdateres så med Y's korteste vej + afstanden mellem Y og Z.

→ Opfordres til at 'slette' ruten og oprette den nye.

4.7.2.4. Hierarkisk rutning



Hver rutning kan have forskellige rutningsmetoder.

4.7.3. IP

Opgaver

Rutning (hierarkiske adresser)

Fragmentering (opdeling i mindre pakker)

Service

Datagrambaseret

Ingen leveringsgaranti

4.7.3.1. Adresser

195.47.14.98 alle (0-255)

(4 bytes / 32 bit)

Opdeling i delnet (fx lokalnet + netkort i router)

4.7.3.2. IP-adresser

Klasse A net

Store organisationer

0 [7 bit = nettet] [24 bit = host]

128 net

Klasse B net

Middelstore organisationer

10 [14 bit = nettet] [16 bit = host]

Ca. 16.000 klasse

Ca. 65.000 hosts

Klasse C net

Små organisationer

110 [21 bit = nettet] [8 bit host]

Ca. 2 mill net

Ca. 256 hosts

Klasse opdeling opgives I 1996. Der indføres i stedet dynamiske IP-adresser.

Fx 195.14.17/24 betyder at nettet er identificeret er de første 24 bit og de efterfølgende er værten/host

4.7.3.3. Afsendelse

TCP

TCP-pakke + IP-adresse

Netværk (skal vide: hvilke IP-adresser er på mit eget net (subnet-mask))

IP-pakke + første omgangs modtager (ved samme subnetmask =TCP IP-adresse ellers = gateway/router)

Datalink (ARB-protokol bruges til at veksle mellem MAC og IP)

4.7.3.4. DHCP

- Uddeler IP-adresser dynamisk (på et lokalnet)
- Sparer på IP-adresser
    - En god ting da mængden af IP-adresser er begrænsede
- Let at konfigurere
- Egner sig ikke til server
    - Der er dog nu en mulighed for at låse en IP-adresse til en MAC-adresse
- Når en ny maskine kommer på udsender den en UDP broadcast hvorpå DHCP svarer

4.7.3.5. NAT (Network Address Translation)

Grund: Mangel på IP-adresser

NAT bygges på klienter, ikke servere. Dvs. det bruges kun på udgående (så svaret kommer rigtigt retur).

Det er muligt at køre en server ved siden af NAT, men den har en anderledes type IP end de der er bag NAT.

IP-adressen 10.xxx.xxx.xxx er reserveret til NAT og bruges ikke på Internettet, tilsvarende er 192.168.xxx.xxx & 127.x.x.x reserveret.

4.8. Datalink-laget

Opgaver

- Inddeling i frames
- Kontrol af transmissionsfejl
- Pålidelighed (stabilitet) og flowkontrol ← sliding window
  - Pakker kan ikke overhale hinanden
- Medie Access Control (MAC)

4.8.1.1. Kontrol af transmissionsfejl

Afsender beregner ud fra data nogle check-bits.

Data	Check-bits

Herefter sendes data og check-bits til modtager, der laver samme beregning

Hvis de beregnede check-bits svarer til de modtagne check-bits er det ok

Den letteste teknik er simpelparitet.

Lige paritet (at der altid er et lige antal 1-taller)

1	0	0	1	0	1	0	0	1 (check)

Opdager hvis 1 bit er forkert (eller et hvilket som helst antal ulige fejl)

Det simpleste tilfælde, der ikke opdages er 2 bit vendt.

Bedre teknik er CRC

Ved CRC-16 (dvs. 16 check-bits) er den mindste uopdagede fejl er 4 bits vendt ikke for tæt på hinanden.

Findes også som 32 bit udgave.

4.8.2. Error-detektion

- paritet
- CRC

Modtager opdager evt. fejl og smider pakken væk eller sender NACK.

Alle metoder har en chance/risiko for at der er fejl, der ikke opdages.

4.8.3. Error-correction

Modtager opdager fejl og retter dem selv.

4.8.3.1. Hamming-kodning

Der er check-bits i alle positioner 2^x hvor x = {0, 1, 2, 3...}. Alle andre er databits.

Eksempel

Jeg vil sende 0110101 (der bruges lige paritet, dvs. lige antal 1-taller)

1	2	3	4	5	6	7	8	9	10	11
1	*0*	0	*0*	1	1	0	*0*	1	0	1
1	2	2 1	4	4 1	4 2	4 2 1	8	8 1	8 2	8 2 1

Alle de steder hvor der er ex. 8 eller 4 eller 2 eller 1 sammenlignes og det der 'mangler' for at få pariteten til at passe sættes ind i check-bits (her 1, 2, 4 og 8)

Hvis <u>1 bit</u> er forkert, kan modtageren se hvilken

Vi vender nu en bit og ser på hvor simpelt det er at rette fejlen.

1	2	3	4	5	6	7	8	9	10	11
1	*0*	0	*0*	1	0	0	*0*	1	0	1
1	2	2 1	4	4 1	4 2	4 2 1	8	8 1	8 2	8 2 1

Fejl på 4 og 2, imens 8 og 1 er korrekt. *4 + 2 = 6* ergo er der fejl på 6.

4.8.3.2. Opgave7

Denne opgave handler om Hamming-kodning til fejlkorrektion. Du skal antage at der anvendes lige paritet.

Spørgsmål a

Lav rammen, der sendes, hvis man ønsker databittene 11010001 sendt.

Ønsker at sende 1 101 0001 (lige paritet)

1	2	3	4	5	6	7	8	9	10	11	12
1	*0*	1	*1*	1	0	1	*1*	0	0	0	1
1	2	2 1	4	4 1	4 2	4 2 1	8	8 1	8 2	8 2 1	8 4

Sendes som 10 11 10 11 00 01

Spørgsmål b

En modtager har modtaget følgende rammer

010111110100

111011000101

Hvad er de korrekte data?

Modtaget er rammerne 01 01 11 11 01 00 og 11 10 11 00 01 01

01 01 11 11 01 00

1	2	3	4	5	6	7	8	9	10	11	12
0	*1*	0	*1*	1	1	1	*1*	0	1	0	0

1	2	2 1	4	4 1	4 2	4 2 1	8	8 1	8 2	8 2 1	8 4

Korrekt: 1, 2, 4, 8.

Fejl: ingen.

11 10 11 00 01 01

1	2	3	4	5	6	7	8	9	10	11	12
1	*1*	1	*0*	1	1	0	*0*	0	1	0	1
1	2	2 1	4	4 1	4 2	4 2 1	8	8 1	8 2	8 2 1	8 4

Korrekt: 2, 8.

Fejl: 1, 4.

Fejl i felt 5! Korrekt data er: 11 10 01 00 01 01

Da correction ikke er særlig fejltolerant bruger langt de fleste detection.

4.8.4. Medie Access Control

Relevant hvor flere deler samme medie

- broadcast medier (svarer til at råbe ud over en flok mennesker)
  - lokalnet og trådløse
- half-duplex (svarer til brug af en walkie-talkie)
  - visse punkt til punkt

Løsninger

- opdeling af kanalen
  - tids eller frekvens multipleksing
- tilfældig adgang
  - gør intet ved problemet, håber at det ikke går alt for galt
  - svarer til et møde uden ordstyrer
- turtagning
  - at man skiftes til at sige noget efter en eller anden algoritme

4.8.5. Tilfældig adgang

Er en simpel algoritme

Filosofi: kollisioner vil opstå, men så må det løses til den tid.

1. bud: Opdelt ALOHA (ca. 70'erne)

   Man havde tidsintervaller svarende til en pakke.

   Man måtte kun starte med at sende ved start af et interval (hvis der opstod kollisioner ville de opstå med det samme).

   Problemet var at holde alle netkort opdateret med hvornår et interval startede.

2. bud: Rent ALOHA (ca. 70'erne)

Send når du har lyst.

3. bud: CSMA/CD (Carrier Sense Multiple Access with Collision Detection)

- Man lytter om der er ledigt inden man sender
- Afsender lytter med om der sker kollisioner

4.8.5.1. Tur baseret protokol

LAN → Token Ring (det er kun den med 'stafetten' der må sige noget)

Algoritmisk meget svært.

Der skal sørges for at der kun er én token der går rundt og at den ikke går tabt samtidig med at maskinerne skal kunne 'meldes' ind og ud af ringen.

Ulempen ved større netværker er at der opstår en forsinkelse alene ved at token passerer rundt.

Half duplex → Polling

Den ene ende 'primær ende' bliver ved med at spørge den anden ende 'har du noget at sende', 'har du noget at sende'.

4.8.5.2. MAC-adresser og ARP

Enhver maskine har tre navne

- Domænenavn
  - Oversættes med DNS
- IP
  - Oversættes med ARP
- MAC

MAC-adresse

Er et id-nummer på netkortet (6 bytes).

Normalt er MAC-adressen hardkodet på netkortet (entydigt i hele verden[3]), der findes dog nogle hvor den kan softkodes.

Filosofien er at netkortene kan frasortere pakker der ikke er til dem selv med det samme. På den måde er netkortet fri for at vække OS for at checke om en pakke er til den pågældende maskine.

ARP

Ikke serverbaseret protokol.

I hver maskine er der en ARP-tabel

MAC	IP

`ipconfig → IP configuration`
`arp –a → liste over dem man har haft kommunikation med for nylig`

[3] Principielt er adresserne delt op imellem producenterne. I praksis er der dog risiko for at få netkort med overlappende MAC-adresser. Softkode kort kan også overlappe hinanden eller andre netkort. Ens MAC-adresser har kun indflydelse indenfor at lokalnetværk.

```
ping [PC-navn] → om der er forbindelse til pågældende PC
eller IP
```

Hvis man spørg på en IP, der ikke er i ARP-tabellen broadcastes IP og rette maskine svarer. Resultat indsættes i ARP-tabellen.

4.8.6. Ethernet (IEEE 802.3)

- baseret på CSMA/CD[4]
- dækker både fysisk og datalink lag
- implementeret på netkort

Ethernet rammen

- præambel (8 bytes) (en form for forvarsel for at der kommer nogle data)
- modtager MAC (6 bytes)
- afsender MAC (6 bytes)
- type (2 bytes) (fx IP, IPX, AppleTalk)
- data (46-1500 bytes)
- CRC-32 check (4 bytes)

Service

- upålidelig
- forbindelsesløs
- fejlkontrol (fejlpakker smides væk)
- Manchester kodning ([K&R] s. 411 figur 5.24)
- Yderligere regler
    - Hvis der opstår kollision sendes et JAM signal. Derefter ventes random tid, inden der sendes igen.
        - Opstår der igen JAM vælges ventetiden inden for den dobbelte periode igen
    - Hvorfor minimum på pakkestørrelse (og maksimum på kabellængde)
        - Kabeltyper
            - 10 Base 2 → 10 = hastighed, 2 = antal hundrede meter uden forstærkninger
                - Gammeldags net hvor ledningerne ligger bag maskinerne og er forbundet med T-stykker → broadcast
                - Gammel/ustruktureret kabling
            - 100 Base T → 100 = hastighed, T = twisted pair kabler
                - Alle maskinerne er forbundet med en HUB (dvs. alt blev videresendt til alle) → broadcast
                - Struktureret kabling

[4] Carier Sence Mulible Access/Colission Detection

4.9. Sammenkobling af net

- Repeater → Fysisk lag
- HUB → Fysisk lag
- Bridge → Datalink
- Switch → Datalink
- Router → Netværk

4.9.1.1. Repeater og HUB

- Sender signalet videre på alle udgange
- Signal aflæses og gensendes
- Ingen buffer
- Tager ingen hensyn til kollisioner

4.9.1.2. Bridge

- Videresender kun den trafik, der skal videre (checkes på MAC-adresser)
- Har buffer
- Undersøger om der er optaget inden der sendes videre
- Har en videresendelsestabel (MAC og retning [højre/venstre])

Selvlærende broer bygger videresendelsestabellen op ved at se på afsenderadresser. (Den trafik der i starten kommer igennem broen uden at blive stoppet kaldes indlæringstrafik)

Eneste problem:

Hvis der er alternative veje, der giver en ring

4.9.1.3. Switch

- en nyere opfindelse
- en bro, der forbinder mange net (stik)
  - betragter hver maskine som et separat net
- selvlærende
- så snart, den har kørt i kort tid, sendes pakker kun til modtager (broadcast pakker dog til alle)
- switch kan videresende en pakke, inden hele pakken er nået frem (i modsætning til en router)

4.9.1.4. Switch vs. Router

Switch	Router
Kun i LAN (kan ikke kende alle IP i verden)	LAN og ”Internet”/WAN
Hurtig (minimal forsinkelse)	Lidt langsommere
Broadcast videresendes	Broadcast stoppes
Plug and play	Kræver opsætning
	Filtrerer trafik mere effektivt + bedre sikkerhed (dog mere ’kassen’ end selve routeren)

4.10. Synkronisering

4.10.1.1. Synkronisering på én maskine

- fælles ur
- fælles memory
- gensidig udelukkelse
    - Petersons algoritme, Hardware, Synchronized/Monitor, Semaphor

4.10.1.2. Synkronisering ved distribueret system

- hverken fælles ur eller fælles memory
- fysiske ure kan kun synkroniseres indenfor et vist interval
- i mange distribuerede systemer, der er begrebet før/efter mere interessant end den faktiske tid

4.10.1.3. Globale tilstande

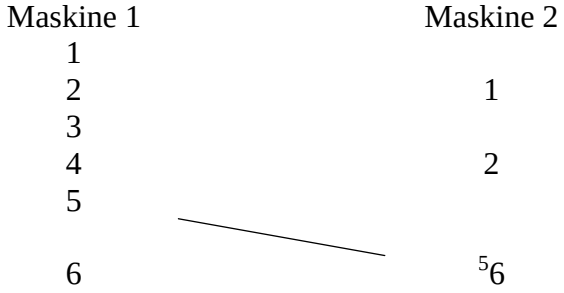
Problemet ved at lave/opdaterer en global tilstand er at der altid vil være noget på vej imellem de forskellige 'lokale tilstande'.

Stort set alle algoritmer/metoder indeholder et element af at 'standse' systemerne.

4.10.1.4. Logisk tid

- Hvis a og b er events på samme maskine og a sker før b så
    - a → b
- Hvis a er afsendelsen af en pakke og b er modtagelsen så
    - a → b
- Hvis a→ b og b → c så
    - a → c

Der kan sættes en tæller C på, så a → b betyder at C(a) < C(b)



For at sikre at to klokkeslæt ikke er ens sættes maskinens nummer ofte efter klokkeslettet (fx 2.1, 3.1, 4.1, 5.1, 6.2 og 6.1)

Eksempel

En konto-tabel der er replikeret flere steder

[5] 'Uret' tælles op til mindst 1 mere end afsendelsen af pakken. Hvis maskine 2 var længere fremme end maskine 1 ville man blot tælle en op.

To transaktioner

- sætte 100 kr. ind på Konto 5
- lægge renter (2%) til på Konto 5

I stedet for at opdaterer med det samme ligges opdateringen 'på køl' imens man sender ACK ud til de andre udgaver af konto-tabellen. Når ACK er nået rundt vil opdateringen så gå igennem.

Det er dog et krav at pakker ikke kan overhale hinanden på turen rundt. På denne måde vil de to opdateringer ske i samme rækkefølge i alle tabellerne.

4.10.2. Valg af koordinator

Typisk algoritme går ud på at vælge den maskine med det højeste nummer (fx MAC-adresse)

'Bully-algoritmen'

Når en maskine føler at der er behov for en koordinator sender den ud til andre at den har 'udråbt sig selv'.

Når der modtages besked fra en at 'jeg er højere end dig' overgives posten.

Til sidst vil koordinator-opgaven ligge hos den maskine der vitterligt har højes nummer.

Ring

Kræver at der er en logisk ring. Men ellers på samme måde som Bully.

Server (a-typisk)

Der er en bestemt maskine der altid er koordinator.

Problemet ved denne teknik er at det kræver at serveren altid er oppe.

Første på (a-typisk)

Den første maskine der kommer 'online' bliver automatisk koordinator.

4.10.3. Gensidig udelukkelse

Kritisk sektion

Noget kun én må gøre af gangen

Løsninger

- Central
- Distribueret
- Token
- (Flertal)

4.10.3.1. Central

Vælg en koordinator (kan også kaldes en server).

Koordinatoren giver lov (opretholder en kø)

Enkel algoritme.

- Sårbar (idet der kun er en koordinator, så hvis koordinator går ned mistes køen som minimum) i forhold til hvis serveren går ned.
- Hvis det er en meget brugt funktion kan serveren overbelastes

- Ikke specielt meget trafik

4.10.3.2. Distribueret algoritme

Hvis 'jeg' vil i KR[6], så spørg alle andre. "Har du noget imod, at jeg går i KR"

Hvis 'jeg' ikke selv er i KR eller vil der ind → "Ok"

Hvis jeg selv er i KR → undlader at svare og husker at 'han' spurgte

Hvis jeg ikke er i KR men gerne vil der ind → kig på timestamp

hvis den andens tidsstemple (logisk tid) < eget → "Ok"

ellers → undlader at svare og huske at han spurgte

Når man forlader KR, sendes "Ok" til dem der har spurgt.

Man må gå ind i KR, når alle andre har svaret "Ok".

- Sårbar (da en der 'hopper af' vil kunne låse hele systemet idet der så mangler et 'ok' svar) uanset hvem der går ned vil der være et problem
- Mere sårbar end Central løsning
- Væsentlig mere trafik end den centrale løsning

4.10.3.3. Token-løsning

Man placerer maskinerne i en logisk ring. (Det kan blot være en liste af IP numre, når bunden nås startes forfra)

Princippet er at et token løber rundt i ringen og den der har token har retten til KR (samme princip som token-ring)

Der skal altid være præcist et token i ringen.

Normalt er der en max tid man må holde token. (også for at kunne spore om token er gået tabt)

- Der er en del tomgangs trafik

4.10.3.4. Flertals algoritmen

Forsøg på at forbedre distribueret algoritme.

Man skal have lov af et absolut flertal.

Deltagere skal huske kun at sige ja til en.

- Hvis 3 vil i KR samtidig risikeres at ingen vil kunne få flertal
  - Deadlock (der må 'stemmes om' for at løse op)

4.10.4. Opgave 8 – repetition

Denne opgave handler om at placere nogle af de ord og begreber, som vi har omtalt i kurset, ind på de 5 lag i netværks-modellen. Det drejer sig om alle de ord og begreber, der er nævnt herunder (i kursiv). De fleste af ordene vil kun skulle placeres ud for et lag, men der er nogle stykker, som naturligt hører hjemme på flere lag. For at systematisere det hele lidt er er nogle af ordene samlet under en fælles overskrift.

Trafikkontrol *flow-control, congestion control og sliding window*

[6] KR = Kritisk Region = Kritisk Sektion = KS

Sammenkobling af net *repeater, hub, router, bridge og switch.*

Adresser *IP-adresser, MAC-adresser og portnumre.*

Protokoller *TCP, ARP, FTP, IP, SMTP, og http.*

Andre ord/begreber *Manchester-kodning , CRC-check, Fuld duplex, Tegnsæt-konvertering, Rutning, Paritetsbit, Koax-kabler, Pålidelighed, Alternative veje, Forbindelsesorienteret, Rammer (frames), Fragmentering og Segmentering.*

Lag	Trafikkontrol	Sammen-kobling af net	Adresser	Proto-koller	Andet
Applikation				FTP, SMTP, http	Fuld duplex, Tegnsæt-konvertering (ASCII-bit→EDCDIC-bit (mainframes) + retur, men kun for tekst, ikke billeder eller andet)
Transport	congestion control, flow-control, sliding window		portnumre	TCP	Pålidelighed, forbindelsesorienteret, segmentering, fuld duplex, (paritetsbit)
Netværk		Router[7], (switch)	IP-adresser	IP	Rutning, fragmentering, Alternative veje
Datalink	flow-control, sliding window	Bridge, switch	MAC-adresser	ARP	Pålidelighed, fuld duplex, rammer (frames), CRC-check, paritetsbit, forbindelsesorienteret
Fysisk		repeater, hub			Koax-kabler, fuld duplex, Manchester-kodning (bit symboliseres med skift), (Alternative veje)

4.10.5. Talk opgave

Bordtennisagtig måde at snakke på → en sende-tråd og en modtage-tråd

Separat klient-program og server-program → samles i hvert program men i hver sin tråd

Persistent contra ikke-persistent forbindelser → brug samme type i begge ender af forbindelsen

4.11. Arkitektur

Typisk hænger den logiske og fysiske arkitektur sammen.

4.11.1. Logisk arkitektur

Hvordan ens arkitektur er delt op i komponenter.

Brugergrænseflade – Model – Database

[7] Nu til dags indeholder den boks routeren er i flere og mere funktionalitet end sit eget lag. Nogle routere har fx funktionalitet der minder om en switch.

4.11.2. Fysisk arkitektur

Forbindelsen mellem de forskellige komponenter dannes ved hjælp af "Middleware".

Handler om på hvilke maskiner komponenterne fra den logiske arkitektur skal placeres (hvor de forskellige ting ligger fysisk).

Kriterier ved valg af fysisk arkitektur

Effektivitet

- Er det vigtigt at systemet kører (specielt) hurtigt
- Løsning
    - Caching/replikering af data
    - Optimering af dataoverførsel
    - Parallelitet (hvis data skal komme fra to steder, så spørg begge steder samtidig)

Tilgængelighed

- Hvor vigtigt er det, at systemet er oppe
- Løsning
    - Replikering af data
    - Program-server (fx web-servere) dubleres
    - Netværk med alternative veje
    - Undgå "single point of failure" (steder hvor et enkelt problem kan resulterer i at hele skidtet får ned)

Sikkerhed

- Se selvstændigt emne

Integration

- Behov for integration med andre systemer
- Løsning (1 af 2)
    - "Online" forbindelse til andre systemer (en ulempe er at hvis det andet system er nede kan ens eget ikke køre)
    - Kopier af andre systemers data ind i eget system

Distribution af programmel-rettelser

- Hvor svært er det at idriftsætte nye programmer
- Løsning
    - Dette kriterium peger altid mod mest muligt på servere
    - Specielt rettelser, der skal ske samtidigt på klient og server er svære (dværgfinker)

Økonomi

- Hvad koster løsninger i alt? (både hardware, software og arbejdstimer)
- Løsning
    - En 'tyk' klient skal spare penge på serversiden, til gengæld gør det opdateringer mere problematiske.

4.11.3. Arkitekturer

Arkitektur pattern er en overordnet måde at bygge et system op på.

Det mest kendte er lagdeling. (Der findes andre, men de bruges sjældent)

Der er rigtig mange afskygninger af lagdeling.

Meget kendt er 3-lagsmodellen (der findes flere forskellige typer)

En klassisk 3-lagsmodel: (har været ved at smelte ned til 2 (midterste lag fik for lidt at lave) kan udvides til 4 lag ved at tilføje database)

* Præsentation
* Logik/Funktioner
* Data

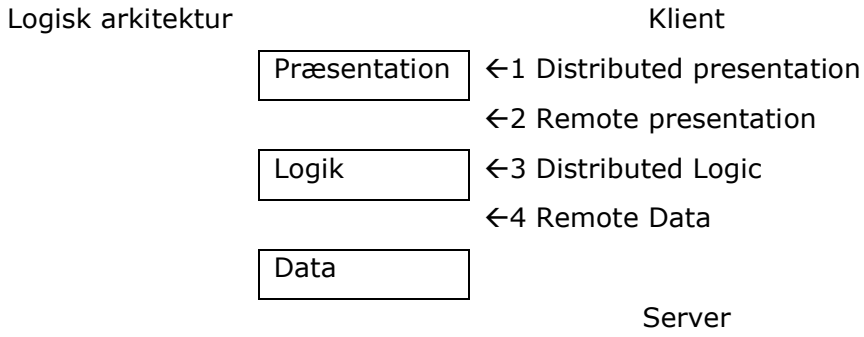
En anderledes 3-lagsmodel:

* Præsentation
* Model
* Database

Typisk datamatiker 3-lagsmodel:

* Præsentation
* Controller
* Model

4.11.3.1. Klassificering af Client/Server løsninger



Remote Procedure Call

- RMI (kun Java)
- Remoting (.NET) ← i et vist omfang sproguafhængigt (baseret på COM+, der er en underliggende teknologi som Microsoft har lavet, COM+ er sproguafhængig)
- CORBA (generelt, sproguafhængigt = forsøg på en standard) ← var tungt, er på vej ud
- Web-services (generelt, sproguafhængigt) ← baserer sig på XML dokumenter, for tungt for tiden og aldeles ikke egnet til internt brug

Konfiguration på serversiden

- Programservere (applikationsserver, Web-server)
- Data-servere (DB-server, filserver)
- Kombination af de to (Model-server)

4.11.3.2. Labyrint-spil (Arnold-spillet)

TCP contra UDP

UDP = hurtig, men fare for pakke tab

TCP = langsom, men sikker overførsel

Broardcast kun muligt med UDP.

Overordnede strategier

- Server
  - Dedikeret server (ikke selv spiller)
  - Fast server (selv spiller)
  - Koordinator (flytbart)
- Token
- Distribueret

Hvor meget skal sendes?

- kun ændringer, er hurtigst
- hele stillingen sendes rundt

4.12. Databaser

- Centrale
  - Program og database på samme maskine
- Remote
  - Program og database er på hver sin maskine
- Distribueret
  - Data er placeret på flere databaseservere (når et program bruger flere databaser)

4.12.1.1. Klassificering af distribuerede databaser

- Lav/høj transparens
  - Lav: Programmøren laver arbejdet
  - Høj: Databasen laver arbejdet (kræver at denne har faciliteter der understøtter det)
- Homogen contra heterogen
  - Hvor ens er de to (eller flere) databaser
- Top-down versus bottom-up
  - Top-Down: nyudvikling (der er ingen systemer i forvejen, de laves samtidig med at databaserne ordnes → betyder at databaserne typisk er ens samt at tabeller m.v. ser ens ud)
  - Bottom-Up: integration af eksisterende systemet (systemerne er der i forvejen, de skal blot samles → motivationen kommer fra data)

4.12.2. Distribueret database

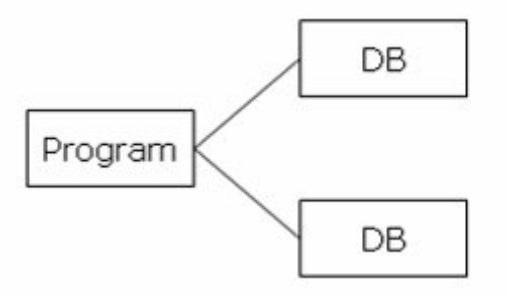
Definition

- Data på flere databaseservere
- Logisk integration i data (der er programmer, der bruger data flere steder fra)

4.12.2.1. Transparens

Lav transparens

Programmerne styrer distributionen. Kræver ikke faciliteter i DB-server.



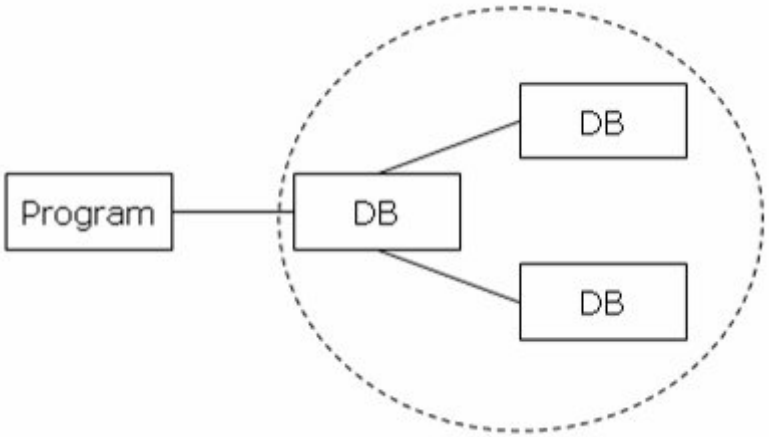
Meget vedligeholdelseskrævende.

Høj transparens

DB styrer distributionen.

Globalt skema = de tabeller der er i hele den distribuerede database.

Programmør laver SQL ud fra globalt skema, DB-server sørger for at finde data. Kræver faciliteter i DB-server.

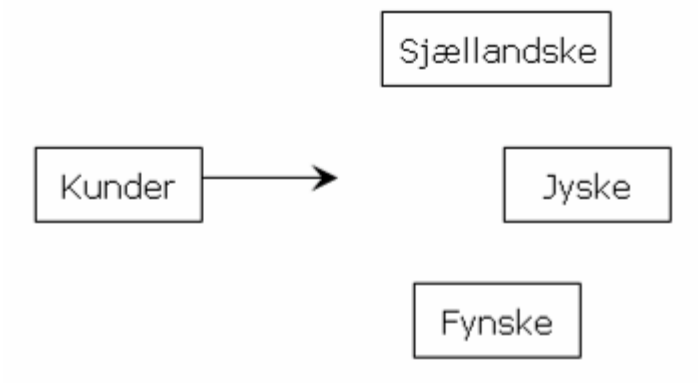


4.12.2.2. Top-down & Bottom-up

Top-down

Ny udvikling.

- Analyse
- Design
  - Klassediagram → Tabeller = globale tabeller
  - Fragmentering



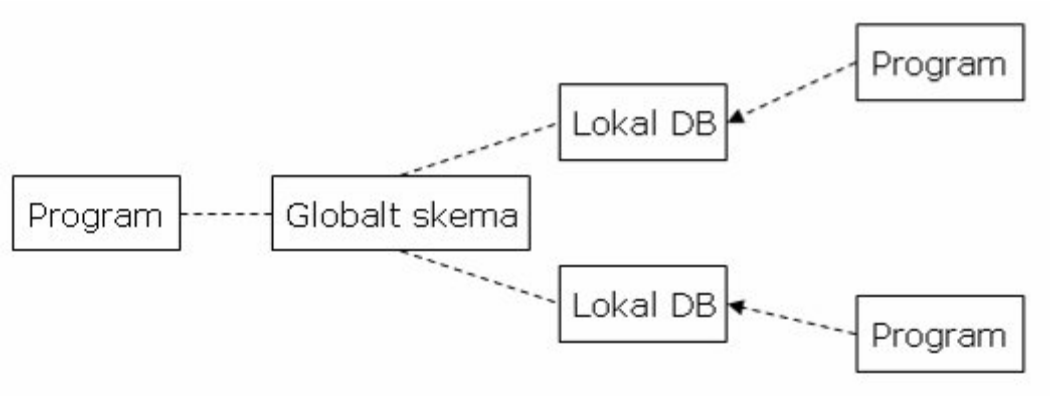
o

- Allokering
    - Beslut hvor fragmenter/tabeller skal placeres. Det er også under allokering, det besluttes om nogle data skal replikeres

Bruges ikke så ofte i virksomheder, idet det findes for besværligt. Til gengæld er replikering ganske brugt.

Bottom-up

Man har eksisterende databaser og ønsker at gøre dem til en distribueret database.



4.12.2.3. Faciliteter i SQL-server (ang. Distribuerede databaser)

- Distribuerede queries (både homogent (mod anden SQL-server) og heterogent (mod anden type server))
- Remote Stored Procedures (kan kalde SP på andre maskiner, dog kun homogent)
- Distribuerede transaktioner
- Replikering

4.12.2.4. Navngivning i SQL-server

Det fulde tabelnavn for tabeller i SQL serveren er

```
OpretterId.TabelNavn
```

Hvis man selv har oprettet tabellen kan man nøjes med at skrive tabelnavnet. Har man ikke selv en tabel med det pågældende navn vil den finde `dbo.TabelNavn` (dbo er identifikationen for standardbrugeren/administrator (sa)).

Til brug for distribuerede databaser skal den dog være således

```
ServerNavn.DatabaseNavn.OpretterID.TabelNavn
```

Der skal [] om servernavn hvis der indeholder speciel-tegn (fx -). Det er desuden en god ide at give den et 'kælenavn' når man kalder den (fx as servTabel).

Kræver to ting (sikkerhedsmæssigt)

- At den server, man vil nå, er oprettet som linked server
- At den login, der affyrer query'en bliver oversat til en login, der er kendt på den modtagende server.

Security → Linked Servers → add (gå ind under security og fortæl hvilket lokalt login der skal svare sig til hvad på den anden server – ude i virkeligheden vil man ikke bruge sa!)

4.12.3. Distribuerede queries

I SQL-server

- navngivning: server.database.opretterlogin.tabelnavn
- sikkerhed: man skal kende et login + password fra den server, hvor tabellen er
- heterogent: virker mod en række andre DB via Providere (remote stored proc virker kun mod samme type)
- transparens: via views

4.12.3.1. Optimering

Hvad sker der når SQL-serveren får en select-sætning

- Check for syntaks-fejl
- Oversætter select-sætningen til relationel algebra
- Optimer
- Find data

4.12.3.2. Opgave 5.1 (fra udleveret kompendium)

En distribueret database indeholder en oversigt over leverandører (Suppliers – kaldet S) og varedele (Parts – kaldet P) og leverancer (SP):

Relationsnavn	S	SP	P
Relation	(sno,city)	(sno,pno)	(pno,color)
Antal tupler	10.000	1.000.000	100.000
Site	A	A	B

Alle felter er 500 bits lange og hver tupel er dermed 1000 bit.

Følgende query skal udføres:

```
Select S.sno
From S,SP,P
WHERE S.city = 'London' AND S.sno = SP.sno
AND SP.pno = P.pno AND P.color = 'red'
```

Antagelser:

Antal røde dele (Parts) = 1000

Antal leveringer af London-Suppliers = 100.000

Datatransmissions rate = 1.000.000 bits/sekund

Access delay (udførelse af arbejde på site) = 1 sekund

Responstiden for forespørgslen kan udregnes efter følgende formel:

T = Total Access delay + (Total datavolume / Datatransmissions rate)

Udregn responstiden ved følgende behandlingsstrategier og diskuter accesstiden contra datatransmissionstiden.

T = 1 sekund * antal pakker + (antal bits sendt / 1 mil. bits/sek.)

1. Flyt relation P til site A og udfør forespørgslen på A.

Flytte hele P til A: 100.000 tupler af 1000 bits (alt sendes i én pakke)

T = 1 sekund + (100.000.000 / 1.000.000) = 1 + 100 = 101 sekunder
= 1 min 41 sekunder

2. Flyt relation S og SP til site B og udfør forespørgslen på B.

Flytte både S og SP til B: 10.000 + 1.000.000 tupler af 1000 bits (alt sendes i én pakke)

T = 1 sekund + (1.010.000.000 / 1.000.000) = 1 + 1010 = 1011 sekunder
= 16 min 51 sekunder

3. Join S og SP på A og udvælg de tupler, der har city = "London". For hver sådan tupel, checkes B for at se, om stykdelen er 'red'. Hvert af disse check vil medføre, at to messages sendes, forespørgslen og svaret.

For join af S og SP forespørges B om P er rød: 100.000 forespørgsler af 500 bits: pno. Samt svar: ja/nej

T = 1 sekund * (100.000 * 2) + (500 * 100.000 / 1.000.000) = 200.000 + 50 = 200.050 sekunder
= 2 dage 7 timer 34 min 10 sekunder

4. Udvælg tupler fra P på B, som har color = 'red' og for hvert af disse check på A, om der eksisterer en levering på den stykdel relateret til en London-supplier. Igen to messages pr. check.

For hver rød P checkes om S er fra London: 1000 forespørgsler af 500 bits: pno. Ditto svar: sno/nej

T = 1 sekund * (1000 * 2) + ((500 * 2 * 1000) / 1.000.000) = 2000 + 1 = 2001 sekunder
= 33 min 21 sekunder

5. Join relation S og SP på A, udvælg de tupler fra resultatet, der har city = 'London' og projekter resultatet over SNO og PNO, hvorefter dette overføres til B for færdigbehandling.

Join S og SP, de hvor S er fra London sendes til B: 100.000 tupler af 1000 bits: sno & pno (alt sendes i én pakke)

T = 1 sekund + (100.000.000 / 1.000.000) = 1 + 100 = 101 sekunder
= 1 minut 41 sekunder

6. Udvælg de tupler P på B, som har color = 'red' og flyt resultatet til A for færdigbehandling.

Flytte alle se røde P til A: 1000 tupler af 500 bits: pno (alt sendes i én pakke)

T = 1 sekund + (500.000 / 1.000.000) = 1 + 0,5 = 1,5 sekunder
= 1,5 sekunder

4.12.3.3. Optimizers algoritme

1. Omform SQL til operatorgraf

2. Brug ækvivalenstransformationer
3. Hvis tabeller er fragmenteret så indsæt rekonstruktion
4. Vælg behandlingssted (dvs. hvad skal sendes)

2-4 laves ikke i rækkefølge.

For at vælge mellem ”gode” strategier har man en profil. Profilen indeholder statistiske oplysninger om data.

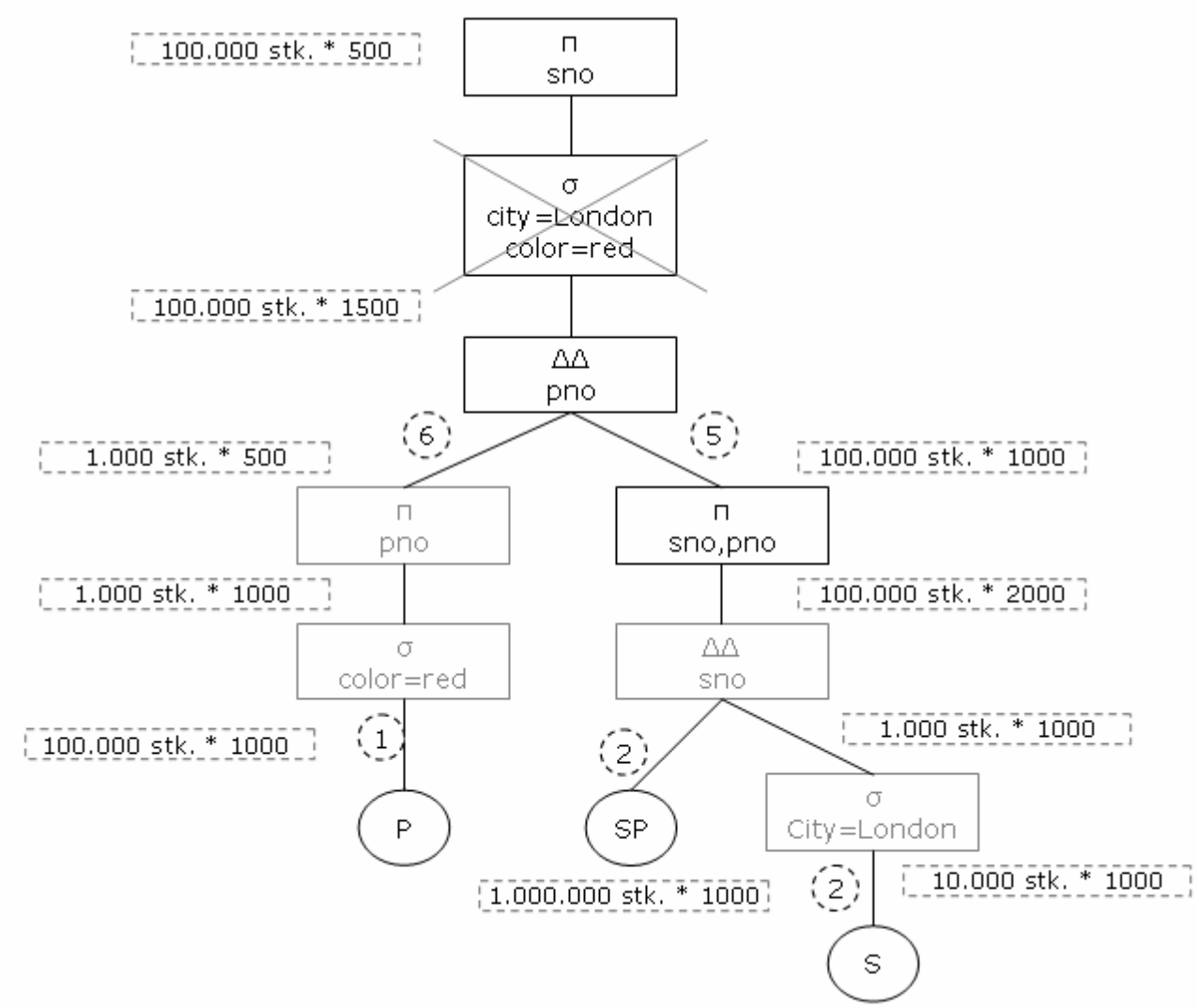
```
C (R) = antal records i tabel R
B (X) = antal bytes attribut X fylder
C (R,X) = antal forskellige værdier, som X antager i R
```

Eksempel (opgave 5.1)

```
C (S) = 10.000
C (SP) = 1.000.000
C (P) = 100.000

B(alle attributter) = 500 bit

C(P,color) = 100
C(S,city) = 10
```



Figur 2: Operatorgraf for opgave 5.1 med optimeringer

Strategier for semi-join eksempel

Deltager på site A

Navn (char(50))	Kursusid (char(10))
Hans Hansen	12
Ole Olsen	11
Henning Henningsen	12
Ib Ibsen	11
Per Persen	4
Kurt Kurtsen	1
Palle Pallesen	1
Jørgen Jørgensen	2
Ebbe Ebbesen	3
Claus Clausen	3

Jesper Jespersen	4

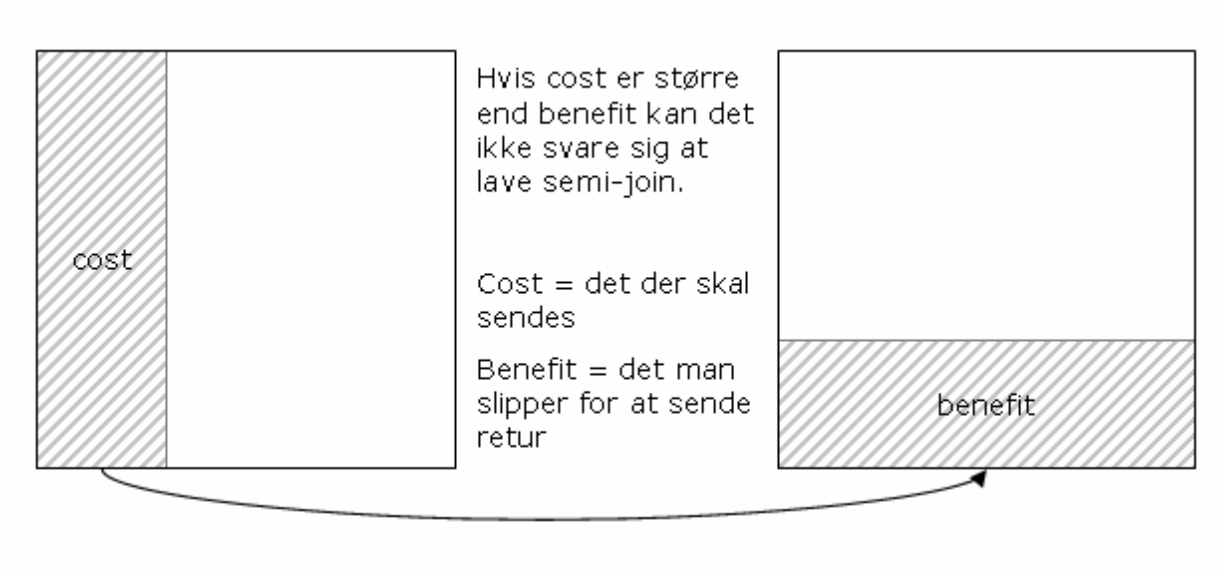
Kursus på site B

Kursusid (char(10))	Kursustrin (char(100))
1	Begynderstrikning 1
2	Begynderstrikning 2
3	Begynderstrikning 3
4	Begynderstrikning 4
5	Begynderstrikning 5
6	Objektorienteret strikning 1
7	Objektorienteret strikning 2
8	Objektorienteret strikning 3
9	Distribueret strikning
10	Client/Server strikning
11	Baglæns strikning
12	Transaktionsstyret strikning

Query (udført på site A)

```
SELECT navn, kursustrin
FROM Deltager, Kursus
WHERE Deltager.kursusid = Kursus.kursusid
```

1) Flyt kursus over på A
    a. 12 stk. * 110 bits = 1320 bits sendes i alt
2) Flyt Deltager over på B, join der og send resultat tilbage til A
    a. 11 stk. * 60 bits = 66 bits sendes
    b. 11 stk. * 150 bits = 1650 bits sendes tilbage
    c. 66 + 1650 = 2310 bits sendes i alt
3) Send $\pi_{kursusid}$A over til B. Udregn B ΔΔ $\pi_{kursusid}$A og send det tilbage til A (semi-join)
    a. 6 stk. * 10 bits = 60 bits sendes
    b. 6 stk. * 110 bits = 660 bits sendes tilbage
    c. 60 + 660 = 720 bits sendes i alt



Mål indenfor oprimering

Central (en DB-server)	Distribueret (flere DB-servere)
Mål: færrest mulige disk- I/O	Mål: færrest bytes sendt
SQL → operatorgraf → brug ækvivalenstransformationer Vælg via profil	Som central + fragmentering + valg af behandlingssted
Hvornår laves optimering Dynamisk SQL: hver gang query modtages Stored procedures: ved oversættelsen + ved recompile	Høj transparens: helt som centralt Lav transparens: programmør laver optimering

4.12.3.4. Opgave 5.8 (fra kompendium) – opgaveformulering ikke vedlagt

```
SELECT KU.navn, KU.adr
FROM KU, KØ, VA
WHERE KU.kundenr = KØ.kundenr AND KØ.varenr = VA.varenr
AND KØ.lev-fra = ”J” AND VA.varenavn = ”jeepdæk”
```

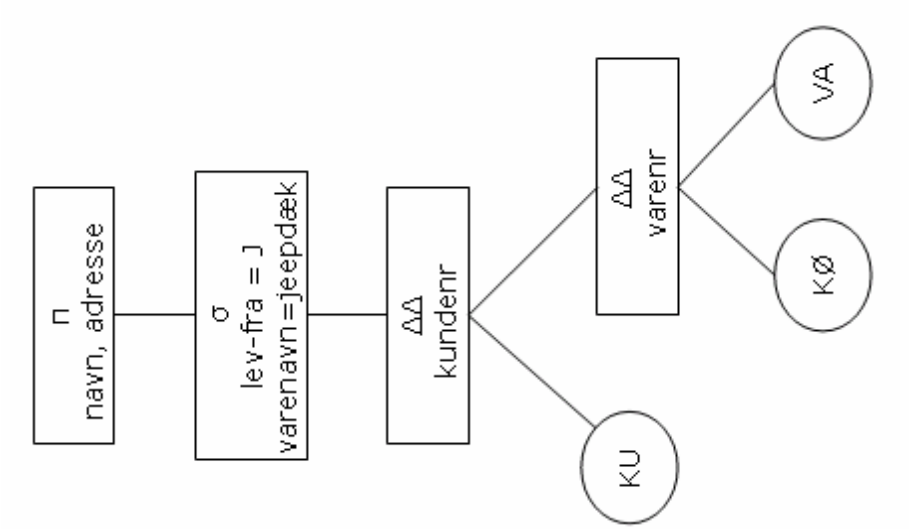
a) Profil

```
C (R) = antal records i tabel R
B (X) = antal bytes attribut X fylder
C (R,X) = antal forskellige værdier, som X antager i R
```

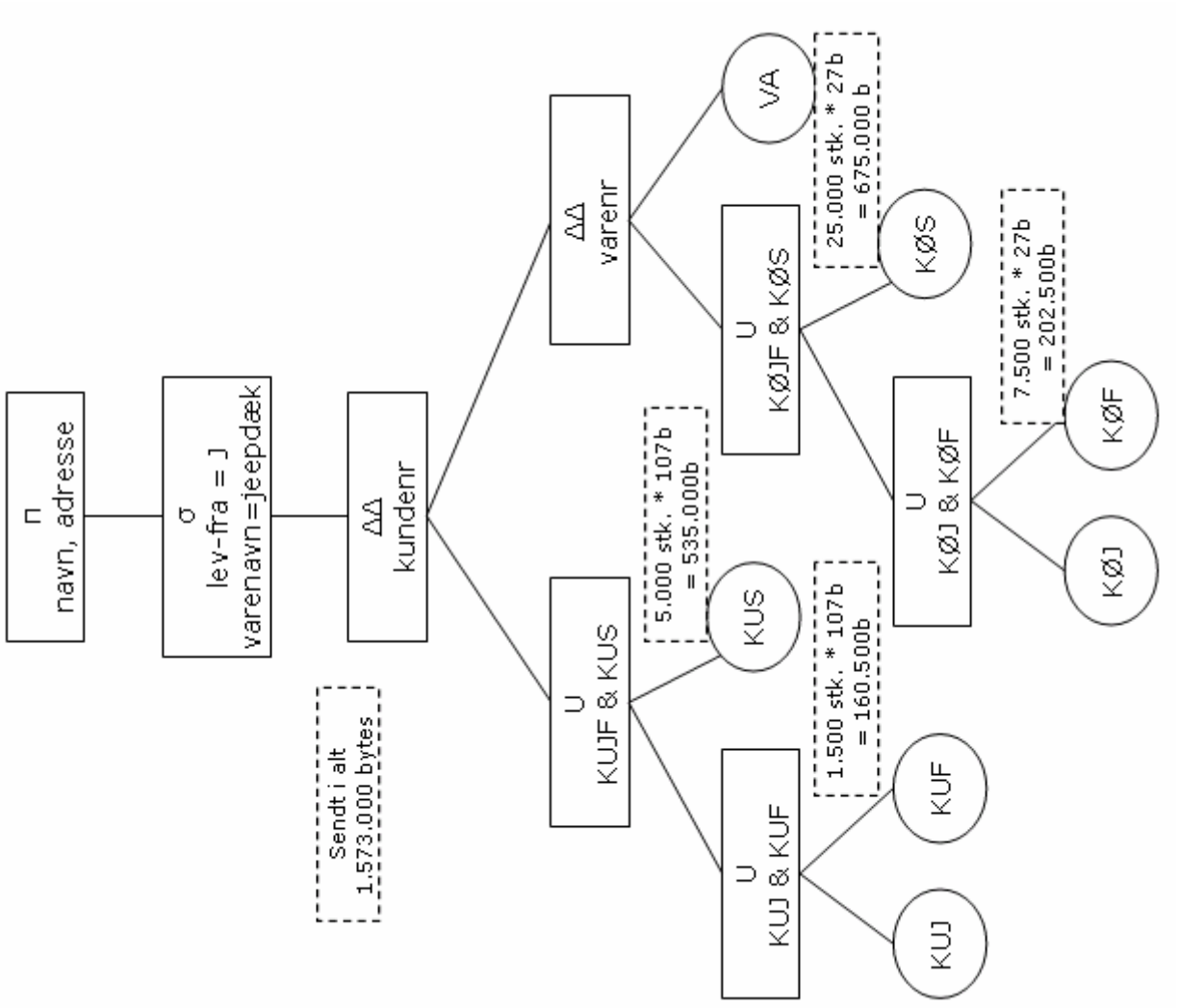
C(R)	J	F	S	I alt	B(tabel)
KU	3000	1500	5000	9500	107 bytes
KØ	15000	7500	25000	47500	27 bytes
VA	2000	2000	2000	2000	62 bytes

C(KØ, kundenr) = 9500 (dvs. gennemsnitligt 5 køb pr kunde)

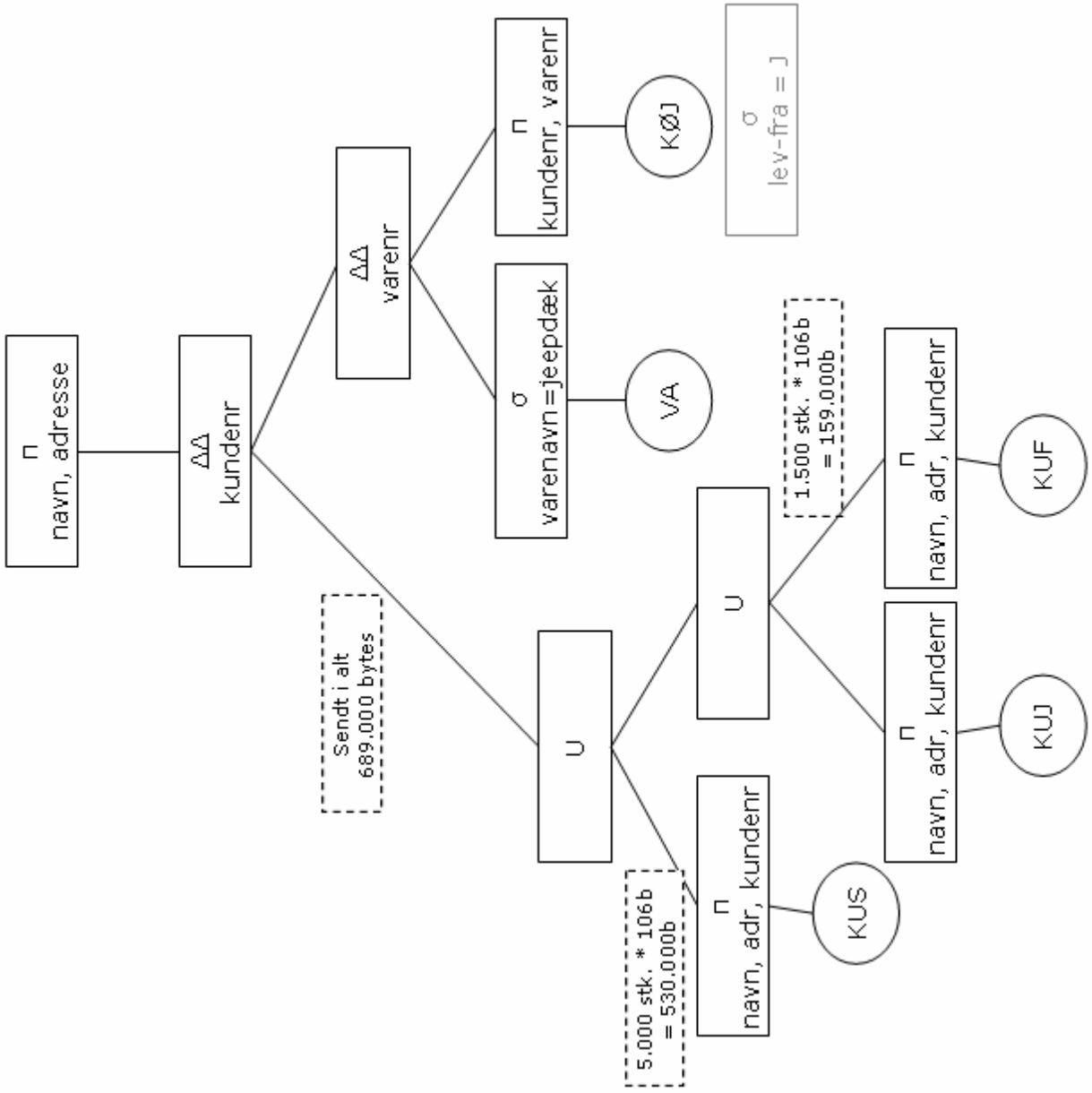
C(KØ, varenr) = 2000 (dvs. gennemsnitligt er hver vare solgt 23,75 gange)



Figur 3: Delopgave b - Operatorgraf



Figur 4: Delopgave c - Operatorgraf med distribution og sendt data mængde



Figur 5: Delopgave d – Ækvivalenstransformationer

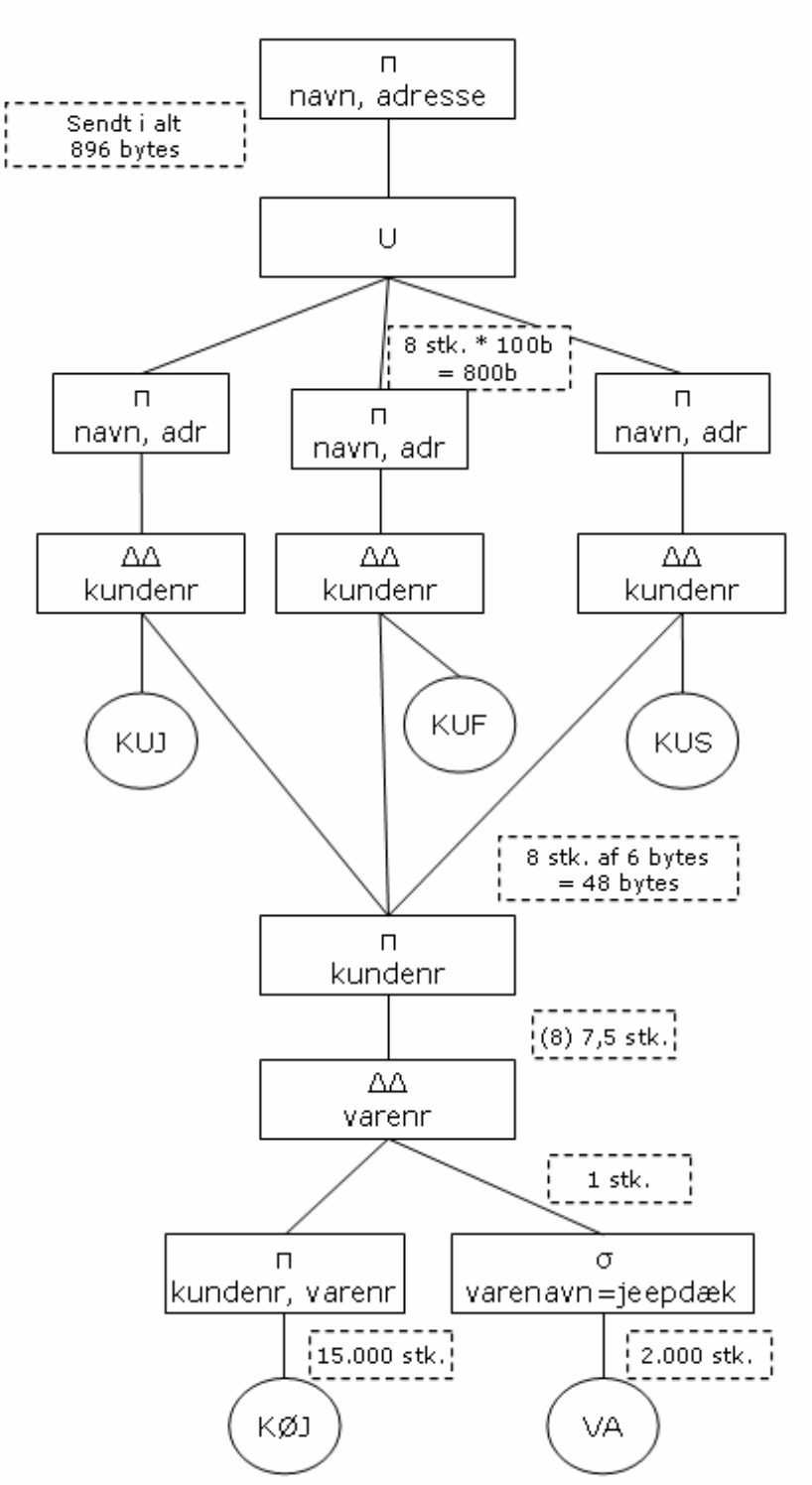
e) Mulige semi-joins

Ved at finde frem til de køb af jeep dæk foretaget i Jylland har vi kunde numre på de kunder vi skal have navn og adresse på.

Ved at sende disse kundenumre (beregnet 8 stk.) til KUS og KUF (samt KUJ internt) kan vi nøjes med at sende 8 kundenumre af hver 6 bits til 2 steder = 96 bits. Vi beder så om kun at få navn og adresse retur idet vi ikke har behov for andre variabler.

I alt bør der ikke kunne være mere end de 8 kunder som svar, dette værende fordelt på alle fragmenterne. Dette vil højest give en forsendelse på 8 gange 100 bytes = 800 bytes.

Samlet giver det en trafik på 896 bytes.



4.13. Commitment-protokol

En commitment-protokol skal bruges, hvis man i en atomar transaktion ønsker at opdatere data på flere sites. (Når man vil opdaterer data på flere sites (db-servere).)

Er en smule bekostelig (over lokalnetværk kan der godt være tale om nogle sekunder).

4.13.1.1. Distribuerede transaktioner

ACID for distribuerede databaser adskiller sig kun fra ACID for centrale databaser ved A hvor man ikke kun gør brug af log men også gør brug af en commitment-protokol. Samtidig er det nødvendigt at de forskellige sites udveksler wait-for grafer, idet det ellers er svært at se om der er opstået en deadlock.

4.13.2. 2-phase commitment (2PC)

Én koordinator. (I Windows 2000 hedder den DTC[8]

Deltagere: de databaseservere, der har data med i transaktionen.

Regler

- Deltagere stemmer commit eller abort (stemmen er bindende)
- Afgørelsen er kun commit hvis alle stemmer commit
- Har man stemt commit skal man afvente global afgørelse

Fejlsituationer

- Deltagere får aldrig 'vote'
    - Deltagere beslutter abort
- Koordinator får aldrig stemme
    - Tager det manglende svar som abort
- Deltagere får aldrig global afgørelse
    - Der startes en termineringsprotokol (startes af en af de ventende deltagere)

4.13.2.1. Termineringsprotokol

Spørg de andre deltagere om de ved noget (de andre deltagere er kendt ud fra 'vote' kaldet)

- Hvis en kender den globale afgørelse, så følg den
- Hvis jeg kan få fat i alle andre deltagere, så stem om
- Hvis en af de andre stemte abort så vælg abort
- Ellers (hvis intet af ovenstående) så er man BLOKERET. Her er manuel indgriben den eneste mulighed for at komme videre.
    - Dette problem/tilfælde bør kun kunne opstå hvis alle har stemt men at den globale afgørelse ikke er sendt ud (og at en af maskinerne går ned sammen med koordinator)

4.13.3. 3-phase commitment-protokol (3PC)

Er kun opfundet for at undgå blokeringssituationen. Den har en bedre termineringsprotokol.

Der er risiko for at ødelægge det atomare men til gengæld kan der ikke ske blokering.

4.13.3.1. Termineringsprotokol

Vælg en ny koordinator blandt dem du kan få fat i. Den nye koordinator spørg alle om status.

Hvis bare én har fået 'pre-commit' så gå mod commit, ellers gå mod abort. (dette er sikkert nok!)

[8] Distributed Transaction Coordinator

Problem i 3-faset commitment (der er kun det ene)

NET-DELINGER (opdelingen af nettet i mindst to dele, der kører)

4.14. Replikering/Replikation

Permanent opbevarer flere kopier af samme data på flere servere.

Hvorfor replikere

+ Større tilgængelighed

+ Ekstra backup (varm backup = kører og er klar til at blive sat ind)

+ Hurtigere svartider (for læse applikationer)

- Vanskeligere opdateringer*

- Risiko for inkonsistens*

- Større diskforbrug

* = det der arbejdes med

4.14.1. Generelle opdateringsmetoder ved replikering

Strongly Consistent

Foregår i en distribueret transaktion (2-faset commitment)

Alle kopier er altid nøjagtig ens, men kræver at alle sites er oppe ligeledes er det meget tidskrævende.

Weakly Consistent

UDEN MASTER

Strategi: Opdater egen kopi, lad programmet køre videre, send så opdateringer til de andre (i et baggrundsjob)

Kan give inkonsistente data.

MED MASTER

For hver tabel udnævnes én kopi til master

Strategi: Opdateringer går først til master, der herefter opdaterer de andre.

4.14.2. Replikering i SQL-server 2000

Hvordan opdateres replikerede data?

Strongly konsistent supporteres via distribueret transaktion (commitment-protokol).

- kræver alle kopier tilgængelige
- sikrer at alle kopier altid er helt ens
- forholdsvis langsom ved opdateringer

Weakly konsistent UDEN master har man ikke.

Weakly konsistent MED master findes i mange varianter. (Master kaldes for Publisher)

- sikres at data bliver ens (på sigt), men kortvarigt kan kopier være forskellige
- hurtig (sammenlignet med strongly)
- kræver kun at master er tilgængelig

- Findes i en del varianter!!
    - Merge (ændringer sendes rundt)
    - Snapshot (hele tabellen sendes, med selvbestemte mellemrum)
    - Transactional (egen kopi må ikke opdateres – kun Publisher)
    - Updatable snapshot (egen kopi og Publishers opdateres i en distribueret transaktion (2-faset commitment-protokol))
    - Diverse kombinationer

4.15. Sikkerhed

- Virus og lignende
- Bruger id/password + autencitet
- Kryptering
- Digital signatur
- Firewall
- Web-sikkerhed

4.15.1. Backup/restore

- Backup-systemer bruges naturligvis ved alle situationer, hvor systemet ikke er tilgængeligt, dvs også hvis en hacker med vilje eller en legitim bruger ved et uheld har ødelagt systemet helt eller delvist
- Når man planlægger en backup/restore strategi er der to væsentlige spørgsmål
    - Hvor lang tid går der fra "uheldet" til systemet er kørende igen
    - Hvor mange data mangler der midlertidigt eller permanent?
- Man taler i erhvervslivet meget om begrebet "varme backup'er
- En varm backup dækker over en kørende maskine, der via replication har en kopi af data.
- Ved varme backup'er vil backup-maskinen typisk være kørende indenfor sekunder eller minutter
- På grund af de anvendte replication-strategier vil back-upmaskinen ofte mangle dataændringer fra det seneste tidsinterval (kan være fra sekunder til et døgn)
- Backup'en skal helst ikke være fysisk i nærheden af den oprindelige maskine
- Alternativet til varme backup'er er at køre ens seneste back-up ind på en maskine og derefter alle logs man har.
- En sådan restore kan tage temmelig lang tid. Hos visse virksomheder op mod i størrelsesorden døgn!
- Uanset back-up strategi er det ekstremt vigtigt at restore-proceduren jævnligt aftestes!
- Mediet er også relevant. Harddisk er hurtigere end magnetstriber
- Generelt problem: for mange 'gamle' data = længere backup og restore tid

4.15.2. Problemer med brugerid og password

4.15.2.1. Hacker-tricks

Phishing

Lav et program, der ligner det rigtige og lad brugeren taste brugerid og password

Social engineering

Snak dig til at få et password

Bestikkelse

Overtalelse

Trojansk hest

Placer et program sammen med en 'uskyldig' download

Brute force

Prøv dig frem (typisk ved hjælp af en maskine) og udnyt at mange passwords er nemme

Aflurer koden

Kig over skulderen, gule sedler med videre

Aflurer password når det endes over nettet eller tastes

Udnytte bagdøre/svagheder i systemet

Dumpster daiming (Led i affald)

Mange virksomheder smider papir ud i en bestemt container

Aflur brugt hardware

Specielle brugere

sa, admin guest

Brugere anvender samme password til alle systemer

4.15.2.2. Modtræk

- Virusskjold mod trojansk hest
- Krav til password (længde, specielle tegn)
- Skift adgangskode tit (både og)
- Max antal login forsøg
- Lav logninger af login forsøg
- Kræv noget mere, ud over password (private keys): nøgle fil, RSA[9]-key, skrabelod, hardware-dims
- Information til brugere
- Brugere låst på en form for net identifikation

4.15.3. Ondsindede programmer (virus)

Virus opstod for alvor med Pc'erne.

[9] En lille plastisk ting med tal, som så skal tastes ind sammen med ens eget password

Tidligere flerbrugersystemer indeholdt et sikkerhedssystem, der i stort omfang forhindrede ondsindede programmer i at gøre noget ondt! Sikkerheden bestod som sådan i at en systemadministrator specifikt skulle give programmet adgang til bibliotekerne.

Pc'ere indeholder ikke sikkerhedssystemer, dvs. et program har i princippet frit spil på hele computeren.

De første virus på PC spredtes via disketter (boot-sektor virus).

4.15.3.1. Typer af ondsindede programmer

Logisk bombe

Programmør har lavet "snavs" i et legitimt program. (bevidst)

Bagdøre

Programmør laver "short-cuts" i systemer

Trojansk hest

Et program, der tilsyneladende laver noget godt (men indeholder noget skidt)

Virus

Et program, der kan sprede sig selv, typisk til andre filer (eller computere).

Tidligere var Word og Excel 'smittebærere', nu er det mere e-mail m.v.

Worm (orm)

Et program, der spreder sig selv gennem netværk.

Typisk via e-mail. Den angrebne mail/vedhæftede fil aflæser ens adresse-kartotek og sender sig selv videre.

Zombie

Ens computer er overtaget delvist af en anden, typisk så den kan deltage i D-DOS[10]-angreb

4.15.3.2. Håndtering af virus trusler

- Sund fornuft
- Anti-virus produkter (altid bagud)
- Lav en speciel opsætning (placer ting et andet sted end der hvor de forventer det)
- Sikkerhedsrettelser

4.16. Net sikkerhed

Kryptering løser ikke alle problemer. Faktisk er der større trusler.

Hvad vil vi gerne opnå?

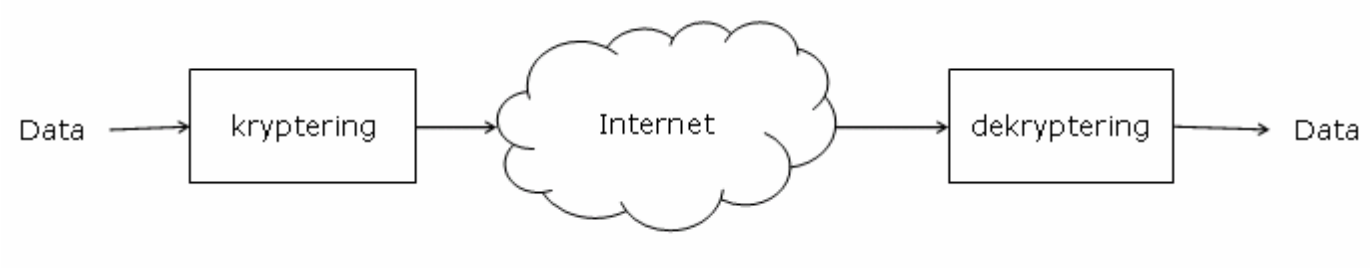
- Fortrolighed (ingen ser oplysninger)
- Authencifikation (sikker på hvem vi "taler" med)
- Integritet (ingen uvedkommende kan ændre data)
- Tilgængelighed (ingen kan tvinge systemet ned)

Hvor er de usikre punkter i netværket?

[10] Distributed-Denial Of Service

- Broadcast medier
    - Trådløst net, mobiltelefoni, sattelit & radio
    - LAN
    - HFC (Internet bygget op på kabel TV teknologien)
- Internettet
    - Ingen har helt styr over det samlede Internet

4.16.1. Kryptering



4.16.1.1. God krypteringsalgoritme

- Svær at bryde
- Skal kunne dekrypteres (kræver forudgående enighed)
- Hurtig algoritme

4.16.1.2. Opgave 10

Afprøvning af simpelt krypteringsprogram

Hvordan krypterer denne metode? Hvilke ulemper har denne metode? Kunne du foreslå forbedringer til metoderne?

- Flytter alle tegn ”key” i tegnsættet
- Nem funktion
- Key har lille værdimængde (let at bryde)

Forslag til forbedringer

- Lad key ændre sig efter en algoritme

4.16.1.3. Hvordan kan man knække en key?

- Prøv med andre key-værdier
- Hvis man har et sammenhørende ikke-krypteret og krypteret så lad et program forsøge, ellers
    - Check selv manuelt
    - Statistisk/ordbogs-analyse

4.16.1.4. Symmetrisk (secret-key)

- Afsender og modtager er enig om en hemmelig nøgle
- Hemmelig nøgle udveksles via en anden kanal
- Mest kendte

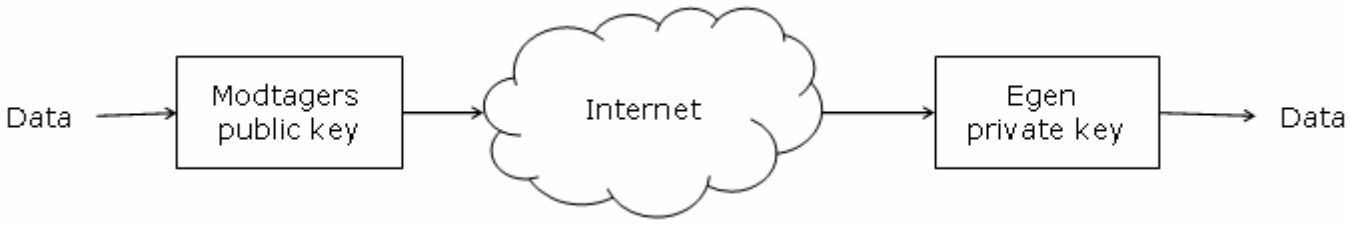
- DES

- Hurtigere end public key

4.16.1.5. Public key

Ideen er at enhver modtager af data laver to nøgler.

- Public key (fortæller den til alle)
- Private key (beholder for sig selv)



- Man skal være sikker på at en public key kommer fra den man tror den kommer fra
- Langsomt

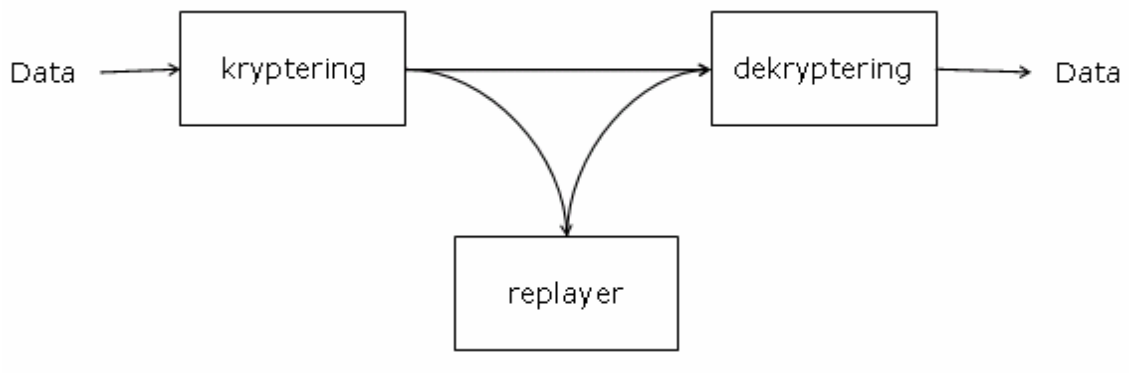
  Man må ikke ud fra metode og public key kunne udregne private key

4.16.1.6. RSA-metoden

Metode	Eksempel
Vælg to store primtal (minimum 100 cifre) p og q	p = 5 & q = 7
n = p * q z = (p-1) * (q-1)	n = 35 z = 24
Vælg e < n indbyrdes primisk med z (e må ikke gå op i z)	e = 5
Find et tal d så z går op i (e * d) - 1	d = 29
Public key = (n,e) private key = d	(35, 5) 29
m = meddelelse der skal sendes Krypteringsfunktion en: m^e % n c = modtagen krypteret tekst Dekrypteringsfunktion: c^d % n Generelt krypteres der i bidder af k bits, hvor 2^k < n	

4.16.2. To angrebsmetoder

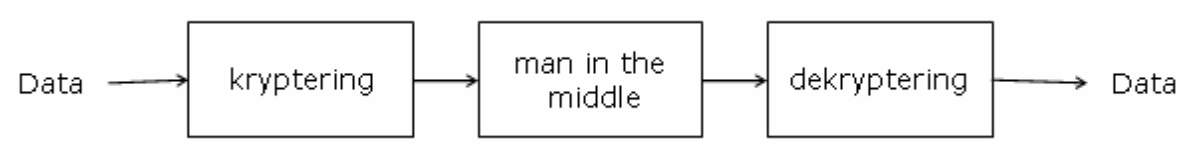
4.16.2.1. Replaying



Optaget det der sendes og prøver at sende det igen på et andet tidspunkt. Gøres for at hærge. (bilnøgle eksemplet)

Et problem med denne teknik kan være at de gensendte data er 'ude af sekvens' og modtageren derfor indser at der er 'noget galt'.

4.16.2.2. Man in the middle attack



Angriber videresender beskeder (evt. modificeret).

Dette kræver i en hvis grad at den i midten har kontrol over hele kommunikationen.

4.16.3. Authencifikation

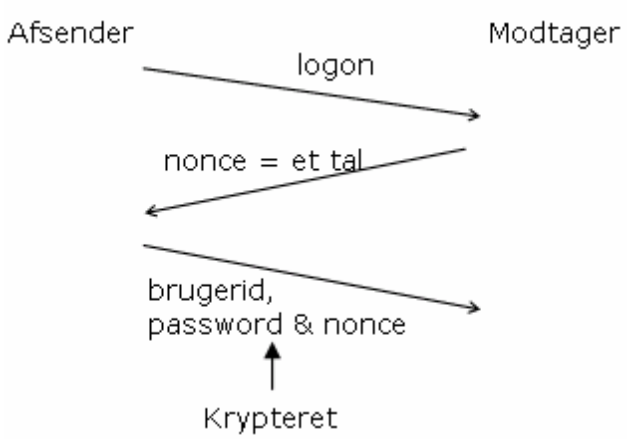
Hvordan er jeg sikker på, at den jeg "snakker" med er den, han siger han er.

4.16.3.1. Forslag

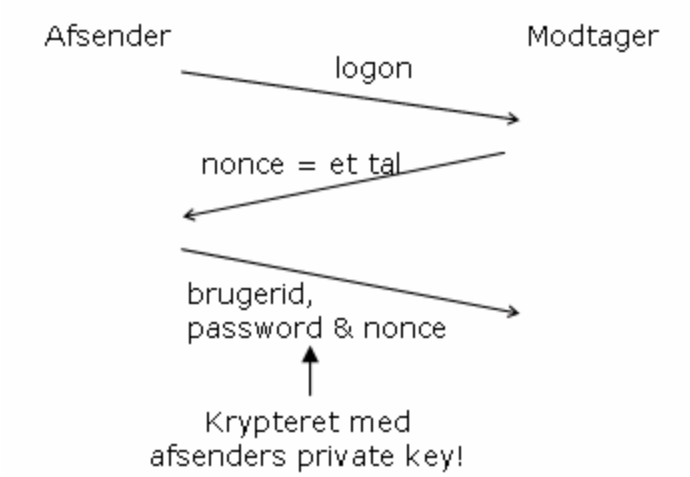
- Præsenter mig selv (let at snyde)
- Identifikation ud fra afsender adresse (let at snyde med adresser)
- Navn + password (afluring i nettet)
- Navn + password krypteret (replaying)

4.16.3.2. Sikre authencifikations protokoller

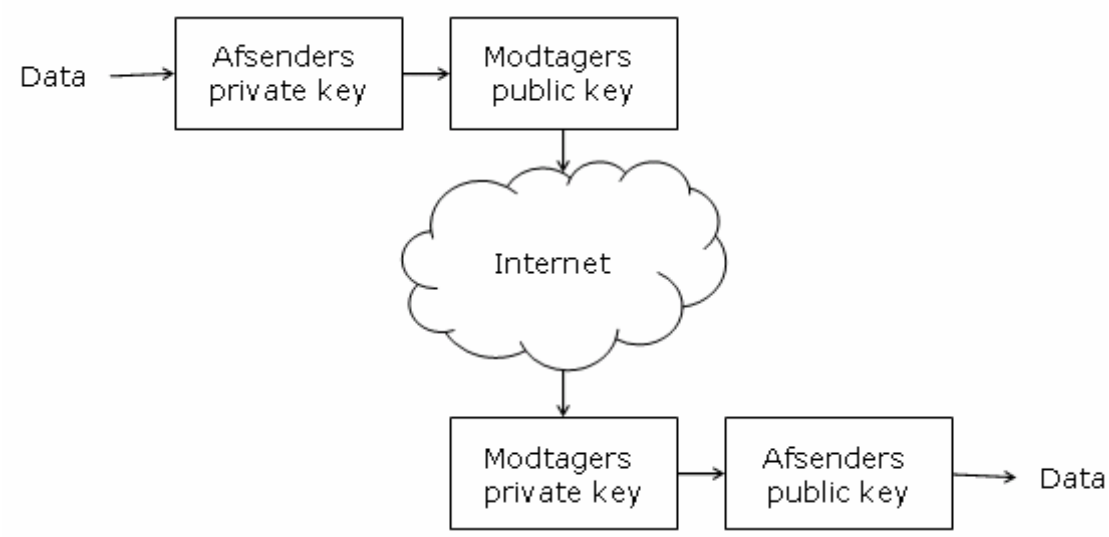
- Symmetrisk key



- Public key



  - Fortroligt og authencitet



    - Denne metode virker hvis man kan stole på at de public keys man har virkelig stammer fra den rigtige!

4.16.3.3. Digital signatur (uafviselighed)

Et digitalt modstykke til en underskrift, der skal sikre at et dokument er

- "underskrevet" af den rette (authencitet)
- Ikke ændret efterfølgende (integritet)

Løsning

Sendes som: dokumentet + {dokumentet} krypteret med egen private key. Modtager kan så checke at dokumentet ikke er blevet ændret.

(Eller: dokumentet + {digest/checksum af dokumentet} krypteret med egen private nøgle.)

Problem

Det fylder dobbelt så meget.

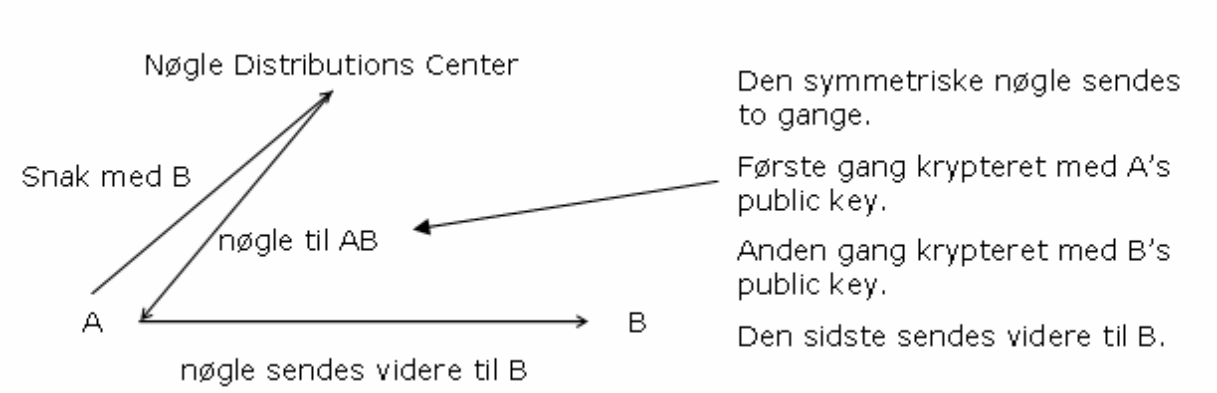
Man kan undlade at kryptere hele dokumentet og i stedet lave en message-digest (en check sum), der krypteres. Modtager checker så at det ukrypterede dokuments check sum passer med den krypterede sum.

4.16.3.4. Nøglecentre

Symmetrisk nøgle

Det er besværligt at have en symmetrisk nøgle til alle man snakker med.

Enhver deltager i kommunikation har én symmetrisk nøgle (til nøglecentret). Når man vil tale med en, beder man nøglecentret om en engangsnøgle til den pågældende modtager.



Public key

Bruges nøglecentre til at kontrollere, at en public key kommer fra den man sider den kommer fra.

Fx Verisign

4.16.4. Firewall

- Placeres i knudepunker i nettet
- Forskellige typer af firewalls
    - Pakkefiltre (den vi mest vil komme til at arbejde med)
        - Arbejder på lag 3+4 i modellen (IP + TCP/UDP niveau). Kan stoppe trafik ud fra indhold i headere.
    - Applikations gateways (kaldes lag 7 gateways)

        - Arbejder på lag 5. Stopper trafik ud fra indhold
    - IDS-systemer (Intrusion Detektion System) – er svære at lave
        - Analyse af datatrafik
        - Kan forveksle belastninger med DOS-angreb

4.16.4.1. Angreb

Portscanning

Check hvilke porte, der er åbne. Man ved hvilke programmer, der kører på serveren

Pakke-sniffere

Anvendes til at se hvad der sendes på broadcast-medier. (Husk at mange steder sendes bruger og password som ren tekst)

IP-spoofing

Angiv en forkert afsender IP. Kan også gøres gennem en proxy-server.

4.16.4.2. Forskellige sikkerhedstiltag

E-mail området

Man ønsker:

- fortrolighed
- authencitet
- digital signatur

Vi har set teknikken til det

{besked, {digest(besked)} krypteret med egen private key} krypteret med modtagers public key

PGP (Pretty Good Privacy) Bruger denne ide. Laver dog distribution af public keys lidt anderledes

SSL (Secure Soket Layer)

Et lag lige ovenpå det almindelige socket-lag.

SSL tilbyder:

- authencifikation af server
- kryptering af datatrafik

SSL bruges i https

- authencifikation af server
- bruger public key til at sende en symmetrisk nøgle, som bruges til krypteringen.

4.17. Web-sikkerhed

Mange angreb rammer alene web-serveren (dvs. design og evt. at 'lukke' siden)

4.17.1. Angreb (web)

- Tilfældige angreb – langt de fleste
  - Overfladiske, simple
- Mere vedholdende angreb
  - Planlagte, langvarige angreb
- Avancerede angreb
  - Eksperter, tidligere ansatte
  - Mod militære, hotte steder, netbanker

4.17.2. Kendte angrebsmåder

- SQL-injektion
- Cross-site scripting
- Buffer overflow
- Session hijacking

4.17.2.1. SQL injektion

Kan laves på alle systemer, der anvender dynamisk SQL. (Dvs. ikke kun WEB)

Typiske kommandoer

' ' i første felt vil typisk give en fejl der lader dig vide hvilken type DB du er oppe imod (i hvert fald i SQL server)

-- i SQL server (ignorerer resten af sætningen)

' OR 1='1 kan for det meste bruges

Simpel løsning på problemet er validering.

4.17.2.2. Cross-site scripting

Avanceret form for phishing.

Snyder en anden til at komme ind på en side som han/hun tror han kender, men det er i virkeligheden hackerens side.

4.17.2.3. Buffer overflow

Findes reelt i alle IT-systemer. Kan løses med validering. Resultatet er ofte at systemet går ned, men det er generelt uforudsigeligt.

Optræder hvis brugeren/hackeren angiver et længere indhold end der er plads til i et input felt.

4.17.2.4. Session hijacking

Hacker laver om på en cookie som er blevet lagt på hans maskine og derved kan han overtage en andens session.

For at undgå dette bruges ofte lange uforudsigelige session ID'er.

4.17.3. Andre metoder

4.17.3.1. Udnytte kendte svagheder i basissoftware

Det eneste man kan gøre for at imødegå hackning er at have de nyeste sikkerhedsopdateringer.

4.17.3.2. Almindelige dårlige vaner

Hav ikke gamle programmer eller én gangs programmer liggende. Generelt skal der ikke på web-serveren ligge noget som helst som andre ikke må kalde.

4.17.3.3. Denial Of Service attack

Forsøg på at ligge en web-server ned eller blot gøre web-sitet uanvendeligt i en periode. Kan udvides til Distribueret DOS angreb.

Den eneste måde man kan imødegå DOS angreb er at have overdimensionerede servere, det kan dog også hjælpe at begrænse ressourcetunge funktioner.

5. REPETITION AF CNDS, 3. SEMESTER

5.1.1. (Intro – generelt)

5.1.1.1. Driftsmodeller

Central: Maskinkraft og data er samlet

Decentral: Maskinkraften er decentral (← relativt let) Data decentralt (kan være svært)

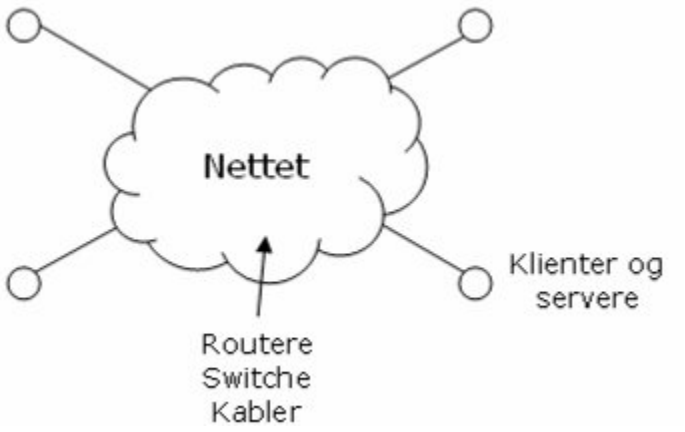
Distribueret: Maskinkraft er decentral, Online access til andres data

Distribuerede systemer

Definition: Et distribueret system afvikles som regel som et samarbejde mellem flere systemer.

Nøgle ord: Tilgængelighed (oppe tid), fejltolerant (kunne gå videre selv ved fejl), skalerbarhed (kan vokse kraftigt uden at skulle re-designes), transparens (programmer bør ikke indeholde viden om konkret struktur/konfiguration), replication (se senere), sikkerhed (se senere).

5.1.2. (Intro til netværk)



Figur 6: Opbygning af Internettet

5.1.2.1. Typer af net trafik

Data til administrative systemer. Krav: korrekte data, forsinkelser accepteres

Lyd, billeder o.l. Krav: forsinkelser accepteres ikke, udfald el. lign. accepteres

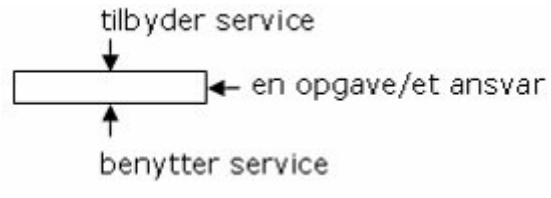
5.1.2.2. Svartider på net

Samlede net tid

kø til (kan variere)+
udbredelsestid (typisk lysets hastighed) +
behandlingstid/ekspeditionstid +
transmissionstid (hvor lang tid tager det at sende en bit)

5.1.3. Lagdeling

Ide: hvert lag har en opgave og et ansvar, tilbyder service til højere lag og benytter sig af service fra lavere lag.



Figur 7: Lags interaktion med hinanden

OSI-modellen har/havde 7 lag

TCP/IP har 5 lag (det er denne model der benyttes)

	Lag	Afsender		Modtager
Program {	1	Application	*logisk kommunikation* fx mellem to mennesker	Application
		data, 'adresse'		
TCP/IP driver { (følger med OS)	2	Transport	snakke mellem programmer	Transport
	3	Netværk	snakke *mellem maskiner*	Netværk
		(Driver til netkort)		(Driver til netkort)
Netkort/Hardware{	4	DataLink	snakke *mellem direkte forbundne* parter	DataLink
	5	Fysisk	fysisk medium, sender bits	Fysisk

Mellem de øvre lag sker der en logisk kommunikation, fysisk sendes data til det nederste lag, der så sender data videre. Data sendes som pakker med data og header (hvert lag tilføjer et nyt lag gavepapir og et nyt 'til & fra' kort).

5.1.4. Applikations lag

Mange protokoller. Standard: HTTP, SMTP, FTP, DNS. Speciallavede: Talk05V.

5.1.4.1. HTTP

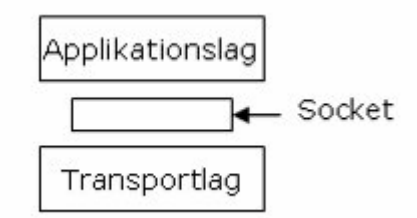
Lavet for en del år siden

Mellem web-server/browser. Stateless/tilstandsløs.

Kan bruge vedvarende forbindelser. Cookies. Autorisation.

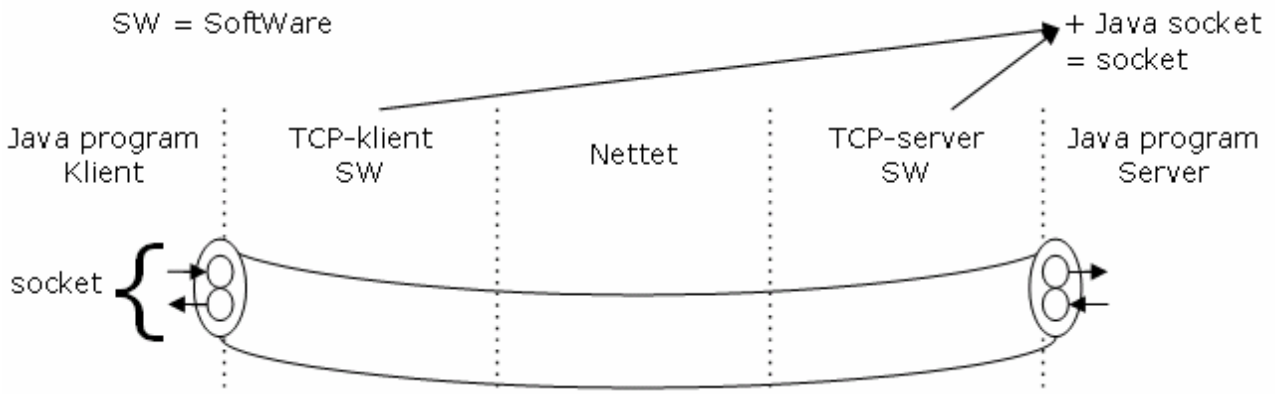
Headere (er tekst)

5.1.5. Socket programmering



Figur 8: Placering af socket

Gennemgang af TCP (og UDP) programmer (fra bogen).



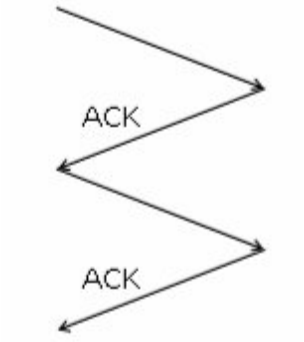
Figur 9: Hvordan Socket fungerer

5.1.6. Transportlaget, TCP & UDP

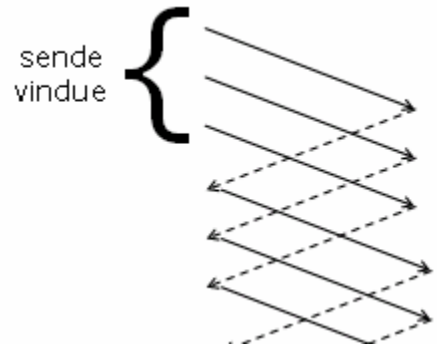
Se udleveret tabel (*multiplexing*, pålidelighed, congestion- & flow kontrol, segmentering, stream-/pakkeorienteret).

Fortæl hvilke faciliteter, der kan være og hvad de dækker over.

Stabilitet/pålidelighed: fortæl om sliding window (stop & vent (sende vindue = 1), reel)



Figur 10: Sliding window - stop og vent (for langsom)



Figur 11: Sliding window - reel metode

Go back n: modtager smider pakker ude af nummerorden væk

Selektive repeat: modtager gemmer alle korrekte pakker

5.1.6.1. TCP

Sessionsopsætning (3-way handshake)

Header (s. 217 fig. 3.29)

Modtagervindue: Hvor mange data der ønskes sendt i næste pakke

Bekræftelsesnumre: byte numre i stedet for pakkenummer (hvor langt de er nået)

Beregning af time-out: (RTT = Round Trip Time = net svartid)

Estimeret RTT = 0,875 * est. RTT + 0,125 * seneste RTT

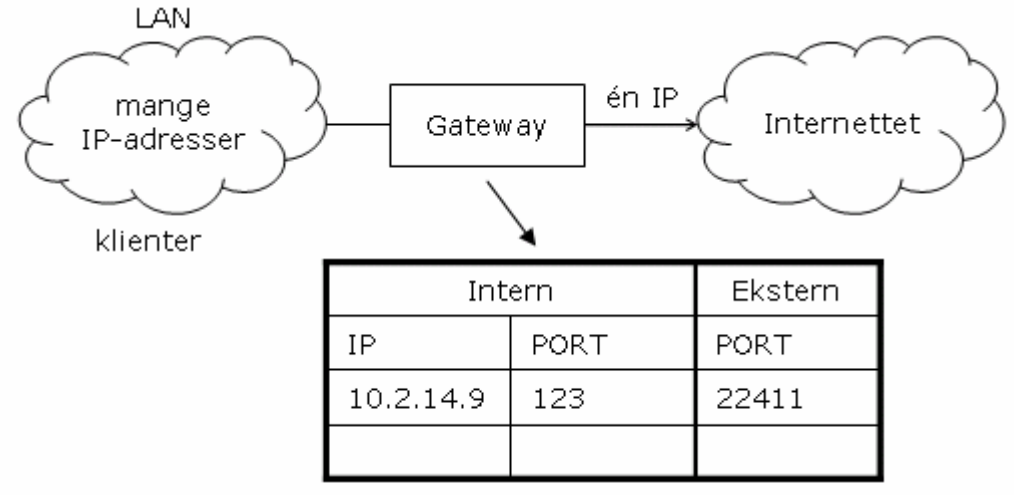
5.1.7. Netværks lag, IP og Rutning

Opgaver: Rutning, fragmentering

Virtual curcuit (rute fastlagt fra starten, reserverede ressourcer)/Datagram (IP)

Rutningsalgoritmer: korteste vej (kræver globalt overblik), distance vektor (rygter fra naboer)

IP: opbygning af IP-adresser (hierarkisk opbygget), IP-header (fig. 4.23), DHCP (uddeler IP-adresser dynamisk (på et lokalnet)), NAT



Figur 12: NAT protokollen

5.1.8. Datalinklaget

Opgaver

indpakning i rammer

kontrol med mediet (hvis det er delt)

stabilitet (pålidelighed) evt. sliding window

fejlkontrol (paritetsbit, CRC).

Ethernet (CSMA/CD)

Tidligere lå ledningerne bag PC'erne – nu går de igennem en switch, der til en vis grad også kan lave trafikseparering (se hvor ting skal hen og ikke sende dem til alle andre)

MAC-adresser + ARP-protokollen

For at undgå at alle maskiner bruger ressourcer på at hive pakker ind og sender dem op hvorpå de vil blive kasseret fordi det ikke er til egen IP.

ARP ligger som sådan presset ind imellem Netværkslaget og Datalinklaget.

5.1.9. Sammenkobling af net

Repeater + HUB → Fysisk lag (ingen buffer) – hovedsagelig forstærkning.

Alt sendes videre

Bridge + Switch → Datalink (buffer) – Arbejder med MAC-adresser.

Kan se om pakker skal videre

Router → Netværk (buffer) – Arbejder med IP-pakker

Learning bridge

Finder ud af hvor maskinerne ligger ved at kigge på afsender adresser.

Switch

Videreudvikling af bridge idéen. En bro med mange "sider".

Typisk i stand til at videresende en pakke inden pakken er helt modtaget.

Routere

Bufferstørrelse (stor nok til at tage udsving i trafikken, men pakkerne må ikke gemmes i for lang tid ide den så vil udsende forældede pakker)

5.1.10. Synkronisering i distribuerede systemer

Problemet: ingen fælles memory, intet fælles ur

5.1.10.1. Gensidig udelukkelse i distribuerede systemer

Central

Fastlægger en maskine der så opbevarer den fælles memory. Meget enkel løsning. Single point of failure, belastning.

Distribueret

Spørg alle om lov. Flot algoritme, men reelt svær at bruge.

Token

En stafet, der løber rundt. Problem i at være sikker på at der er én token.

Brug gerne Arnold-spillet som eksempel

5.1.11. (Fysiske arkitekturer)

Sprunget over i gennemgangen.

5.1.12. (Intro til distribuerede databaser)

Sprunget over i gennemgangen.

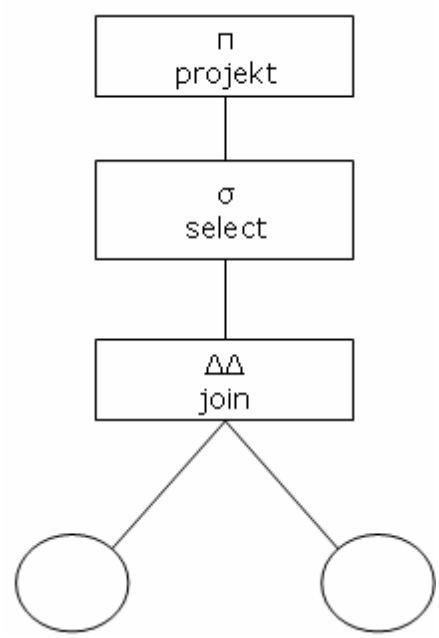
5.1.13. Optimering (af databaser)

Formler (kompendium udleveret med lektion 17): øverst s. 25, nederst s. 22, glem s. 17.

Problem: SQL er non-procedural (hvad og ikke hvordan)

5.1.13.1. Optimizer

Laver SQL om til en operatorgraf og laver ækvivalenstransformationer. (sker i alle relationelle databaser)



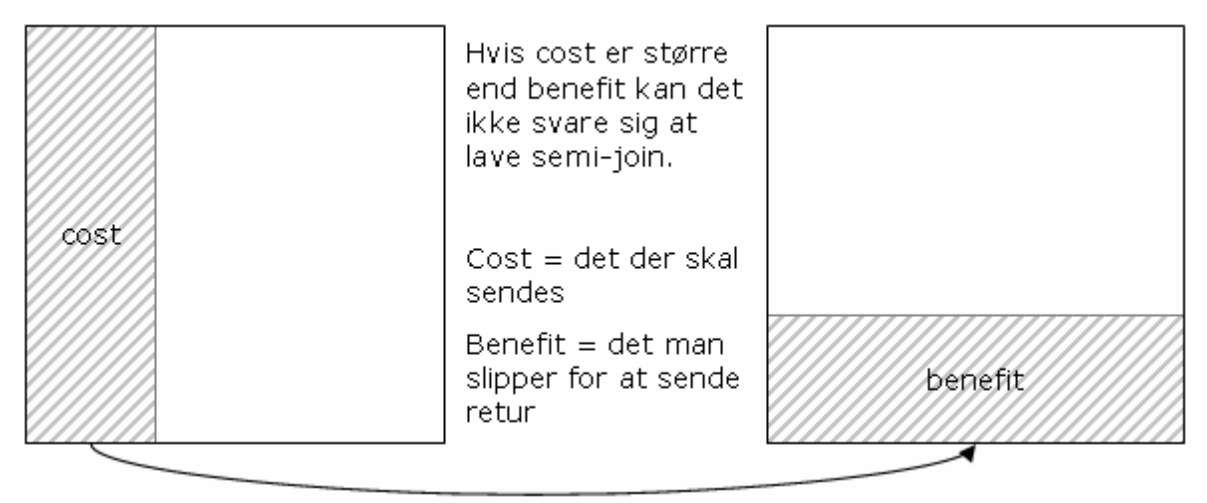
Figur 13: Operatorgraf

Central DB har som mål at have få disk I/O.

Distribueret DB har som mål at have få transmitterede bytes.

I visse situationer har optimizer brug for at vælge mellem flere strategier, den kan have brug for oplysninger om data! Resulterer i profil og estimeringsregler.

Husk også semi-join og cost-benefit.



Figur 14: Cost-benefit ved semi-join

5.1.14. Commitment protokoller & Replication

Hvorfor og hvornår skal den bruges?

Når der er flere databaser involveret og det ønskes at de gør det samme.

2-faset commitment: regler og fejlsituationer

3-faset commitment: fejlsituationer

5.1.15. Sikkerhed – generelt

Passwords & vira.

5.1.16. Net sikkerhed (inkl. Web-sikkerhed)

Kryptering, authensifikation, digital signatur, firewall og web-sikkerhed.