



Komplement till Computer Networkning

Innehållsförteckning

Förord	3
Kompendiet	3
Korrekturläsning.....	3
Författaren	3
Referensmodeller.....	4
Protokoll	4
OSI-modellen	5
Internet-modellen	18
Övningar på nodfördöjningar.....	32
Uppgifter	32
Lösningar.....	33
Övningar på Internet-checksummor.....	35
Uppgifter	35
Lösningar.....	36
Felövervakning.....	37
Introduktion	37
Omsändningsmetoder	39
Piggyback acknowledgement.....	50
Checksummor.....	51
Flödesreglering.....	64
Övningar på LS	68
Uppgifter	68
Lösningar.....	69
Lokala nät	70
Typiskt för lokala nät	70
Topologier	72
Principer för accessmetoder	76
Ethernet	82
Fältbussar.....	92
Inledning.....	92
Principen master/slave och accessmetoder.....	94
Fältbussmodell	96
CAN	98
Profibus	107
Sammanställning av några fältbussar	113

Förord

Kompendiet

Kompendiet innehåller komplementerande material till kursboken *Computer Networking – A Top-Down Approach*, av James Kurose och Keith Ross. Komplementeringen är anpassad till kursen **Datorkommunikation och nät** vid Örebro universitet.

Korrekturläsning

Ett varmt tack till fil.mag. Christer Lindkvist för korrekturläsning och förslag till förändringar. Korrekturläsningen gjordes av en större version av kompendiet. Den versionen kallas **Datakommunikation för ingenjörer – Grundläggande teori för modern kommunikation**. Christer har mångårig och djup erfarenhet av datorer och datakommunikation både i egenskap av datalärare på universitetsnivå och nätverksadministratör vid Örebro universitet.

Undantaget från korrekturläsningen är avsnittet **Övningar på nodförförjningar** som finns inte med i ovanstående kompendieversion.

Författaren

Jack Pencz är gymnasieingenjör (fyraårig teleteknisk linje) och civilingenjör (teknisk fysik och elektroteknik). Utöver dessa grundutbildningar har han läst kurser i lödningsteknik, pedagogik, forskning (doktorandkurser i tillämpad elektronik, data teknik och vetenskapsteori), matematik och teologi.

Jack har varit anställd vid bl.a. AB Bofors, Linköpings universitet, Ahlström Automation AB och Ericsson Mobile Communications AB. Nu är han adjunkt i data teknik vid Örebro universitet där han började 1997 som timlärare i elektronik och programmering (vid den dåvarande Högskolan i Örebro). Jack var under två år programansvarig för Dataingenjörsprogrammet, var under flera år ledamot av den tidigare institutionsnämnden och var under några år (t.o.m. 2006) vice ordförande i institutionens styrelse.

För övrigt har han varit anställd som elektronikkontrollant, elektronikingenjör, serviceingenjör (styr- och reglertechnik i processindustrin), nätverksadministratör, vik. forskningsingenjör, lärarassistent (tillämpad elektronik vid universitet) och vik. adjunkt (elteknik och data vid gymnasium).

Under anställningarna som lärare har Jack undervisat blivande gymnasieingenjörer, IT/PC-samordnare, högskoleingenjörer, teknologie magistrar, civilingenjörer m.fl. Jack är författare och medförfattare till flera vetenskapliga artiklar och rapporter. Dessutom har han skrivit kompendier samt utvecklat övningar/laborationer för både gymnasium och universitet. Under sina anställningar har Jack utvecklat dataprogram för tekniskt avancerade applikationer som har använts inom både forskning och utbildning.

Referensmodeller

Protokoll

För att kommunikationen ska fungera i datanät, krävs speciella program och hårdvaror. Sådana beskrivs av **protokoll**. Ett protokoll är helt enkelt en beskrivning över hur något ska fungera. Protokoll skrivs som bekant under möten för att man ska komma ihåg de beslut som har fattats, t.ex. under ett föreningsmöte. I detta fall är det inte möten som dokumenteras. Istället är det beslut om hur datautrustning ska fungera under kommunikation som skrivs ned.

I många fall utfärdas protokoll som standarder (rekommendationer). Fördelen med att utfärdta protokoll som standarder är att programmerarna och hårdvarutillverkarna får vetskapp om hur produkter ska fungera för att passa ihop med annan utrustning. För oss konsumenter innebär detta att vi kan köpa datautrustning som passar för sitt ändamål. Annars är detta inte givet. Utan standarder skulle det kunna finnas lika många protokoll för ett ändamål som det finns tillverkare.

De protokoll som är generella brukar kallas öppna. Man talar om **öppna system** som innebär att oavsett tillverkare av program och hårdvara så fungerar produkten i sitt sammanhang. Några exempel är protokollen i **Internet**, det lokala nätet **Ethernet** och persondatorns **Universal Serial Bus (USB)**.

Motsatsen är **slutna system** (tillverkarspecifika system). Det finns en rad nät som är tillverkarspecifika (plug-compatible systems). För att kunna använda sådana, krävs att man har utrustning som just den aktuella tillverkaren har tagit fram. Tillverkarna har tagit fram bl.a. System Network Architecture (SNA) från IBM, Distributed Network Architecture (DNA) från Digital Equipment, Distributed Communication Architecture (DCA) från Univac och Distributed System Architecture (DSA) från Honeywell Bull.

Sammanfattningsvis kan följande sägas om protokoll för datakommunikation:

- Protokoll är i detta fall beskrivningar över hur kommunikation ska fungera. Alla datorer (parterna i samtalen) måste ”tala samma språk”.
- Protokoll kan skrivas ned i olika former: klartext, tillståndsgraf, tillståndstabell, pseudoprogram (program som är skrivna med klartext) och program som är skrivna med något programspråk.
- Protokollen realiseras (förverkligas) oftast i form av program men även hårdvaror förekommer (i synnerhet på fysisk nivå).

OSI-modellen

På 1970-talet beskrev **International Organization for Standardization** (ISO) en modell för samarbetande protokoll som kallas **Open Systems Interconnection** (OSI). Denna ska betraktas som en **referensmodell**, dvs. en beskrivning av protokollstackarnas principiella struktur. Protokollen finns i **sju skikt** som tillsammans bildar en **protokollstack**. Varje enhet som t.ex. en dator har vanligtvis en protokollstack (i vissa fall flera).

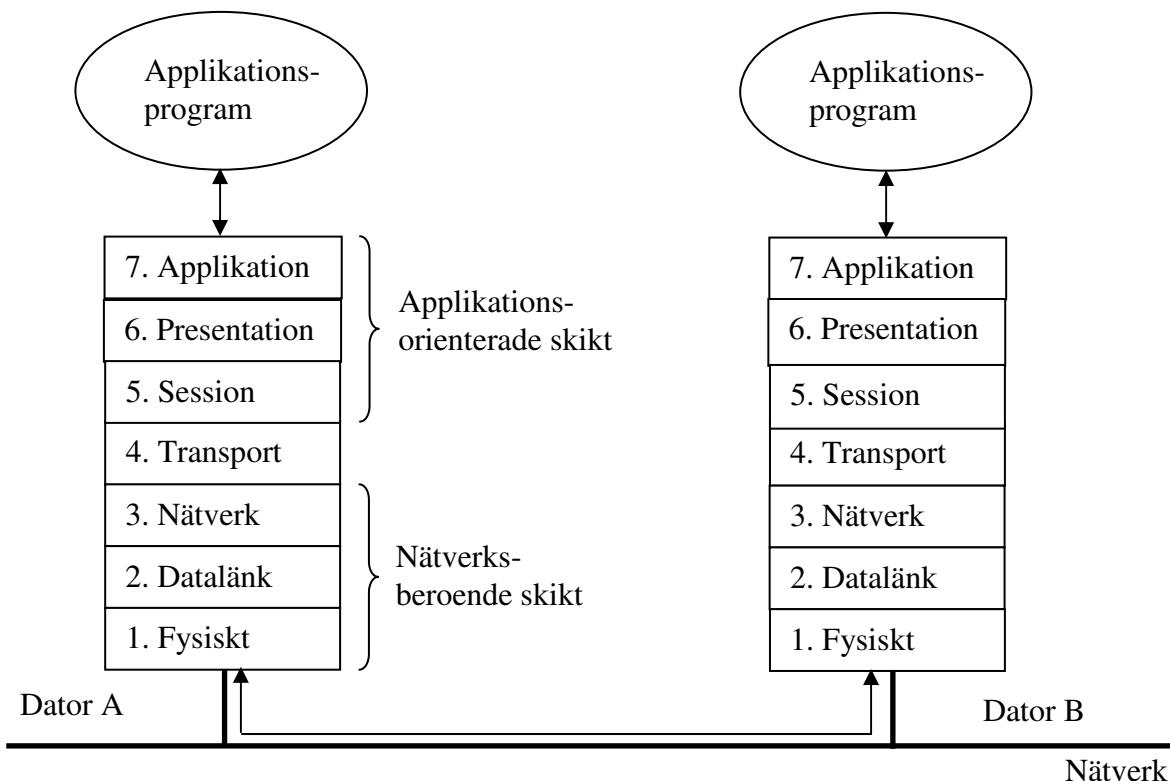
OSI kan användas för olika typer av protokoll. Det är alltså inte de enskilda protokollen som beskrivs av OSI. Istället beskriv protokollens huvudfunktioner i varje skikt samt hur protokollen ska kommunicera med varandra inom en protokollstack och mellan protokollstackar.

Innan vi beskriver de sju skikten i OSI-modellen är det på sin plats att ange fördelarna med att överhuvudtaget dela upp i skikt. Innan man utvecklade protokollstackar kunde kommunikationsprotokollet skrivas som ett enda stort program. Detta program omfattade allt och kunde hantera alla tänkbara situationer. För att datorerna skulle kunna kommunicera krävdes kompatibla kommunikationsprotokoll. Dessa kunde till och med vara versionsberoende. Genom att dela upp detta stora kommunikationsprogram i flera delar (skikt) vann man följande fördelar:

- Reducerad komplexitet vid design
- Uppgradering eller förändring av ett skikt kan göras utan att andra skikt berörs

Eftersom både protokoll och tjänster finns i skikten, kan man ställa sig fråga om de har med varandra att göra. Det har de naturligtvis. Protokollen beskriver nämligen exakt hur tjänsterna ska utföras. Tjänsterna finns alltså realiserade i protokollen. Dessutom är gränssnitten mellan skikten i OSI-modellen väldefinierade. Inget lämnas åt ”slumpen” i OSI-modellen.

Eftersom OSI-modellen beskriver protokollens huvudfunktioner i varje skikt, medger detta (i vissa fall) att protokollen som används i två kommunicerande datorer kan vara olika versioner. Det viktiga är att kommunikationen fungerar inom varje protokollstack och mellan dessa. Naturligtvis kan man inte använda vilka protokoll som helst, men OSI-modellen medger stor variation i användandet av protokoll.



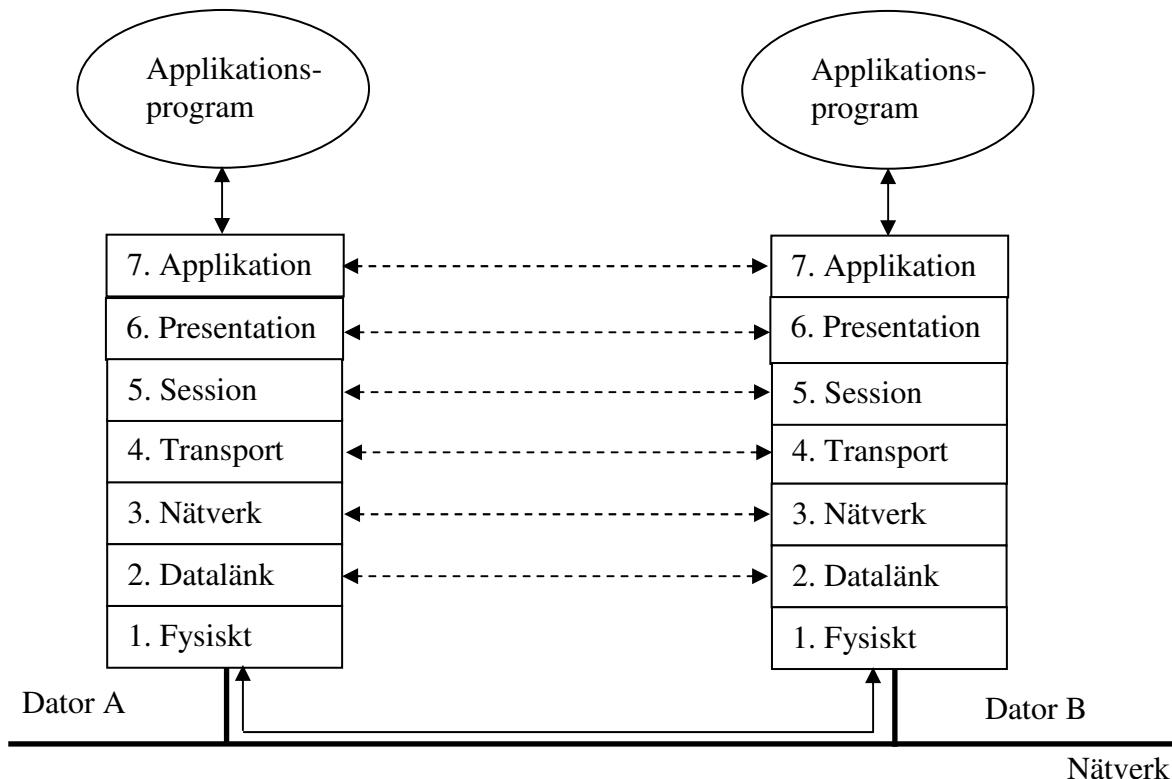
Figur 1. Två applikationsprogram som kommunicerar med hjälp av protokollstackar (OSI).

I figur 1 visas två applikationsprogram som kommunicerar med hjälp av protokollstackar som har sju skikt vardera. De tre nedersta skikten är beroende av nätverkstypen och de tre översta är inriktade på att ge tjänster åt applikationsprogrammet. Det är applikationsprogrammen i respektive dator som samtalar. Applikationsprogram kan vara t.ex. webbläsare, webbserver, nätverksklient, filserver, filhanterare (t.ex. Utforskaren i Windows-system), filtransportprogram (klient/server), databasklient, databasserver, utskrivande applikationsprogram och utskriftsserver.

Applikationsprogrammen kommunicerar med varandra med hjälp av protokollen som finns i protokollstackarna. Det är den huvudsakliga kommunikationen. Dessutom kommunicerar protokollet i ett skikt med protokollet i grannskicket för att skicka meddelandet vidare. I en protokollstack som sänder, skickas meddelandet nedåt från skikt till skikt. Naturligtvis sker det motsatta i en protokollstack som tar emot ett meddelande, dvs. meddelandet skickas uppåt skikt för skikt.

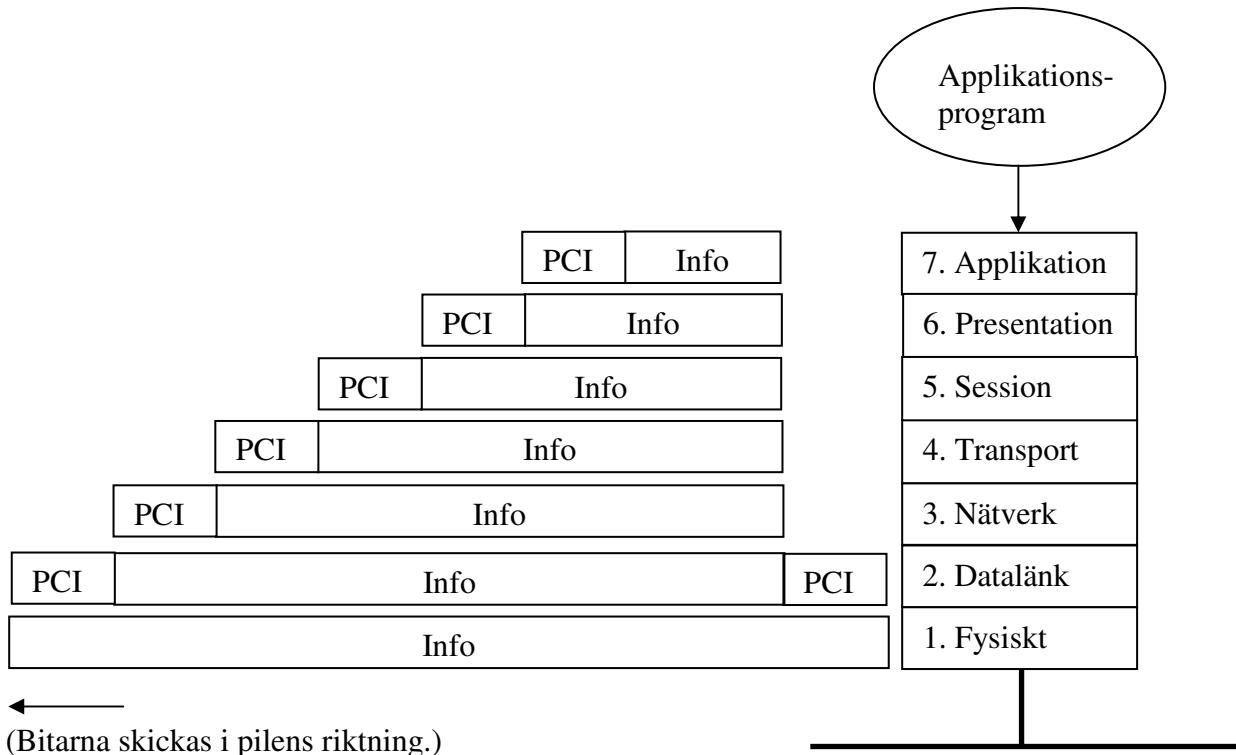
Dessutom skickas extra information med i meddelandet från nästan varje skikt i den sändande protokollstacken. Det är endast det fysiska skiktet som inte skickar med någon extra information. Sådan information benämns **kontrollinformation (Protocol Control Information, PCI)** och används av motsvarande skikt i den mottagande protokollsstacken. I de flesta fall placeras PCI först i det meddelande som skiktet vidarebefordrar. Därför brukar detta kallas **header**. Principen att skicka kontrollinformation till motsvarande skikt hos mottagaren illustreras i figur 2 med streckade pilar.

Kontrollinformation behövs för att motsvarande skikt hos mottagaren ska få instruktioner för behandling av data i meddelanden. Datalänkskiktet lägger till två PCI-fält, ett vanligt för kontrollinformation (header) och ett för **ramens checksumma (trailer)**. Checksumman används av mottagaren för att undersöka om bitarna i meddelandet har överförts utan fel.



Figur 2. Varje skikt utom det fysiska skickar kontrollinformation till motsvarande skikt hos mottagaren.

Eftersom skikten lägger till kontrollinformation (PCI), kommer meddelandena att växa i storlek. Detta visas i figur 3. Formellt kallas ett meddelande för **Protocol Data Unit** (PDU) med en bokstav framför som anger vilket skikt som avses, t.ex. PPDU som betyder det meddelandet som sätts samman i presentationsskiktet. Meddelanden brukar populärt kallas **bitar** i det första skiktet, **ramar** i det andra skiktet och **paket** i det tredje skiktet.



Figur 3. Varje skikt utom det fysiska sätter samman en PDU.

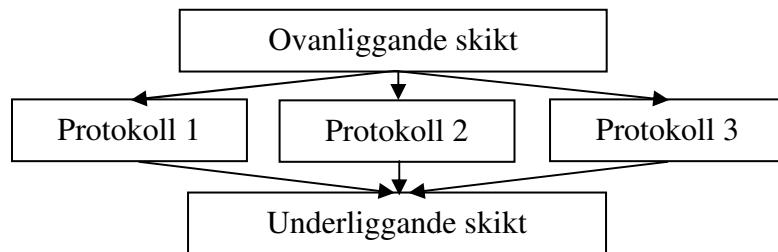
Ett skikt har till uppgift att tillhandahålla **tjänster** som de närmaste grannskikten kan begära. I den sändande protokollstacken är det skiktet ovanför som kan begära tjänster och i den mottagande protokollstacken är det skiktet under. I OSI-modellen byggs sådana tjänsteuppsättning upp av **fyra grundprimitiver**. Det är följande grundprimitiver:

- Request (begäran)
- Response (svar på begäran)
- Indication (indikering om inkommande begäran)
- Confirm (bekräfelse på en begäran)

Dessa grundprimitiver används för att skapa de **primitiver** som utgör **tjänsteuppsättningarna** som t.ex. CONNECT-uppsättningen med **CONNECT.request**, **CONNECT.response**, **CONNECT.indication** och **CONNECT.confirm**. Andra exempel är uppsättningar för DATA, DISCONNECT, LISTEN, SEND och RECEIVE. För att ange vilket skikt som tillhandahåller tjänsten, skriver man in skiktets begynnelseläge, t.ex. **T_CONNECT.request**, där T står för transportskiktet. (I praktiken är detta ett exempel på en funktion som kan väljas i de program som utgör protokoll och gränssnitt.)

Eftersom både protokoll och tjänster finns i skikten, kan man ställa sig fråga om de har med varandra att göra. Det har de nämligen. Protokollen beskriver exakt hur tjänsterna ska utföras. Tjänsterna finns alltså realiserade i protokollen. (Även gränssnitten beskrivs i detalj av

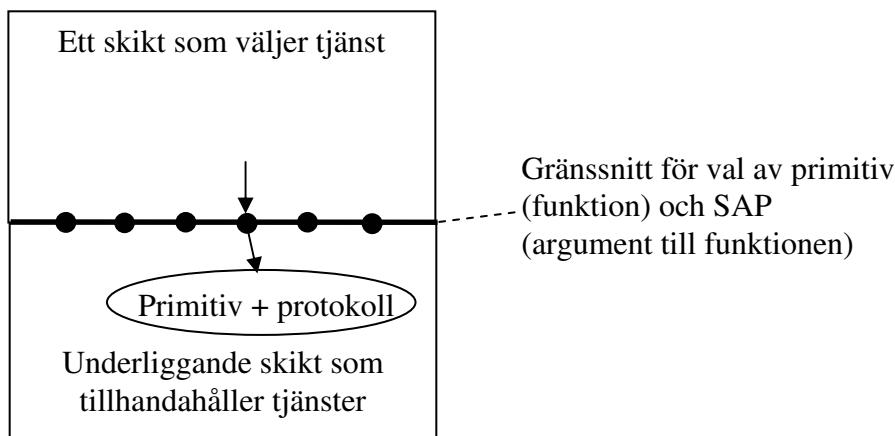
protokoll.) Eftersom OSI-modellen beskriver det som kan (eller ska) göras i ett skikt, har detta med de generella tjänsterna (primitiverna) att göra. Därmed kan man inte exkludera begreppet tjänst. Hur tjänsterna ska utföras preciseras däremot av protokollen. Respektive protokoll bestämmer alltså exakt hur tjänsterna ska utföra. För att kunna utföra tjänsterna på olika sätt, kan flera protokoll samsas i samma skikt. Principen visas i figur 4.



Figur 4. Flera protokoll i samma skikt.

Om det kan finnas flera tjänster och även flera tjänsteuppsättningar i ett skikt, måste man kunna begära vilken tjänst som ska utföras när ett meddelande skickas till nästa skikt. Tjänsteuppsättningarna realiseras av protokollen och det kan också finnas flera protokoll i samma skikt. Val av primitiv i ett gränssnitt ger bestämd typ av tjänst och det protokoll som önskas anges i valet av **Service Access Point** (SAP). Sådana kan betraktas som portar med olika nummer (deladresser). Genom att välja en primitiv och en SAP, väljs tjänsttyp och protokoll i grannskiktet. (SAP anges som argument till funktionen för den önskade primitiven.)

Om en SAP befinner sig i **gränssnittet** mellan det fjärde och det femte skiktet, kallas denna TSAP, där T står för transportskiktet som är det fjärde skiktet. (Service access points i andra gränssnitt betecknas efter samma princip.) De service access points som betecknas med TSAP är av speciellt intresse i Internet-modellen. Det är val av TSAP som ger önskat applikationsprotokoll. Se figur 5. Eftersom ett applikationsprotokoll är integrerat med tillhörande program (applikation), ger val av TSAP indirekt val av program hos mottagaren. Ett exempel är port (TSAP) 80 som vanligtvis används för applikationsprotokollet HTTP. Detta kan vara integrerat med webbläsare och webbservrar som är exempel på tillhörande program. För att garantera att kommunikationen ska fungera inom en protokollstack, skiljer OSI-modellen noggrant på tjänst, gränssnitt och protokoll. I Internet-modellen är åtskillnaden inte fullt så tydlig.



Figur 5. Gränssnitt.

Den fullständiga adressen för service access points inom en och samma protokollstack kallas **AP-adress** (Access Point). Den sätts samman av de nödvändiga portnumren (deladresserna) genom konkaterning så att

$$\text{AP-adress} = \text{PSAP} + \text{SSAP} + \text{TSAP} + \text{NSAP}$$

Trots skrivsättet betyder + inte addition utan hopsättning av strängar (konkatenering). AP-adressen omfattar varje önskad SAP från gränssnittet 7-6 (PSAP) ned till gränssnittet 4-3 (NSAP). NSAP är nätverksadressen som finns i det tredje skiktet. Denna kan t.ex. vara en IP-adress. I praktiken är vanligtvis PSAP = SSAP. Följande exempel beskriver ett tjänsteanrop från sessionsskiktet till transportskiktet:

```
T_CONNECT.Request(anropad adress, anropande adress, ..., SD)
```

Parametrarna är i detta exempel bl.a. anropad adress, anropande adress och SD. I detta exempel är en adress varje SAP i gränssnitten nedanför sessionsskiktet (5-4 och 4-3), dvs. TSAP + NSAP. Anropad adress gäller i mottagarens protokollstack och anropande adress gäller i avsändarens.

Service Data Unit (SDU) är det ovanliggande skiktets PDU, dvs. det ovanliggande skiktets PCI konkaternerad med sin **info**. Se figur 3. I exemplet med tjänsteanropet är SDU i transportskiktet lika med PDU från sessionsskiktet. SD är pekaren till den minnesbuffert som innehåller SDU.

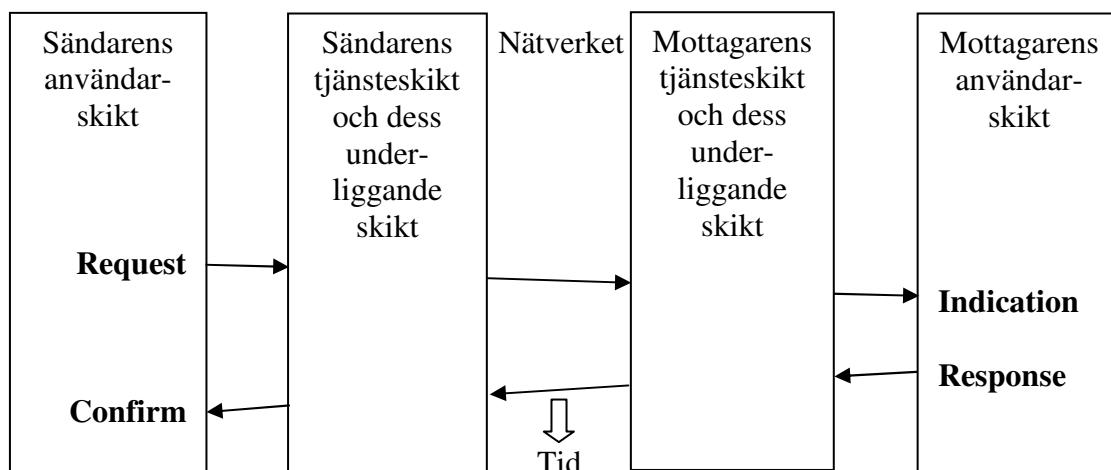
Om ett skikt har gjort uppkoppling, finns inget behov av adressparametrar. Ett exempel på tjänsteanrop efter uppkopplingen är följande:

```
T_DATA.indication(förbindelsenummer, SD)
```

De grundläggande primitiverna visar tydligt principerna för kvitterad och okvitterad tjänst. I figur 6 visas hur ett skikt (användarskiktet) anropar sitt närmaste, underliggande skikt och begär på detta sätt att få en tjänst utförd. Tydligen är det en tjänst som bygger på den grundläggande primitiven request. Vi behöver inte känna till den typ av tjänst (primitiv) som det är fråga om. Samtliga behandlas på liknande sätt. Tjänsten utförs av det underliggande skiktet (tjänsteskiktet) som i sin tur anropar sin underliggande granne, osv. Resultatet blir att det fysiska skiktet skickar ut bitarna i den ram som datalänkskiktet har satt ihop.

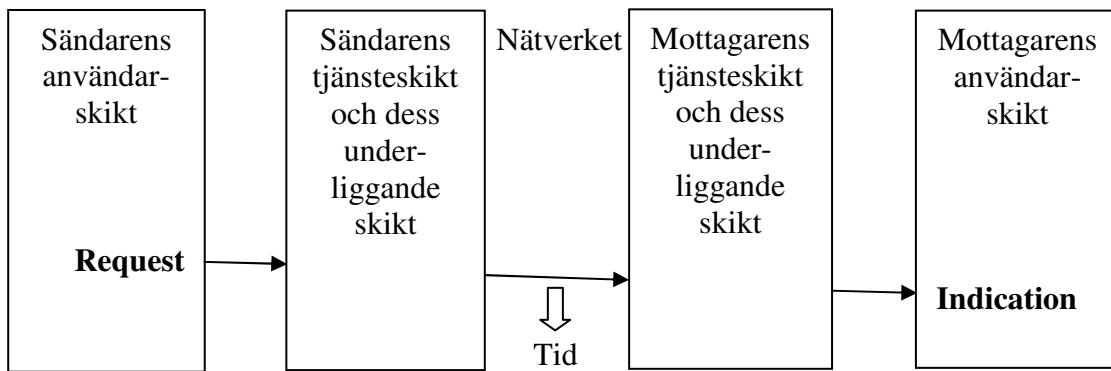
Bitarna tas emot av mottagarens protokollstack. I denna görs transport uppåt genom successiva anrop av det närmaste, överliggande skiktet. När motsvarande tjänsteskikt i mottagarens protokollstack mottar PDU från sin underliggande granne, görs anrop till användarskiktet. Den grundläggande primitiven i detta anrop är indication. Mottagarens användarskikt utför begärd tjänst och sätter ihop ett svar till sändarens användarskikt. Den grundläggande primitiven som mottagarens tjänsteskikt använder i svaret är response. Svaret behandlas på liknande sätt som anropet och resulterar till sist i ett anrop med confirm i sändarens användarskikt. På detta sätt får sändaren bekräftat att tjänsten har utförts av mottagaren.

I detta sammanhang är det lätt att glömma bort att det egentligen är applikationsprogrammen som kommunicerar. Principen i figur 6 visar alltså en del av denna kommunikation, nämligen utbyten som sändarens skikt har med respektive skikt hos mottagaren. I denna figur är det kvitteringen som är det väsentliga.



Figur 6. Principen för kvitterad tjänst.

I figur 7 visas princiken för okvitterad tjänst. Denna är enklare än den kvitterade tjänsten eftersom användarskiktet i mottagarens protokollstack inte skickar svar tillbaka till användarskiktet hos mottagaren.



Figur 7. Principen för okvitterad tjänst.

OSI-modellen har specificerat hur de sju skikten ska användas i protokollstacken. Specifikationen är följande:

Det fysiska

Detta skikt beskriver kablar, kontakter, elektroniska kretsar och elektriska egenskaper. Man brukar kallas sådant för **mekaniska och elektriska gränssnitt**. Det överför **bitarna** som finns i datalänkskiktets ram. Här ingår också protokoll för **upp- och nedkoppling** samt **styrning** av den fysiska förbindelsen.

Datalänk (länk)

Detta skikt använder de fysiska länkarna mellan noderna i nätet och ger **säker överföring**. Det lägger in paket i **ramar** för sändning och hos mottagaren görs det motsatta. Protokoll på denna nivå kallas ofta **linjeprocedurer**. På denna nivå används **hårdvaruadresser**, s.k. **MAC-adresser** som t.ex. **Ethernet-adress**. MAC är akronymen för **Media Access Control** eller (**Medium Access Control**). MAC-adresserna finns vanligtvis i hårdvaran, t.ex. på datorns nätverkskort, där de anges med logiska nollar och ettor.

Förutom hårdvaruadressering hanterar linjeprocedurerna **synkronisering**, **felövervakning** och **flödesreglering**. Att mottagaren är synkroniserad innebär att den kan sampla insignalen i vid rätt tidpunkter. Felövervakningen innebär bl.a. att ramarnas sekvensnummer kontrolleras och att ramarnas checksummor används för att avgöra om deras innehåll är att lita på. Flödesreglering kan göras av antingen mottagaren eller sändaren. På länknivå kan mottagaren skicka kontrollsinyaler på speciella ledare i kabeln eller kontrolltecken på dataledaren. Om det är sändaren som reglerar flödet, kan detta göras genom kontroll av mottagarens svar inom bestämd tid. Den senare metoden används också i transportskiktet, i synnerhet då TCP används för transport över Internet.

Nätverk (nät)

I detta skikt styrs informationsflödet med ev. **prioritering**, **multiplexering** (flera samtal delar på tillgängligt utrymme), **segmentering** (uppdelning av meddelanden i mindre paket) och **blockning** (hopsättning av paket till kompletta meddelanden).

Vissa protokoll har **nätverksadresser** på denna nivå, t.ex. **IP-adresser** för kommunikation både lokalt i nätet och över Internet. Nätverksskiktet ansvarar för paketens **vägval** över nätet.

I detta skikt väljs antingen **förbindelseorienterad tjänst** (uppkoppling görs innan det egentliga meddelandet skickas) eller **förbindelselös tjänst** (ingen uppkoppling innan det egentliga meddelandet skickas). I Internet-modellen kan endast det förbindelselösa IP användas. Detta avviker alltså från reglerna för OSI-modellen. Om tjänsten i OSI-modellens nätverksskikt är förbindelselös, är det istället transportskiktets uppgift att hantera felövervakning. (Jämför med TCP i Internet-modellens transportskikt.) Förbindelseorienterad tjänst medför felövervakning i nätverksskiktet.

Transport

Detta skikt ger transport mellan datorer. Det kopplar ihop de övre skikten (5–7, de **applikationsorienterade**) med de lägre (1–3, de **nätverksberoende**). Skiktet ger **pris/prestanda-optimerad nätoberoende transportförbindelse**. (Fler-förbindelser kan förekomma.) Det kan ge **felövervakning** och **flödesreglering** av olika kvalitet.

Skiktet kan kompensera för olika **Quality of Service** (QoS) i de nätoberoende skikten. Därför erbjuds tjänster i fem olika klasser, från enbart basfunktioner (klass 0) till full felövervakning och flödesreglering (klass 4).

Session

En session är en **logisk förbindelse** mellan applikationsprogrammen i datorerna. Flera sessioner kan finnas samtidigt mellan två datorer. En dator kan också ha samtidiga sessioner till olika datorer. Sessionsskiktet **kopplar upp och ned** sådana sessioner samt **styr** och **övervakar** dialogerna. En session kan upprätthållas så att applikationsprogrammet inte märker tidvis nedkoppling på transportnivån. Denna funktion är bra på nät som använder tiddelar.

Några av tjänster är följande:

Synkronisering och **avgränsning** av sammanhörande data vid halv duplex (dubbelriktad trafik fastän inte samtidigt).

Felrapportering av sådana fel som inte hanteras av sessionsskiktet.

Hjälp vid omstart efter fel. Här kan **synkroniseringsbrytpunkter** användas. Detta innebär att man efter omstart fortsätter efter den senaste brytpunkten (innan felet).

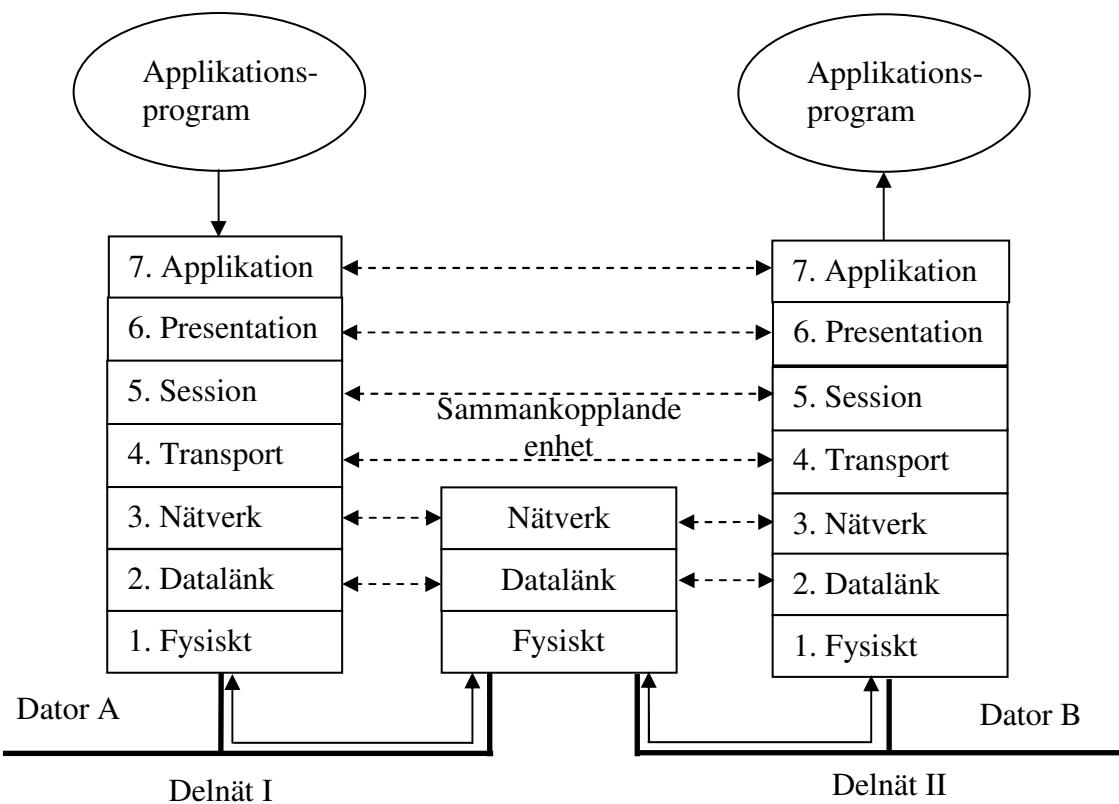
Token management kan användas då protokollen hos sändare och mottagare inte får utföra samma operation samtidigt. Endast den part som för tillfället har token (stafettpinnen) får utföra den känsliga operationen.

Presentation

Detta skikt förhandlar om rätt **överföringssyntax** och gör **konverteringar** för t.ex. kod, tecken och dataformat (för datum, valuta etc.). Presentationsskiktet kan begära upp- och nedkopplingar av sessioner, **komprimera** och **kryptera**.

Applikation (tillämpning)

Detta skikt tillhandahåller **applikationsprotokoll** (datakommunikationstillämpningar) för applikationsprogrammen, t.ex. för elektronisk post och filöverföring. Skiktet **styr** det totala systemets processer och resurser. Det finns flertalet tjänster i detta skikt. Dessa ger inställningar för användandet av det totala systemet, dvs. även för underliggande protokoll i stacken.



Figur 8. Protokollstack i sammankopplande enhet.

Sammankopplande enheter har vanligtvis inte alla sju skikt i sina protokollstackar. Det är nödvändigt att ha med det fysiska skiktet för att kunna ta emot bitar och sända bitar.

Repeterare, hubbar och ringkoncentratorer är exempel på sammankopplande enheter som endast har detta första skikt.

Antalet skikt förutom det första skiktet har att göra med den sammankopplande enhetens funktioner. Om den sammankopplande enheten kan adressera (välja väg) med hjälp av på MAC-adresser, omfattas också det andra skiktet. **Bryggor** och **switchar** är exempel på sammankopplande enheter som omfattar det första och det andra skiktet. (Då kallas adresseringen bridging respektive switching.)

Routrar kan göra routing (vägval) på adresser som finns i nätverksskiktet, dvs. sådana omfattar de tre nedersta skikten. Ett exempel är routrar på Internet som gör routing med avseende på IP-adressen. Routing görs på intelligentare sätt än bridging/switching genom att välja länkar med minst trafikbelastning, minst antal hopp (eng. *hop*, minst antal routrar längs vägen) mellan avsändare och mottagare, etc. (Router kallades förr **gateway**. Därför benämns företagets/organisationens router mot omvärlden för **default gateway** eller **standard-gateway**. Routrar kan vara utrustade med **brandväggsfunktioner**.)

Den sammankopplande enhet som man numera kallar **gateway** omfattar liksom persondatorer samtliga skikt. Vanligtvis är det en persondator med speciellt program för protokollkonvertering. En gateway behövs när hela protokollstacken ska ändras för att kunna kommunicera utåt med t.ex. speciella e-postnät, stordatorer och minidatorer. (Stordatorer kallas i engelsk litteratur för *main frames*.) En gateway har en typ av protokollstack mot det lokala nätet och en annan typ mot omvärlden. Förr kallades dessa sammankopplande enheter för **protocol converters**.

När det finns minst en sammankopplande enhet mellan två datorer som kommunicerar, skickas inte kontrollinformation (PCI) direkt från alla skikt till motsvarande skikt hos mottagande dator. Den sammankopplande enheten kommer att fungera som mottagare och sändare för de skikt som denna enhet omfattar. Detta illustreras i figur 8 med pilar till och från den sammankopplande enhetens skikt.

Sammankopplande enheter

Det första skiktet

Repeterare: Förstärkare för att förlänga bussar.

Hubb: Nav i stjärnnät som realiseras logisk busstopologi. Kan i motsats till repeterare stänga av portar som ansluter trasiga nätverkskort.

Ringkoncentrator (kabelkoncentrator): Nav i stjärnnät som realiseras logisk ringtopologi.

Up till och med det andra skiktet

Brygga: Kopplar ihop lokala nät utan att höja den totala trafiken. (Kopplar endast ihop två portar vid behov.) Läser MAC-adresser.

Switch: Nav med bryggfunktion i stjärnnät.

Upp till och med det tredje skiktet

Router: Läser paketens nätverksadresser (t.ex. IP-adresser) och kopplar, liksom bryggan, ihop två portar endast vid behov. Har förmågan att kunna välja bästa väg (port).

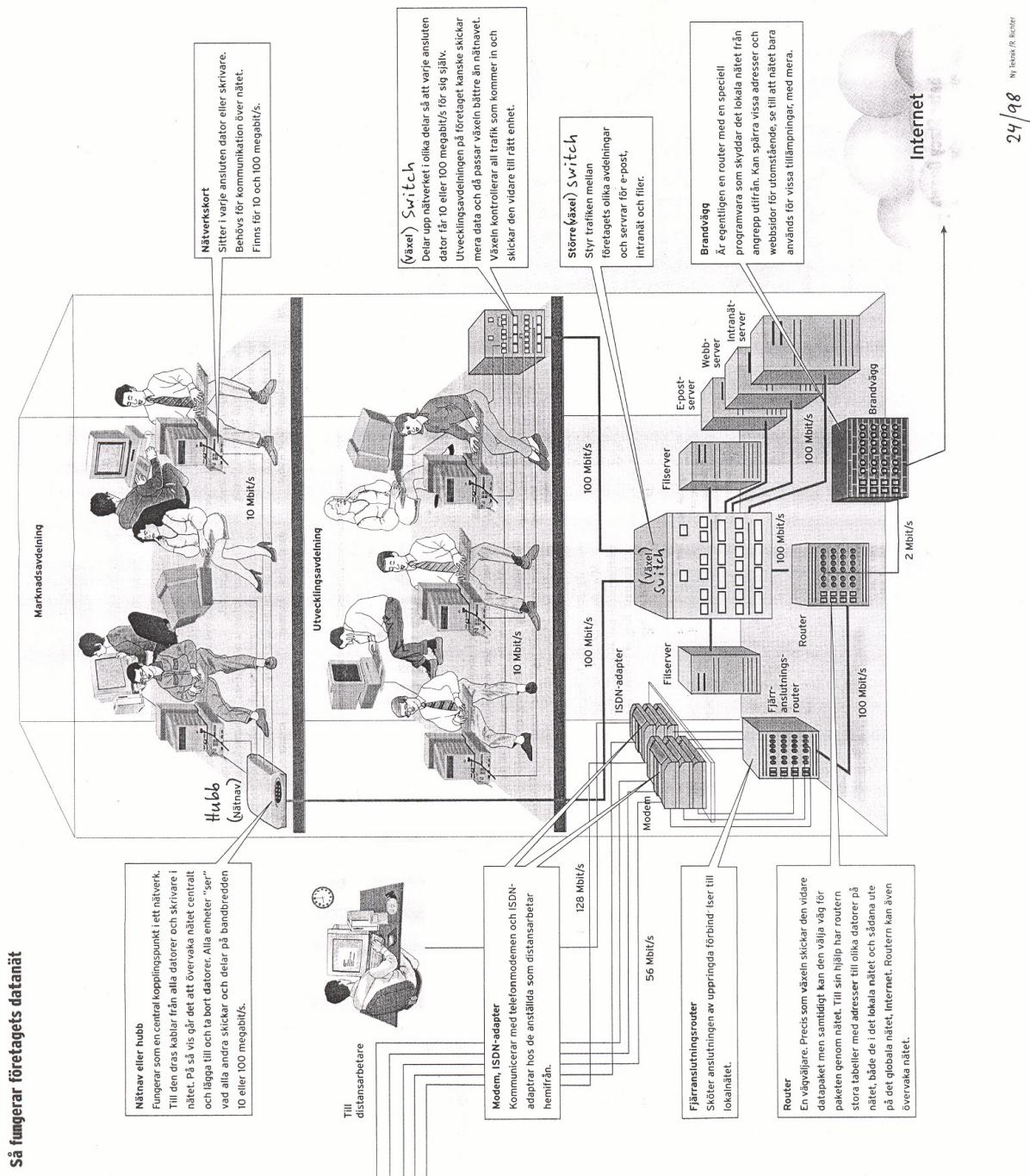
Upp till och med det sjunde skiktet

Gateway: Är vanligtvis ett program i någon av persondatorerna. Ger i kombination med modem (eller liknande utrustning) förbindelser till e-postnät, centrala datorer (stor- och minidatorer) m.m.

I figur 9 visas ett lokalt nät med sammankopplande enheter och servrar hos ett företag.

Standarder för OSI-modellen

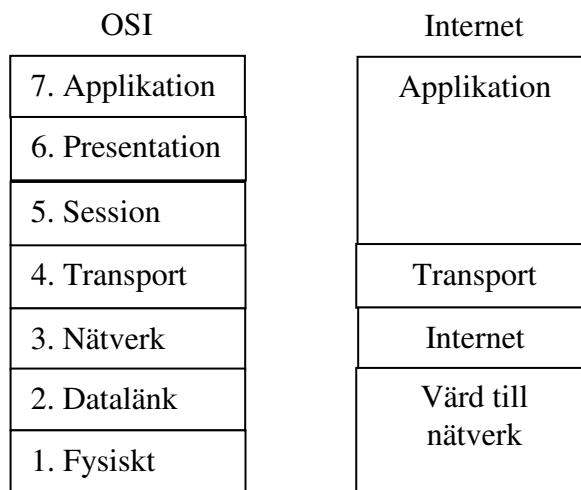
Man skulle kunna tro att det enbart är ISO som har tagit fram protokoll för OSI-modellen. Om detta antagande är rätt, begränsas användningsområdet. Dessbättre stämmer detta inte med verkligheten. Många protokoll som tas fram kan katalogiseras med hjälp av skikten i OSI-modellen. I litteraturen finns det figurer där man har placerat in de rekommenderade serierna (i synnerhet X, T, U och I) från ITU-T och 802-serien från IEEE i OSI-modellen. De flesta modeller över nätverk och nätverksoperativsystem jämförs med OSI-modellen för att man ska förstå dessa. Att OSI-modellen är en **referensmodell** innebär alltså att den både är ett **ramverk för design av protokollstackar** och en **referens** för jämförelse av det mesta inom datakommunikation. Därför ska vi jämföra Internet-modellen (TCP/IP-modellen). Senare kommer även det lokala nätet Ethernet att jämföras med OSI-modellen.



Figur 9. Ett företags lokala nät.

Internet-modellen

Protokollen som används på Internet kan placeras i skikt som liknar de som finns i OSI-modellen. Man kan kalla denna modell för **Internet-modellen** eller **TCP/IP-modellen**. Liksom för OSI är den korrekta benämningen **referensmodell**.



Figur 10. Internet-modellen i jämförelse med OSI-modellen.

Både OSI-modellen och Internet-modellen bygger på protokollstackar och somliga av skikten är likvärdiga. Däremot har Internet-modellen inte lika skarpa gränser mellan tjänster, gränsnitt och protokoll. Detta har sin förklaring i att Internet-modellen togs fram genom praktiska försök medan OSI-modellen är en ”skrivbordsprodukt”. Dessutom används inte samma uppställning av förbindelsetyper i de båda modellernas skikt. Det är i nätverks-/Internet-skiktet och transportskiktet som detta märks. Se tabell 1.

OSI-modellens två nedersta skikt brukar slås samman i Internet-modellen eftersom dessa inte har med Internet-protokoll att göra. Protokollen i dessa skikt bestäms helt av det nät som används, t.ex. det lokala nätet Ethernet. Det ska tilläggas att det ursprungliga Internet byggdes upp av nätet **ARPANET**. Naturligtvis finns det protokoll för detta nät. Numera kan man använda många andra nät för att skicka meddelanden över Internet. Därför kan man säga att Internet-protokollen är oberoende av det fysiska nätet.

Sessions- och presentationsskikten används inte i Internet-modellen. Därför sägs Internet-modellen bestå av fyra skikt. Om man räknar **värd till nät** som två skikt, blir det fem skikt i Internet-modellen.

Skikt	OSI-modellen	Internet-modellen
Transport	Förbindelseorienterade eller förbindelselösa tjänster	Förbindelseorienterade eller förbindelselösa tjänster
Nätverk/Internet	Förbindelseorienterade eller förbindelselösa tjänster	Förbindelselösa tjänster

Tabell 1. Jämförelse av förbindelsetyper i OSI- och Internet-modellen.

Protokoll för Internet

De centrala protokollen i Internet-modellen är **Internet Protocol** (IP) i Internet-skiktet och **Transmissions Control Protocol** (TCP) i transportskiktet. TCP och IP kommer från **Defense Advanced Research Project Agency** (DARPA) i USA och utvecklades för nätet ARPANET. (Det är amerikansk stavning av *defense*.) Byrån DARPA tillhör det amerikanska **Department of Defense** (DoD). Därför sägs TCP och IP vara DoD-standarder. I figur 11 visas TCP och IP tillsammans med några andra Internet-protokoll.

Applikation	HTTP	FTP	SMTP m.fl.	DNS
Transport	TCP	UDP	NVP	
Internet	IP	ICMP	ARP	RARP
Värd till nätverk	ARPANET	Lokala nätverk som t.ex. Ethernet		Andra nätverk

Figur 11. Några protokoll för Internet.

Hyper-Text Transfer Protocol (HTTP)

HTTP används för att överföra webbsidor från webbservrar till webbklienter. Det är de senare som vi brukar kalla webbläsare (eng. *browsers*). Protokollen TPC/IP räcker inte till för att överföra webbsidor eftersom dessa protokoll inte ger tillräcklig PCI för sådana. Med hyper-text avses länkar i texter och bilder. De webbsidor som visas kan alltså innehålla länkar (dirigeringar) till andra webbsidor. Objekten på en webbsida är inte bara texter och bilder. Man använder också skript (ej kompilerade program), kompilerade program, ljud och videoer.

Texter skrivs och formateras med **Hyper-Text Markup Language** (HTML). De dokument (webbsidor) som visas på webben är vanligtvis skrivna med HTML-kod.

Det finns även webbsidor som är uppbyggda av serverbaserade skript som t.ex. **Active Server Pages** (ASP) och **PHP: Hypertext Preprocessor** (PHP). Sådana skript översätts av webbservrarna till ren HTML-kod som skickas över till webbklienterna. (Om koden innehåller klientbaserade skript, översänds sådana som de är till webbklienterna.)

Bilder kan vara sparade med olika komprimeringsmetoder. Detta brukar framgå av **file extensions**, t.ex. JPEG och GIF. Bilderna är komprimerade för att snabba upp överföringarna. JPEG och GIF är alltså exempel på komprimeringsmetoder. Liksom bilder kan infogas i texter, kan andra objekt också infogas så att man exempelvis klickar på en ikon (liten bild) eller textlänk för att få höra ett ljud spelas upp.

File Transfer Protocol (FTP)

FTP används för att skicka och ta emot filer med godtyckliga innehåll. Användaren har en FTP-klient som anropar FTP för att skicka till och ta emot filer från en FTP-server. I användarvänliga FTP-klienter brukar man i ena halvan av programfönstret se egna kataloger och filer. I den andra halvan brukar FTP-serverns kataloger och filer visas. Hanteringen kan påminna om hanteringen i Utforskaren (i Windows-system) så att man kan dra och släppa filer för att kopiera till/från FTP-servern.

En välkänd FTP-klient är WS_FTP. I Windows-systemen brukar det finnas en FTP-klient som hanteras från kommandoprompten (MS DOS-prompten). Även program för utveckling av webbsidor kan innehålla FTP-tjänster. Två exempel är Microsoft FrontPage och Macromedia DreamWeaver.

Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP) och Internet Message Access Protocol (IMAP).

Lika vanligt förekommande som att surfa på webben är det att skicka **e-post** (elektronisk post, eng. *electronic mail – email*). Det kan man göra från webbplatser som t.ex. Hotmail och Yahoo. Om man istället vill förvara sina e-brev i den egna datorn, används en e-postklient som t.ex. Outlook Express. Sådana e-postklienter anropar SMTP och POP (vanligtvis POP3, dvs. POP av version 3).

SMTP används för att skicka e-brev från e-postklienten till ett e-postkontor på Internet. POP används för det motsatta, dvs. att hämta inkommande e-brev från sin e-postbox på e-postkontoret. SMTP är ett betydligt större protokoll än POP. Det används också på e-postkontoret för administrera e-brev och för att skicka e-brev mellan e-postkontor.

IMAP används för att kunna bevara e-post på en server men ändå läsa sådan från olika datorer. Korgarna behöver alltså inte finnas i de lokala datorerna utan kan sparas på en server.

Domain Name System (DNS)

Det råder delade meningar om hur S i förkortningen DNS ska uttydas. I viss litteratur sägs att det ska betyda service eller server. Författaren föredrar S som i system eftersom det finns DNS-servrar. Varje gång som en användare surfar och anger webbplats med **Uniform Resource Locator (URL)**, t.ex. <http://www.oru.se>, måste webbläsare först få detta översatt till motsvarande IP-adress. Alla värdar som t.ex. webbservrar har minst en egen IP-adress. Det är denna adress som gör att meddelanden kan skickas till en bestämd värd på Internet.

Utan IP-adresser skulle ingenting fungera på Internet. Det är DNS-servern som översätter aktuellt domännamn (som finns i en URL eller e-postadress) till IP-adress. Ett domännamn kan också ha alias, dvs. inte endast ett namn. Även detta håller en DNS-server reda på.

Eftersom en enda DNS-server inte klara av att lagra alla IP-adresser och besvara alla uppslagsfrågor, finns det många DNS-servrar på Internet. Om den närmaste DNS-servern inte vet svaret på en uppslagsfråga, skickas denna vidare till nästa DNS-server i en kedja av sådana.

Transmission Control Protocol (TCP)

TCP är det mest använda av alla transportprotokoll på Internet. Det ger en förbindelse-orienterad tjänsteuppsättning, dvs. innan det egentliga meddelandet skickas så begärs uppkoppling. Detta innebär att avsändaren kontrollerar att mottagaren är igång och beredd innan det verkliga meddelandet skickas. Efter positivt svar från mottagaren påbörjas sändningen av det egentliga meddelandet. När meddelandet har skickats, görs nedkoppling.

Under tiden som en koppling finns, gör TCP både felkontroll och flödeskontroll. Vid fel, begärs omsändning. Detta innebär att TCP garanterar felfri överföring.

Meddelandet som sätts samman i transportskiktet kallas **TCP-segment** när detta protokoll används.

User Datagram Protocol (UDP)

UDP används istället för TCP när snabbheten sätts före säkerheten. UDP ger nämligen en förbindselös tjänsteuppsättning, dvs. det görs ingen uppkoppling innan det egentliga meddelandet skickas. I värsta fall är inte mottagaren igång när UDP skickar ett meddelande. UDP gör ingen flödeskontroll. Däremot finns det liksom i TCP en checksumma (i header) som sätts samman i transportskiktet. Om checksumman påvisar felaktig överföring, kastar mottagaren helt enkelt bort meddelandet utan att begära omsändning. UDP garanterar alltså inte felfri överföring.

Meddelandet som sätts samman i transportskiktet kallas **UDP-datagram** när detta protokoll används.

Genom att välja mellan tjänsterna i TCP och UDP kan kommunikationen med Internet-modellen göras antingen förbindelseorienterad eller förbindselös. IP i Internet-skiktet ger alltid förbindselösa tjänster, dvs. IP är neutralt i detta val. Därför bestämmer valet mellan TCP och UDP vilken typ av förbindelse som kommer att gälla.

Network Voice Protocol (NVP)

NVP är en transaktionsbaserad realtidstjänst för överföring av digitaliserat och komprimerat tal.

Internet Protocol (IP)

IP tillhandahåller transaktionstjänst mellan nät. Det är transportskiktet som begär sådana tjänster. Mestadels överförs TCP-segment och UDP-datagram i **IP-paket (datagram)**. IP är som tidigare har nämnts ett förbindselöst protokoll. Det finns checksumma i header som detta protokoll bestämmer vid sändning och kontrollerar vid mottagning. Lägg märke att checksummorna i TCP och UDP också omfattar IP-adressen som är en del av header i IP. Detta är helt otänkbart enligt OSI-modellen och brukar anges som ett starkt argument för att Internet-modellen inte har skarpa gränser (gränssnitt) mellan skikten. Enligt OSI-modellen får meddelanden i det fjärde skiktet inte ha checksummor som omfattar delar av header i det tredje skiktet.

Internet Control Message Protocol (ICMP)

ICMP används av värd datorer och routrar för att överföra nätinformation, dvs. sådan information som har med nätverkets funktion att göra. ICMP används t.ex. i de välkända programmen Ping och Traceroute. Ping används för att kontrollera om en värd (dator) är igång och för att bestämma svarstiden. Om man är intresserad av att undersöka vilken väg över routrarna på Internet som meddelanden går för att komma till en värd (mottagare), används Traceroute.

ICMP används alltid som ett komplement till IP, dvs. ICMP är i sig inte ett fullständigt nätverksprotokoll.

Address Resolution Protocol (ARP)

Vi har tidigare sagt att en värd på Internet endast kan adresseras av sin unika IP-adress. Detta gäller på det vida Internet, där inga andra adresser förekommer. Däremot används MAC-adresser i lokala nät. Varje nätverkskort (eller motsvarande adapter) har sin adress. MAC-adressering är alltså nödvändig för att hitta till bestämda mottagare inne i det lokala nätet. Om adressering görs med IP-adresser inne i ett lokalt nät, måste förstås varje IP-adress översättas till motsvarande MAC-adress. Översättning från IPv4-adress till MAC-adress görs av ARP. För IPv6 görs motsvarande översättning av **Neighbor Discovery Protocol (NDP)**.

Reverse Address Resolution Protocol (RARP)

Ibland uppstår behov att översätta från MAC-adress till IP-adress. Detta är främst ett behov för intern kommunikation i lokala nät. RARP översätter från MAC-adress till IPv4-adress.

IP-adresser

IP-adresserna finns som del av PCI som sätts samman i Internet-skiktet, dvs. header i IP-paketet. Det är **IPv4**, version 4, som för närvarande används. I denna version används 32 bitar (logiska nollar och ettor) för varje adress. Eftersom bitarna delas upp i två delar, **network identity** och **host identity**, kan adresserna delas in klasser, s.k. **classful addressing**. Vi ska lista de tre viktigaste klasserna och för att se vilken betydelse som respektive klass har. Det är den första oktetten (sett från vänster) som anger klassen. Dessa bitar är antingen hela network identity eller en del av detta. Ett exempel på IP-adress:

11000010 11110011 01101000 00000011

Oktetten längst till vänster är 11000010. Om detta binära tal översätts till motsvarande decimala, fås 194. Detta är exempel på en IP-adress av klass C eftersom 194 är större än 191 och mindre än 224. Se tabell 2. Man kan också studera begynnelsebitarna i 11000010 som är 110. Om begynnelsebitarna är 110, är det en IP-adress av klass C.

Vi översätter alla fyra oktetterna till var sitt decimaltal och får

194.243.104.3

Detta är betydligt lättare för en människa att memorera än IP-adressen på binär form. Notationen kallas **dotted decimal** (decimalform med punkter). Vanligtvis skrivs IP-adresser just på denna form.

När det är en IP-adress av klass C, är network identity lika med de tre oktettarna på vänstersida. Resten är host identity. Detta ger oss följande:

Network identity = 1100010 11110011 01101000
Host identity = 00000011

Om identiteterna skrivs med dotted decimal, får vi

Network identity = 194.243.104
Host identity = 3

Det är network identity som gör att ett meddelande kan hitta fram till mottagarens nätverk. Problemet är sedan att hitta rätt dator i detta nätverk. Då används host identity för att adressera mottagande dator (värd). Om det är ett lokalt nät, måste IP-adressen översättas till MAC-adress med hjälp av ARP. Network identity avslöjar endast datorns nät på Internet. För att komma fram till avsedd dator, behövs även delen som kallas host identity.

Om det finns en klass som kallas C, borde det också finnas klass A och B. Detta visas i tabell 2. Förutom klasserna A, B och C, finns klass D för **multicasting** och klass E som är reserverad för framtida bruk. Multicasting innebär att en grupp av mottagare får samma

meddelande. Om alla datorer får samma meddelande i ett nät, kallas detta **broadcasting**. Vanligtvis adresseras en mottagare. Detta kallas **unicasting**. Klass A ger relativt få network identities. Därför blir det många host identities i varje nät. Detta passar bra för mycket stora företag/organisationer. Klass C är raka motsatsen, dvs. många network identities med få host identities i var och ett av näten. Detta passar bra för små företag/organisationer. Klass B har lika många network identities som host identities.

Klass	Typ	1:a oktetten	Karakteristiska bitar i 1:a oktetten	Network identity	Host identity
A	Få nät med många värdar	0-126 (0-127*)	0	xxx	xxx.xxx.xxx
B		128-191	10	xxx.xxx	xxx.xxx
C	Många nät med få värdar	192-223	110	xxx.xxx.xxx	xxx

*) 127 är reserverad för loopback test och inre processkommunikation i datorerna.

Tabell 2. Klasserna A–C i IPv4.

Eftersom lediga IP-adresserna har börjat att ta slut, har förslag på nya versioner tagits fram. Den version som kommer att införas kallas **IPv6**. Denna version använder IP-adresser med 128 bitar. Detta ger drygt $3,4 \cdot 10^{38}$ adresser, dvs. ett mycket stort antal adresser som kommer att räcka länge. Det sägs att det mycket stora antalet IP-adresser räcker till alla värdar och till mycket mer. Man har på både skämt och allvar föreslagit att ge varje lyktstolpe sin egen IP-adress för individuell inställning över nätverk. Andra föreslår att brödrostarna ska få var sin IP-adress. Detta ger en aning om det enorma antalet IP-adresser som 128 bitar ger.

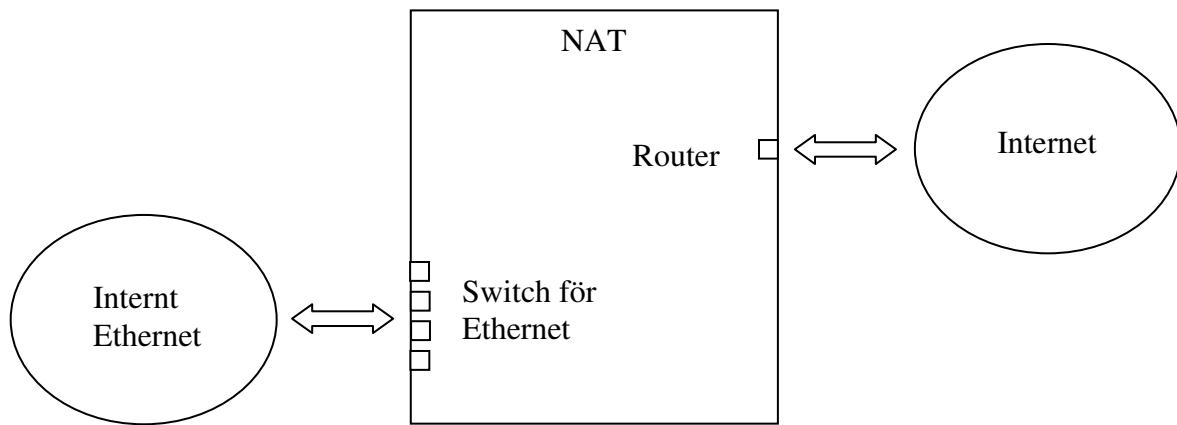
För att översätta adresser från IPv4 till IPv6 används **Network Address Translation - Protocol Translation** (NAT-PT). För att göra adresssystemen kompatibla används bl.a. tunnelteknik (inkapsling) som innebär att IPv6-paket skickas med hjälp av IPv4-paket, s.k. **IPv6-over-IPv4**.

Ett annat sätt att få IP-adresserna att räcka till är att endast använda ett fåtal publika IP-adresser mot omvärlden och inte i sitt nät istället använda privata. De interna IP-adresserna kan då väljas godtyckligt som t.ex. 192.168.0.0–192.168.0.255. För att översätta mellan en publik, extern IP-adress och privata, interna används en **Network Address Translator** (NAT). Denna kan vara implementerad i en sammankopplande enhet eller i en dator som har minst två nätverkskort.

Man kan köpa en typ av sammankopplande enhet som kallas **Internet gateway**. En sådan brukar vara försedd med en port till omvärlden som närmast är att jämföra med en routerport. Denna port har den publika, externa IP-adressen. Portarna på insidan är anpassade till något slags lokalt nät som t.ex. Ethernet. Dessa brukar vara portarna på en switch. Datorerna på insidan ska vara konfigurerade på så sätt att standard gateway är satt till den interna IP-adressen för Internet gateway. Då kommer anrop till IP-adresser utanför det interna nätet alltid att gå via Internet gateway. NAT som i detta fall finns i Internet gateway har till uppgift att med en tabell förmedla inkommende IP-paket till avsedd dator på insidan. Tabellen får en ny post då en dator skickar ett IP-paket till omvärlden. NAT ersätter datorns IP-adress och portnummer med sin publika, externa IP-adress och ett ledigt portnummer. Då datorn får ett

svar från utsidan kan NAT kontrollera IP-paketets portnummer och på så sätt översätta till datorns privata, interna IP-adress och portnummer.

Det är alltså inte svår teknik för att kunna begära svar från en yttre dator. Däremot kan en yttre dator inte utan vidare begära svar från en inre. Endast de datorer som i NAT är konfigurerade som servrar kan adresseras från utsidan. En server på insidan har inte en egen publik IP-adress men ändå ett bestämt portnummer. För att anropa servrar på insidan används en och samma IP-adress nämligen den publika, externa IP-adressen till Internet gateway i kombination med det bestämda portnumret.



Figur 12. Internet gateway.

En Internet gateway brukar vara försedd med **Dynamic Host Control Protocol (DHCP)**. På den externa porten fungerar den som DHCP-klient som kan begära en IP-adress från någon extern DHCP-server. Detta innebär att Internet gateway får låna en publik IP-adress istället för att alltid ha samma. På insidan brukar en Internet gateway fungera som DHCP-server för datorerna på insidan. Detta innebär att datorerna på insidan kan tilldelas privata IP-adresser av Internet gateway med undantag för de datorer som är servrar. De datorer som är servrar har som regel fasta, privata IP-adresser.

Delnätmasker

En delnätmask (subnetwork mask, eng. *subnetwork mask*) visar hur host identity ska delas upp i dels subnetwork identity och internal host identity. Delnätmasker anges där man skriver in adresser i IP-protokoll som t.ex. i datorer, skrivare och routrar. Fördelarna med uppdelning i IP-delnät är följande:

- Administrativt skäl

Om företaget/organisationen har ett site-wide LAN och/eller har LAN på olika orter som administreras av olika grupper av personer, är det praktiskt att känna till de intervall av IP-adresser som får användas av inom respektive delnät. Vanligtvis har företaget/organisationen tillgång till en network identity och alla tillhörande host identities. Det är de senare som delas upp i intervall med hjälp av delnätmasken.

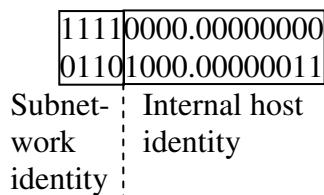
- Reducerad trafikbelastning

Om delnäten avgränsas med hjälp av routrar och/eller switchar som avläser IP-adresser, kan dessa ställas in med delnätmasken. Routing (vägval) görs med ledning av delnätmasken så att paketen inte skickas utanför delnätet om mottagaren finns inom detsamma. Detta reducerar nätverkets totala trafikbelastning.

Vi undersöker IP i en dator och avläser att IP-adressen är **190.243.104.3** och delnätmasken är **255.255.240.0**. Eftersom $127 < 190 < 192$ är det fråga om klass B. Se tabell 2. Klass B medför att IP-adressen delas upp på följande sätt:

$$\begin{aligned} \text{Network identity} &= 190.243 \\ \text{Host identity} &= 104.3 \end{aligned}$$

Då ser vi att masken är **255.255** i de grupper som motsvarar network identity. Det ska en delnätmask alltid vara. Vidare är delnätmasken **240.0** i de grupper som motsvarar host identity. Vi skriver **240.0** (delenätmasken) ovanför **104.3** (host identity) med binära tal.



De bitar i host identity som står under logiska ettor i delnätmasken, bildar subnetwork identity. De övriga bitarna, dvs. som står under logiska nollor i delnätmasken, bildar internal host identity. Vi får

$$\begin{aligned} \text{Subnetwork identity} &= 0110_2 = 6_{10} \text{ alternativt } 01100000_2 = 96_{10} \\ \text{Internal host identity} &= 1000.00000011_2 = 8.3_{10} \text{ alternativt } 100000000011_2 = 2051_{10} \end{aligned}$$

Detta innebär att datorn befinner sig i delnätet som har adressen **6** (alternativt **96**) och att datorn har den interna värdadressen **8.3** (alternativt **2051**). Denna uppdelning gäller endast inom företagets/organisationens lokala nät som kan vara sammansatt av flera delar i ett site-wide LAN och/eller finnas på flera orter. När datorn adresseras från utsidan (ute på Internet), används först network identity för att hitta fram till företaget/organisationen. Från utsidan har datorn host identity **104.3** och inget annat. Det är endast på insidan som delnätmasken avslöjar att datorn har subnetwork identity **6** (alternativt **96**) och internal host identity **8.3** (alternativt **2051**).

Om delnätmasken hade varit **255.255.0.0** i datorn, skulle inte något subnetwork identity finnas. En sådan delnätmask för klass B delar inte upp nätet i delnät. Motsvarande ej uppdelande delnätmask för klass A är **255.0.0.0** och för klass C **255.255.255.0**.

En annan delnätmask för klass B som inte är lika sannolik som de tidigare är **255.255.255.240**. En sådan delnätmask ger betydligt fler delnät än det finns värder i respektive delnät. Några av de möjliga och sannolika delnätmaskerna för klass B är följande:

```
11111111.11111111.00000000.00000000
11111111.11111111.10000000.00000000
11111111.11111111.11000000.00000000
11111111.11111111.11100000.00000000
11111111.11111111.11110000.00000000
11111111.11111111.11111000.00000000
11111111.11111111.11111100.00000000
11111111.11111111.11111110.00000000
11111111.11111111.11111111.00000000
```

Delnätmaskerna är skrivna på binär form för att påvisa den möjliga strukturen hos delnätmaskerna. Den första är delnätmasken som inte alls delar upp. Den sista delnätmasken använder alla åtta bitarna i den tredje oktetten. Den sista delnätmasken medför att internal host identity består av endast den fjärde oktetten, dvs. inga bitar från den tredje oktetten ingår. Därför kan man på skoj säga att det är en klassändring från B till C. Lägg märke till att delnätmasken inte får innehålla några luckor mellan de ingående logiska ettorna. Därför är delnätmaskens möjliga decimala värden i den tredje oktetten 0, 128, 192, 224, 240, 248, 252, 254 och 255.

Klasslös routing

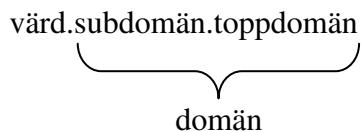
Classless InterDomain Routing (CIDR, uttalas ”cider”) är en teknik som **blockindelar** nätverksadresser (adressaggregation). Blockindelningen infördes för att spara på adressutrymme vid utdelning av IPv4-adresser. RFC 1918 (feb, 1996) förklarar syftet med blockindelning och att detta endast är ett annat skrivsätt jämfört med klassindelning. Skrivsättet ändrar inte på klasser, delnätmasker och delnät. En delnätmask används tillsammans med IP-adressen för att skapa ett delnät med hjälp av bitar från host identity. Klasserna finns fortfarande och den maskningsteknik som hör till dessa likaså. De används främst i de privata näten. Ett bevis för att klassindelningen fortfarande används är att det är svårt att hitta ett operativsystem för IPv4 som inte använder tillhörande maskningsteknik.

CIDR ställer stora krav på routrarna. Anledningen är att **Internet Service Providers** (ISPs) tillhandahåller delar av hela network identities åt företag/organisationer. För att klara av detta infördes ”nätmasker” i routrarna som samlar företags/organisationers adresser till en och samma post i en routingtabell, nämligen posten för adressen till aktuell ISP. Sådana ”nätmasker” kan skilja mellan adresser till företag/organisationer som hör till samma ISP. Därför kan man skriva IP-adresser som börjar med samma bitar i en post (för att spara utrymme i routerns minne). Ett exempel är 200.23.16.0/20 som innebär att adresserna i intervallet 200.23.16.0–200.23.31.255 ska utgöra en post i routern. Skrivsättet **IP-adress/x** anger vilka bitar i network identity som ska avläsas. Man ska läsa **x** antal bitar från vänster. Dessa bitar kallas **prefix** eller **nätprefix**. Detta samlar adresserna i intervall, för t.ex. en ISP, till en bestämd utport på routern. Tekniken kallas aggregering av adresser. Här framkommer den klasslösa principen.

Numera används samma skrивsätt även för att dela upp privata nät, t.ex. 223.1.1.0/24 som ger **delnätet** 223.1.1.0–223.1.1.255. I ett sådant fall kallas /24 för **delnätmask** och 223.1.1 för **delnätsadress**. Detta påvisar likheten mellan klasslösa och klassindelade adresser.

Uniform Resource Locator (URL)

När man surfar eller skickar e-post, kan det vara svårt komma ihåg IP-adresser. Det är betydligt lättare att skriva domännamn, t.ex. **www.oru.se**, och e-postnamn, t.ex. **jack.pencz@tech.oru.se**. Vi ska studera hur en URL byggs upp och vi börjar med domännamnet. Vi nämnde **www.oru.se** som ett exempel på domännamn. Sådana har följande format:



Värd: datorn kallas ofta **www**.
(Namnet är godtyckligt, den skulle lika gärna kunna heta puttrik).

Subdomän: företag, institution, ort etc.

Toppdomän: com (företag)

landskod
edu (utbildning)
gov (myndighet)
org (organisation)
net (nätverk)
mil (militärt)
int (internationellt)
m.fl.

Subdomänen kan ibland bestå av flera delar. I följande exempel består subdomänen av **tech.oru**:

<http://www.tech.oru.se>

För att erhålla en komplett URL, ska tjänsttypen anges, t.ex. **http://www.oru.se**, där http anger att tjänsttypen är Hyper-text transfer protocol. Om inte värdatorns normala startfil används, ska önskad fil också anges. Vanligt förekommande startfiler är **index.htm**, **index.html**, **index.asp**, **default.htm**, **default.html** och **default.asp**. Då får vi följande format för URL:

tjänst://värd.subdomän.toppdomän/filnamn

Exempel:

<http://www.april.se/tult.html>

<ftp://ftp.geocities.com>

Man kan också ange katalogsökvägen till önskad fil. I följande exempel är **aktuellt** en rot-katalog och **mat.html** är önskad fil (dokument):

<http://www.oru.se/aktuellt/mat.html>

Sökvägarna kan vara långa. I följande exempel används rotkatalogen **org** och under-katalogerna **avd**, **rektor**, **centraladok** och **csr**:

<http://www.oru.se/org/avd/rektor/centraladok/csr/index.html>

Det är inga problem med att ha flera servrar av olika typ på en och samma dator. Applikationsprogram inklusive servrar är nämligen kopplade till applikationsprotokoll. Det finns service access points i mottagande dator mellan transportskiktet och applikationsskiktet (i Internet-modellen). En sådan TSAP består av ett 16-bitars portnummer. Varje servertyp har en rekommenderad port. Vi har t.ex. 80 för webbservrar (HTTP) och 20/21 (svar/anrop) för FTP-servrar. Val av TSAP hos mottagaren ger önskat applikationsprotokoll i applikations-skiktet. Varje applikationsprogram som används på Internet är kopplat till ett applikations-protokoll. Därför ger val av TSAP önskat applikationsprogram (via applikationsprotokollet). Därför går det bra att ha t.ex. en webbserver på samma dator som en FTP-server.

Det finns knep för att kunna ha flera servrar av samma typ på en och samma dator, t.ex. att använda olika IP-adresser till servrarna. Om det redan finns en typ av server på en dator, kan andra av samma typ tilldelas ej använda (och ej rekommenderade) portar. På så sätt kan flera servrar av samma typ köras på en och samma dator och använda samma IP-adress. (Man kan naturligtvis också kombinera med olika IP-adresser och olika portar. Nackdelen med detta är bristen på IP-adresser.) Om man ska surfa till en webbserver t.ex. på port 8900 som finns i samma dator som en annan webbserver på port 80, måste porten 8900 anges i anropet. Vi tänker oss att den önskade webbserver finns på **www.januari.se**. Då ska följande URL anges för att komma till önskad webbserver:

<http://www.januari.se:8900>

Om man surfar till webbservern som använder port 80, behöver man inte ange porten, dvs.

<http://www.januari.se>

Med **fully qualified address** avses IP-adress och portnummer. Antag att **www.januari.se** har IP-adressen **130.255.100.15**. Då är fully qualified address till webbservern på port 8900 följande:

130.255.100.15:8900

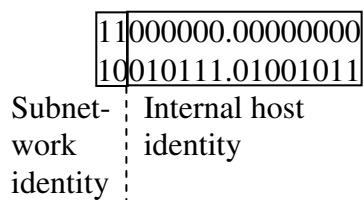
Exempel och övningar

IP-adressen är **150.211.151.75** och delnätmasken är **255.255.192.0** i en skrivare. Vi ska bestäm klass, network identity, subnetwork identity och internal host identity.

Eftersom $127 < 150 < 192$ är det fråga om klass B. Detta medför att IP-adressen delas upp på följande sätt:

$$\begin{aligned} \text{Network identity} &= \underline{150.211} \\ \text{Host identity} &= 151.75 \end{aligned}$$

Delnätmasken är **192.0** i de grupper som motsvarar host identity. Vi skriver **192.0** (delnätmasken) ovanför **151.75** (host identity) med binära tal.



De bitar i host identity som står under logiska ettor i delnätmasken, bildar subnetwork identity. De övriga bitarna, dvs. som står under logiska nollor i delnätmasken, bildar internal host identity. Vi får

$$\begin{aligned} \text{Subnetwork identity} &= 10_2 = \underline{2}_{10} \text{ alternativt } 10000000_2 = \underline{128}_{10} \\ \text{Internal host identity} &= 010111.01001011_2 = \underline{23.75}_{10} \text{ alternativt } 01011101001011_2 = \underline{5963}_{10} \end{aligned}$$

1. Du undersöker IP-protokollet i din dator och finner IP-adressen 200.200.200.74. Vidare ser du att delnätmasken är inställd på 255.255.255.240. Ange klass, network identity, subnetwork identity och internal host identity.
2. Du undersöker IP-protokollet i din dator och finner IP-adressen 128.240.100.12. Vidare ser du att delnätmasken är inställd på 255.255.240.0. Ange klass, network identity, subnetwork identity och internal host identity.
3. Du undersöker IP-protokollet i din dator och finner IP-adressen 114.240.15.185. Vidare ser du att delnätmasken är inställd på 255.192.0.0. Ange klass, network identity, subnetwork identity och internal host identity.
4. Välj delnätmask så att blir maximalt åtta delnät, dvs. delnät nr. 0–7 (alternativt delnät nr. 0, 32, 64, 96, 128, 160, 192 och 224), för adresserna 193.204.194.0–193.204.194.255.

Lösningar

1. Klass C, network identity = 200.200.200, subnetwork identity = 4 (alternativt 64) och internal host identity = 10.
2. Klass B, network identity = 128.240, subnetwork identity = 6 (alternativt 96) och internal host identity = 4.12 (alternativt 1036).
3. Klass A, network identity = 114, subnetwork identity = 3 (alternativt 192) och internal host identity = 48.15.185 (alternativt 3149753).
4. 255.255.255.224 eftersom det är klass C.

Övningar på nodfödröjningar

Uppgifter

1. Beräkna nodfödröjningen i en router som har processfödröjningen 0,21 ms.

Väntetiden (köfödröjningen) är 0,34 ms.

Bithastigheten är 100 Mbps och bitarna inordnas i paket som rymmer 2 KiB. (Om vi med paket avser bitar i paket eller bitar i ramar, är oväsentligt. Huvudsaken är att bithastigheten anges för rätt meddelandetyp, dvs. hastighet för paketbitar eller hastighet för rambitar.)

Observera att B står för byte, 1 B = 8 bitar och 1 KiB = 1024 B.

Transmissionsmediet till nästa router är fiberoptisk kabel med längden 500 m.

Utbredningshastigheten är ungefär 2/3 av ljushastigheten i vakuum.

2. Vad blir den totala nodfödröjningen (enligt uppgift 1) om paketen skickas över tre lika länkar, dvs. över tre routrar med tillhörande transmissionskanal, som har samma prestanda och samma köfödröjning?
3. Antag att avsändande värd behöver ungefär samma processtid som en router och att transmissionskanalen till den första routern är 500 m fiberoptisk kabel. Vad blir då den totala födröjningen från värd till värd (end-to-end delay) i uppgift 2?
4. Antag att mottagande värd behöver ungefär samma processtid som avsändande värd. Vad blir då round-trip delay?
5. Beräkna nodfödröjningen i en satellit (router) som har processfödröjningen 0,19 ms.
Väntetiden (köfödröjningen) är 0,82 ms.
Bithastigheten är 100 Mbps och bitarna inordnas i paket som rymmer 2 KiB.
Avståndet till mottagande markstation (värd) är 39.000 km.
6. Vad är $I = La/R$? Vilka storheter är L , a och R ? Vilka enheter har L , a och R ?

7. Inne i en router levereras paket i en jämn ström till en av utportarnas kö.

Det kommer i medeltal in 10 paket per sekund till kön.

Paketen har storleken 10.000 bitar.

Dessa sänds ut från porten med bithastigheten 100 Mbps.

Beräkna trafikintensiteten på den nämnda utporten och ange om det är god design eller ej.

Lösningar

1. Vi får

$$d_{\text{proc}} = 0,21 \text{ ms}$$

$$d_{\text{queue}} = 0,34 \text{ ms}$$

$$L = 2 \text{ kB} = 2 \cdot 8 \text{ kb} = 16 \cdot 1024 \text{ b} = 16.384 \text{ b}$$

$$R = 100 \text{ Mbps}$$

$$d_{\text{trans}} = L/R = 16384/100 \cdot 10^6 \approx 164 \cdot 10^{-6} = 164 \mu\text{s}$$

$$d = 500 \text{ m}$$

$$s \approx 3 \cdot 10^8 \cdot 2/3 = 2 \cdot 10^8 \text{ m/s}$$

$$d_{\text{prop}} = d/s = 500/2 \cdot 10^8 = 2,5 \cdot 10^{-6} \text{ s} = 2,5 \mu\text{s}.$$

Eftersom $d_{\text{trans}} \gg d_{\text{prop}}$, kan den senare försummas. Detta ger

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} \approx d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} = 0,21 + 0,34 + 0,164 \text{ ms} = 0,714 \text{ ms} \approx \underline{0,71 \text{ ms}}.$$

2. Vi får

$$d_{3 \text{ links}} = 3d_{\text{nodal}} = 3 \cdot 0,714 \text{ ms} = 2,142 \text{ ms} \approx \underline{2,1 \text{ ms}}.$$

3. Vi får

$$d_{\text{end-end}} = 4d_{\text{nodal}} = 4 \cdot 0,714 \text{ ms} = 2,865 \text{ ms} \approx \underline{2,9 \text{ ms}}.$$

4. Vi får

$$d_{\text{rond-trip}} = 2d_{\text{end-end}} = 2 \cdot 2,865 \text{ ms} = 5,73 \text{ ms} \approx \underline{5,7 \text{ ms}}.$$

5. Vi får

$$d_{\text{proc}} = 0,19 \text{ ms}$$

$$d_{\text{queue}} = 0,82 \text{ ms}$$

$$L = 2 \text{ kB} = 2 \cdot 8 \text{ kb} = 16 \cdot 1024 \text{ b} = 16.384 \text{ b}$$

$$R = 100 \text{ Mbps}$$

$$d_{\text{trans}} = L/R = 16384/100 \cdot 10^6 \approx 164 \cdot 10^{-6} = 164 \mu\text{s}$$

$$d = 39000 \text{ km} = 39 \cdot 10^6 \text{ m/s}$$

$$s \approx 3 \cdot 10^8 \text{ m/s}$$

$$d_{\text{prop}} = d/s = 39 \cdot 10^6 / 3 \cdot 10^8 = 0,13 \text{ s} = 130 \text{ ms} = 130.000 \mu\text{s.}$$

Eftersom $d_{\text{trans}} \ll d_{\text{prop}}$, kan den förra försummas. Detta ger

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}} \approx d_{\text{proc}} + d_{\text{queue}} + d_{\text{prop}} = 0,19 + 0,82 + 130 \text{ ms} = 130,01 \text{ ms} \approx \underline{131 \text{ ms.}}$$

7. Vi får

$$L = 10.000 \text{ b}$$

$$a = 10 \text{ paket/s (eller paketfrekvensen } a = 10 \text{ 1/s} = 10 \text{ s}^{-1})$$

$$R = 100 \text{ Mbps} = 100 \cdot 10^6 \text{ bps}$$

$$I = aL/R = 10 \cdot 10000 / 100 \cdot 10^6 = \underline{0,001}.$$

Det är god design eftersom $I < 1$.

(Detta kan uttryckas som att $aL/R < 1 \Leftrightarrow a < R/L$ eftersom vi har $10 < 10.000$. Paketen skickas ut snabbare än de kommer in i kön.)

Övningar på Internet-checksummar

Uppgifter

1. Tänk dig att följande data ska täckas av checksumman i UDP header:

```
0101010101010101  
1111000011110000  
1100110011001100  
1110111011101110
```

Egentligen är det många fler data som täcks av checksumman i ett verkligt UDP-datagram men använd trots detta de fyra data och beräkna denna.

2. Tänk dig att följande data ska täckas av checksumman i UDP header:

```
01010101010101  
1111000011110000  
1100110011001100  
1100111011101110
```

Egentligen är det många fler data som täcks av checksumman i ett verkligt UDP-datagram men använd trots detta de fyra data och beräkna denna.

3. Till en värd anländer ett UDP-datagram. Tänk dig att det endast består av följande fyra data och en checksumma (längst ned):

```
01010101010101  
1111000011110000  
1100110011001100  
1111111011101110  
1111011000000000 (checksumman)
```

Visa om överföringen har givit bitfel eller ej. (I verkliga UDP-datagram ingår betydligt fler data.)

4. Till en värd anländer ett UDP-datagram. Tänk dig att det endast består av följande fyra data och en checksumma (längst ned):

```
01010101010101  
1111000011110000  
1100110011001100  
1111111011101110  
1110110111111101 (checksumman)
```

Visa om överföringen har givit bitfel eller ej. (I verkliga UDP-datagram ingår betydligt fler data.)

5. Varför har man checksummar i TCP-segment och UDP-datagram? Räcker det inte med att IP-paketen (IPv4) har egna checksummar och att många nät har checksummar i ramarna som sätts ihop i länkskiktet, exempelvis Ethernet-ramar?

Lösningar

1. Checksumma = 11111011111101.
2. Checksumma = 00011011111110.
3. Summan av de fyra data och checksumman ger 0000100000000011 som inte är 1111111111111111, dvs. fel under överföringen.
4. Summan av de fyra data och checksumman ger 1111111111111111, dvs. korrekt överföring.
5. TCP-segment och UDP-datagram kapslas in i IP-paket, men paketen kan delas upp av routrarna p.g.a. olika nätegenskaper och trafikförhållanden. Då beräknas nya checksummar, en för varje IPv4-paket. (IPv6 har ingen checksumma.) Därför kan man inte förlita sig på en IP-checksumma under kommunikationen sändare till mottagare. Den checksumma som mottagare får kan ju ha blivit beräknad av en router längs vägen.

På liknande sätt är det med ramarnas checksummar. Vid byte av bärarnät längs vägen, förlorar en rams checksumma betydelse för transporten från till till. Det är endast inom ett och samma bärarnät som en ramchecksumma är relevant.

Därför kan man endast förlita sig på checksummorna i TCP-segment och UDP-datagram i kommunikationen *end-to-end*. Dessa checksummar räknas aldrig om längs transportvägen.

Transportskiktet som sätter samman TCP-segment och UDP-datagram, hanterar ju **kommunikationen mellan processer**. Jämför med den sändande värdens Internet-skikt (nätverksskikt) som endast hanterar kommunikationen fram till den första routern längs vägen. (Undantaget är förstås mottagarens IP-adress som skrivs in av sändaren.)

Felövervakning

Introduktion

Under överföring av meddelanden (ramar, paket, segment etc.) kan olika fel inträffa. Det mest drastiska är att hela meddelanden eller delar av meddelanden försvisser. Meddelanden kan också komma fram i oordning. Dessa typer av fel kan mottagaren hantera genom att undersöka de sekvensnummer som sändaren tilldelar. Låt n vara antalet bitar i fältet för sekvensnummer. Då finns det 2^n antal sekvensnummer. Ett exempel är $n = 3$ som innebär att det finns åtta sekvensnummer, dvs. $000\text{-}111_2$ eller $0\text{-}7_{10}$. För att kunna numrera betydligt fler meddelanden än åtta återanvänds sekvensnumren cyklistiskt. I vårt exempel får vi följande sekvensnummer: $0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, \dots$. Detta är en talserie av de naturliga talen modulo 8. I det generella fallet ger n bitar sekvensnummer som är talserier av de naturliga talen modulo 2^n .

En annan typ av fel är att någon eller några bitar har blivit så störda under överföringen att mottagarens sampling ger inverterade värde för dessa. Sådana fel kan mottagaren inte hantera genom att kontrollera sekvensnummer. Istället krävs det checksummor (kontrollsummor) för att upptäcka bitfel. I det enklaste fallet används en enkel paritetsbit för kontroll av enskilda data. Vi ska återkomma till felupptäckande koder och till sådana koder som även kan rätta till felaktiga bitar.

En enkel form av felövervakning är **echo checking** (test genom eko). Denna typ av felkontroll är tämligen ineffektiv och passar därfor bäst för terminaler och vid teckenorienterad¹, asynkron² kommunikation. När användaren trycker ned en tangent eller en kombination av flera tangenter på tangentbordet, skickas en kod till mottagaren som kan vara en dator. Mottagaren sparar koden (för tecknet/tecken-kombinationen) och skickar även en kopia av denna tillbaka till terminalen. Då visar terminalen tecknet (eller resultatet av tecken-kombinationen) på bildskärmen. Det är alltså användaren som kontrollerar huruvida det är samma tecken (eller tecken-kombination) som kommer tillbaka eller ej. Naturligtvis blir denna typ av felövervakning relativt långsam eftersom den bygger på att koden skickas tillbaka till terminalen. Även om echo checking automatiseras så att terminalen kontrollerar returkoden så passar metoden inte för stora och snabba överföringar.

I de flesta kommunikationssystem används principen att det är mottagaren som gör felövervakning. Detta kallas **Automatic Repeat Request** (ARQ). Det finns två huvudversioner av ARQ som kallas **idle repeat request** (send-and-wait, stop-and-wait) och **continuous repeat request**.

¹ Med teckenorienterad kommunikation avses överföring av data som tolkas efter bestämd teckentabell. Motsatsen är att data inte tolkas efter bestämd teckentabell under överföringen. Detta kallas bitorienterad kommunikation. Detta hindrar inte att tecken kan skickas likaväl som all annan binär information.

² Asynkron kommunikation innebär att data som överförs förses med start- och stoppbitar. Antal bitar och logiska nivåer anges av linjeproceduren (datalänkprotokollet). Kombinationen teckenorienterad och asynkron är vanligt förekommande. Motsatsen till asynkron kommunikation kallas synkron. För synkron kommunikation används inte speciella start- och stoppbitar. Istället används start- och stoppflaggor (sekvenser av bitar) för bitorienterade protokoll och speciella tecken enligt bestämd teckentabell för teckenorienterade.

Omsändningsmetoderna (ARQ) används i system där mottagarna kan upptäcka fel (**feedback error control**) och även i system där mottagarna kan korrigera bitfel (**forward error control**). ARQ behövs för att kunna repetera meddelanden som har kommit bort och som har blivit störda under överföringarna.

Omsändningsmetoder

Idle repeat request

Det finns två versioner av idle repeat request som skiljer sig från varandra beroende på om det förutom **positiva kvittenser (Acknowledgement, ACK)** används **negativa kvittenser (Negative Acknowledgement, NAK)**. Idle repeat request passar bäst för terminaler och instrument som har små sändnings- och mottagningsbuffertar. Det gäller datakommunikation över moderata avstånd och med måttliga bithastigheter. (En alternativ stavning för kvittens på engelska är *acknowledgment*.)

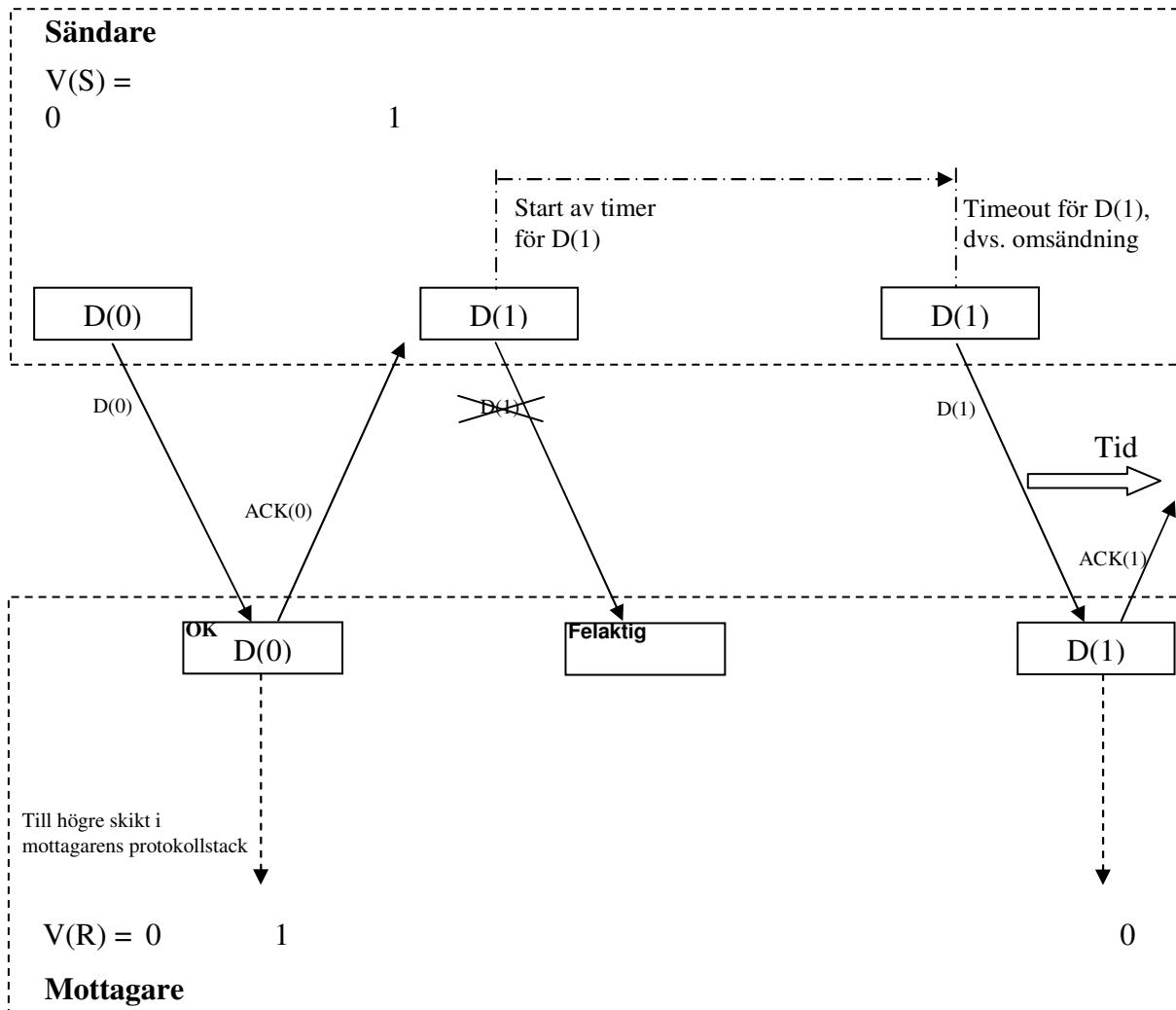
Idle repeat request bygger på principen att sändaren inte får skicka nästa meddelande (ram, paket, segment etc.) innan positiv kvittens för det förra meddelandet har kommit till sändaren. Därför behövs det endast två sekvensnummer för att mottagaren ska kunna felövervaka. Om fältet för sekvensnummer görs så litet som möjligt, blir det endast en bit. Då kan man ange sekvensnumren 0 och 1. Detta är tillräckligt för att mottagaren ska se skillnad på meddelanden som tas emot.

Implicit retransmission

Sändaren skickar nästa meddelande endast om den får ACK från mottagaren för det förra. Detta kommer att fungera tills det blir fel på en ACK eller att ett meddelande blir felaktigt. För att skydda kommunikation mot felaktig eller utebliven ACK förses sändarna med var sin **timer**. Sändaren inväntar alltså antingen ACK eller **timeout** för att avgöra vilket meddelande som ska skickas härnäst. Timeout ger repetition av det senaste meddelandet. Detta är också tekniken för mottagaren att meddela fel på meddelande. Mottagaren låter helt enkelt bli att skicka någon ACK. Tids nog blir det timeout som gör att sändaren repeterar det senaste meddelandet. Om mottagaren medvetet har undvikit att skicka ACK, blir det inget problem med det meddelande som skickas på nytt. Annars, om en ACK har blivit felaktig eller uteblivit, tror mottagaren att det är nästa meddelande som kommer. Därför används sekvensnumret för att avslöja en ev. **dubblett**.

Vi kallar sändarens sekvensräknare för V(S). Innehållet i denna modulo 2-räknare är antingen 0 eller 1 och värdet ska tolkas som sekvensnumret för nästa meddelande som kommer att sändas. Man kan använda fler sekvensnummer med två stycken är minimum för att kunna hålla kontroll på meddelandena. Mottagaren har en liknande räknare (modulo 2) för att hålla kontroll på nästa förväntade meddelande. Denna räknare kallas V(R). I figur 1 visas principen för **implicit retransmission**.

Eftersom sändaren inväntar ACK eller timeout innan nästa meddelande skickas, kan sändnings- och mottagningsbufferten göras minimala, dvs. med plats för ett meddelande hos både sändare och mottagare.

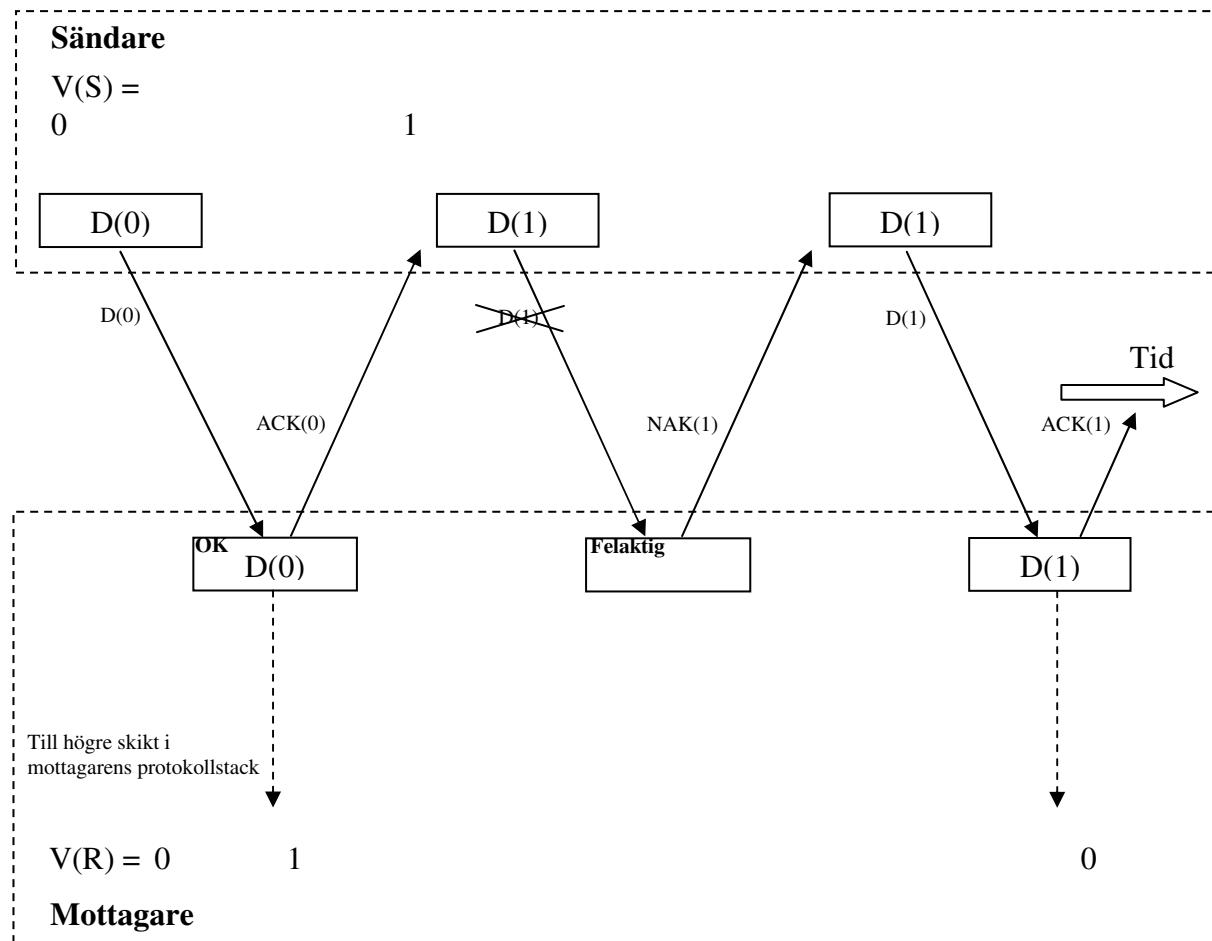


Figur 1. Idle repeat request with implicit retransmission.

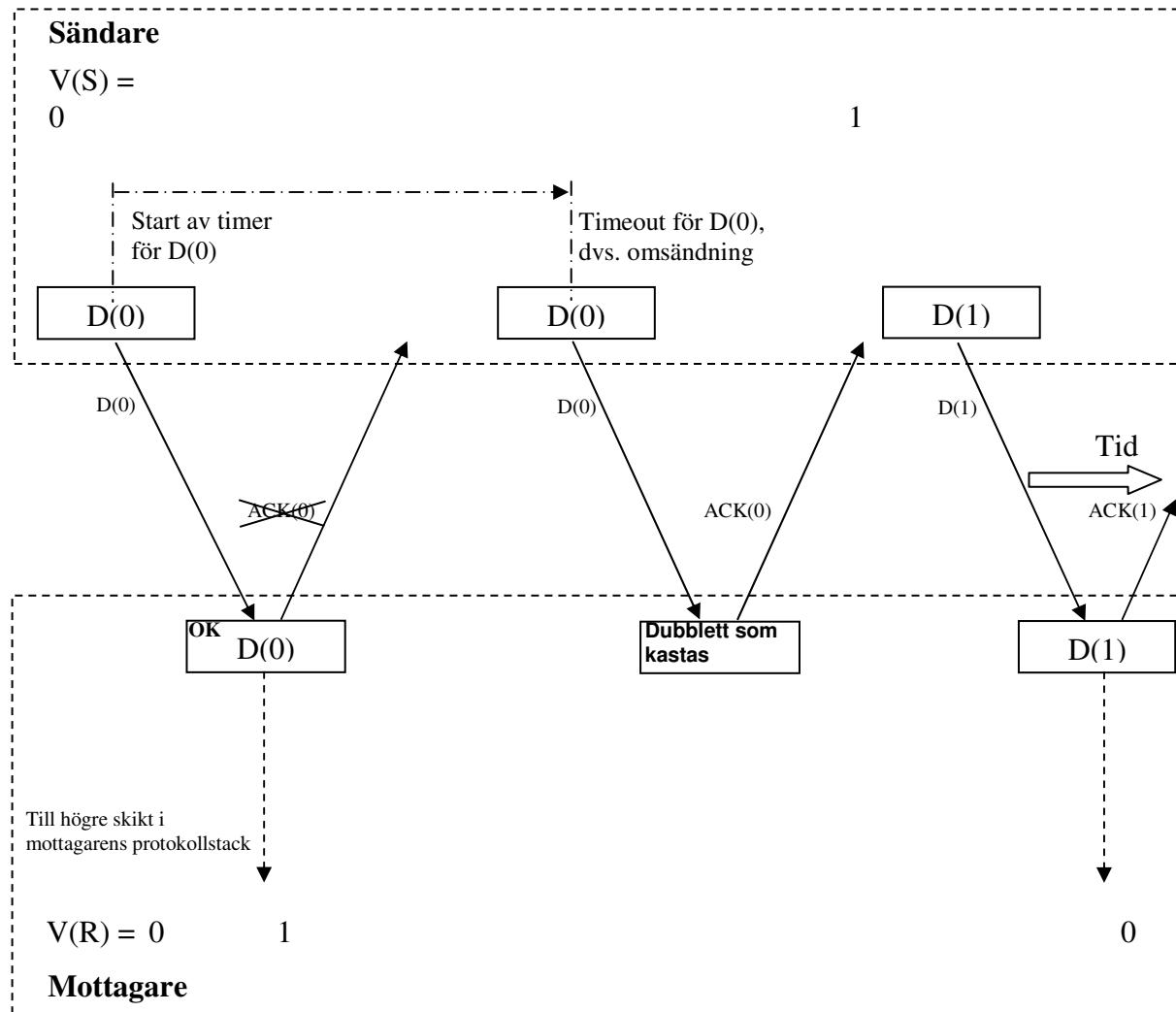
Explicit request

Explicit request fungerar på liknande sätt som implicit retransmission. Skillnaden mellan dessa är att den förra använder NAK. Detta gör begäran om omsändning snabbare eftersom sändaren får NAK innan timeout. Explicit request visas i figur 2.

Givetvis förses explicit request med timers på liknande sätt som för implicit retransmission. Om varken ACK eller NAK kommer i tid, skickas det senaste meddelandet på nytt. Anledningarna till utebliven ACK/NAK kan vara att den har blivit förstörd under överföringen eller att dataramen aldrig kom fram. Oavsett felorsak skickar sändaren alltså ett nytt meddelande. Detta innebär att mottagaren måste kunna ta bort ev. dubblett. Detta visas i figur 3.



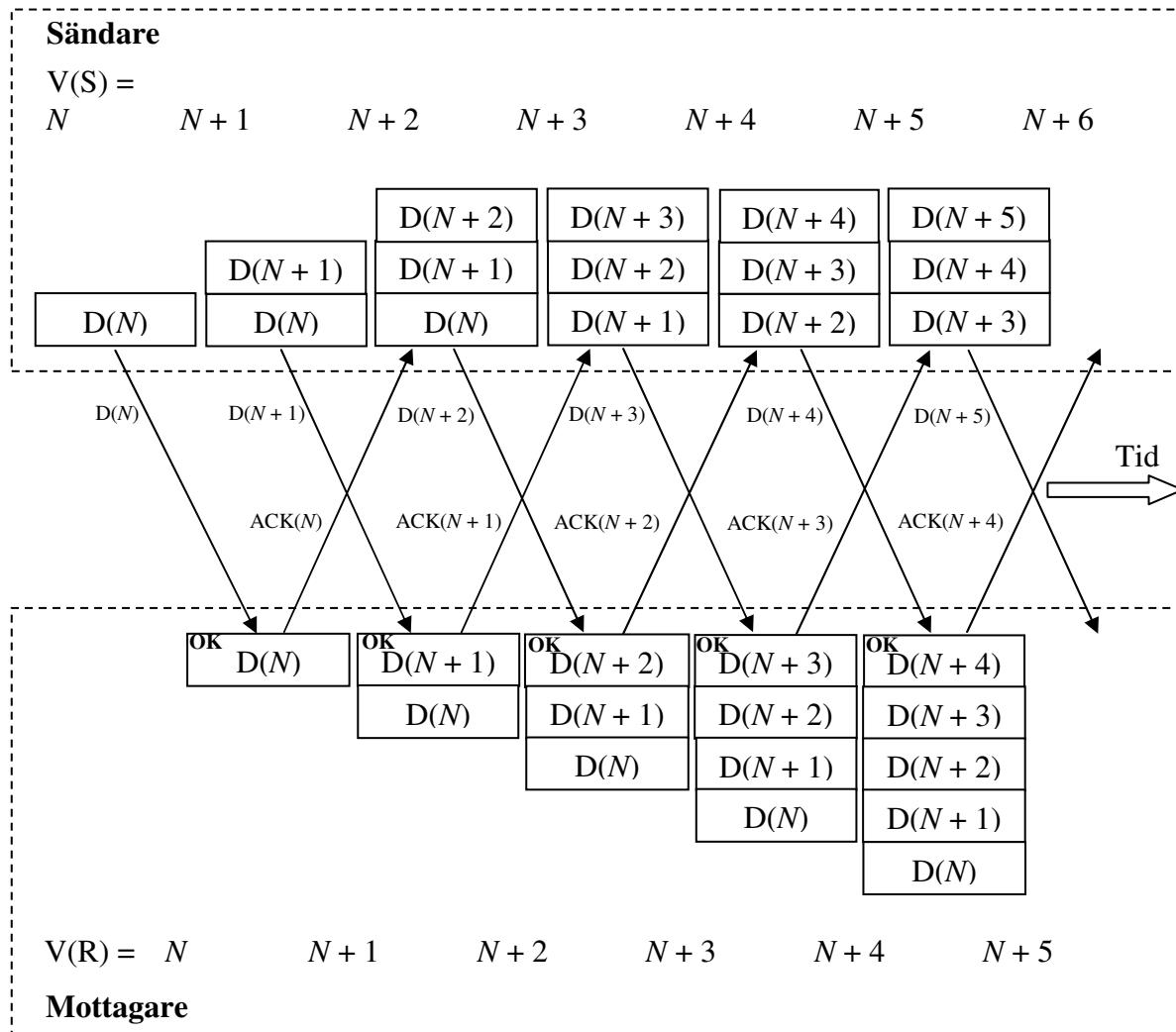
Figur 2. Idle repeat request with explicit request.



Figur 3. Idle repeat request i fallet med en dubblett.

Continuous repeat request

De kontinuerliga omsändningsmetoderna använder större sändnings- och mottagningsbuffertar än för idle repeat request. Av namnet framgår det att denna ARQ-huvudgrupp kan skicka meddelanden (ramar, paket, segment etc.) som i strömmar utan att behöva invänta varje ACK, NAK eller timeout. Det är antalet sekvensnummer som sätter den övre gränsen för antalet möjliga meddelanden att skicka utan att först få någon form av kvittering. Sändnings- och mottagningsbuffertarnas nödvändiga storlekar bestäms i sin tur av antalet sekvensnummer. Eftersom det inte är nödvändigt med en ACK för varje meddelande, blir de kontinuerliga omsändningsmetoderna effektivare (snabbare). I denna huvudgrupp finns det tre versioner: **selective repeat with implicit retransmission**, **selective repeat with explicit request** och **go-back-N**. Principen för continuous repeat request visas i figur 4.



Figur 4. Continous repeat request.

Selective repeat with implicit retransmission

- Selective repeat innebär att enbart felaktiga meddelanden (ramar, paket, segment etc.) sänds om vid begäran. Go-back-N är i detta fall motsatsen, dvs. vid begäran skickas samtliga meddelanden från och med det första felaktiga på nytt. Implicit retransmission betyder att NAK inte används. Detta innebär att selective repeat with implicit retransmission är jämförbar med idle repeat request with implicit retransmission. Det som skiljer de båda åt är att den förra medger utsändning av många meddelanden innan sändaren inväntar kvittering. Dessutom behövs ingen timer för att kontrollera ACK:ar hos sändaren. (Däremot kan timers användas för kunna att upptäcka avbrutna sändningar.)

Några speciella egenskaper är följande:

- Om meddelande $N + 1$ kvitteras innan meddelande N , sänds meddelande N på nytt.

- Fel på ACK uppfattas av sändaren som ingen ACK.
- Dubbla försändelser kan hanteras av mottagaren.
- Det behövs inga timers för att kontrollera ACK:ar.

Principerna visas i figurerna 5 och 6.

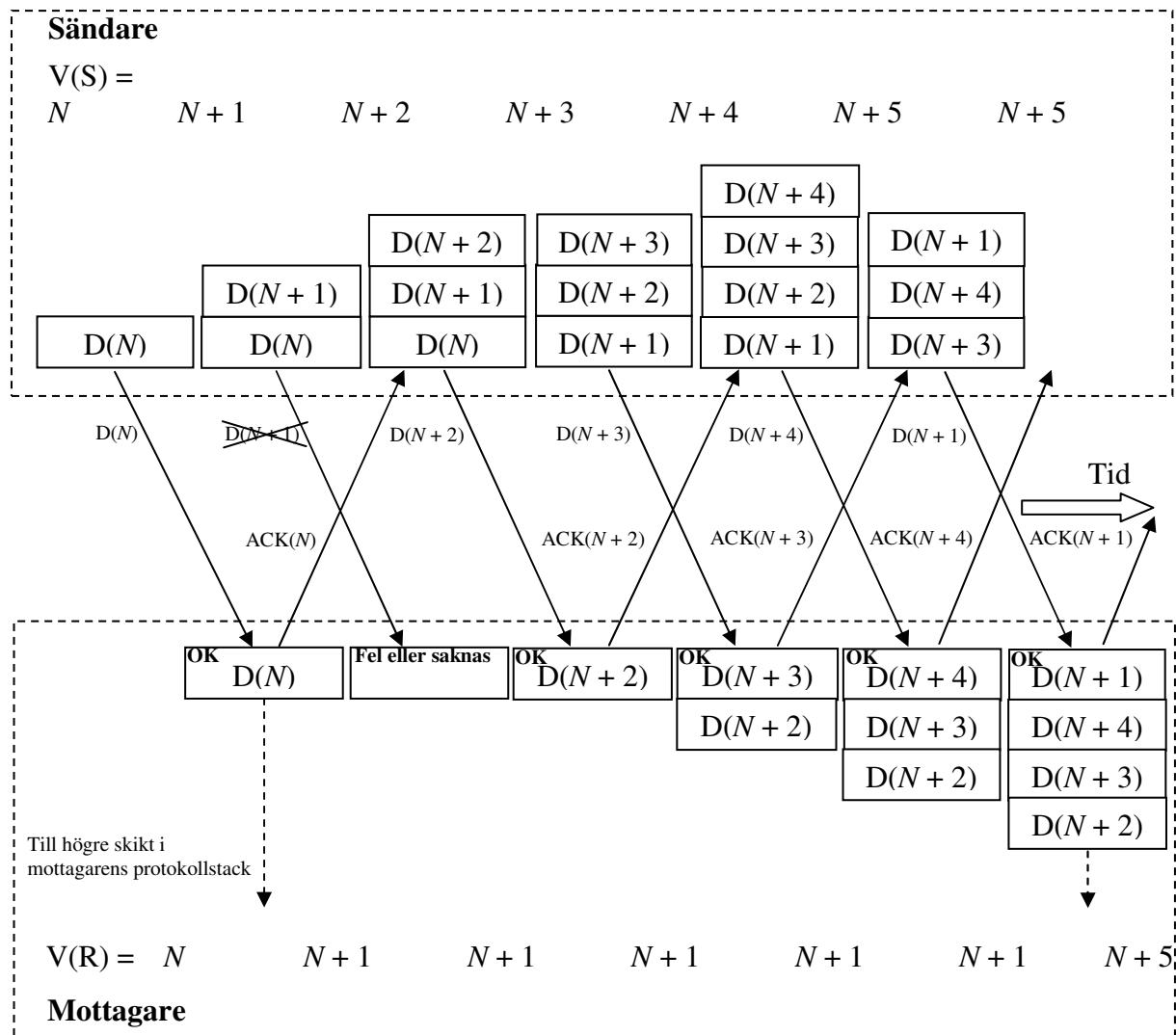
Selective repeat with explicit request

Selective repeat with explicit request är jämförbar med idle repeat request with explicit request, dvs. båge använder NAK. Det som skiljer de båda åt är att den förra medger utsändning av många meddelanden innan sändaren inväntar kvittering.

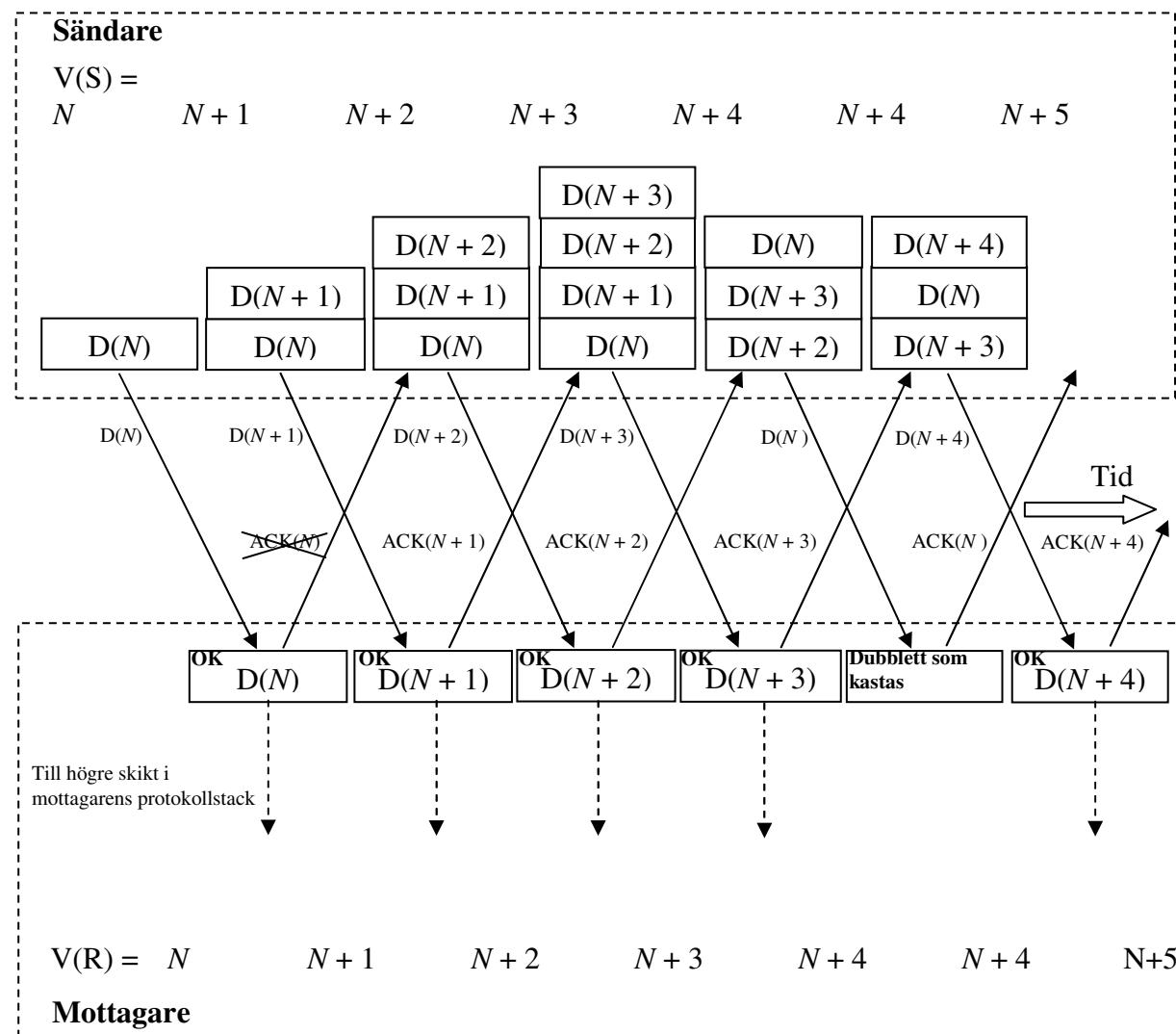
Några speciella egenskaper är följande:

- ACK(N) kvitterar samtliga meddelanden till och med meddelande N .
- Mottagaren har timer för att förhindra uteblivet meddelande vid ev. fel på NAK.

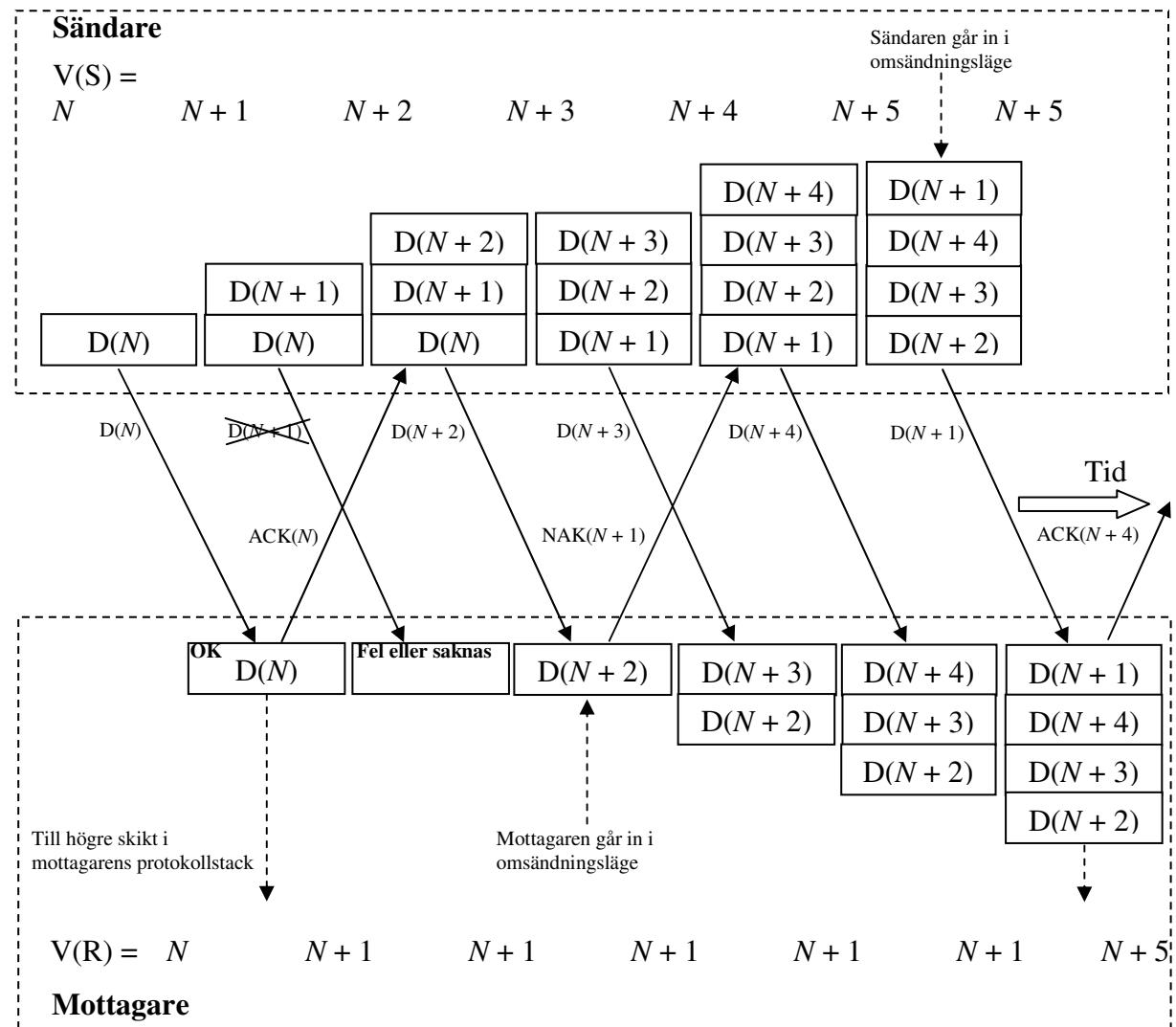
Principerna visas i figurerna 7 och 8.



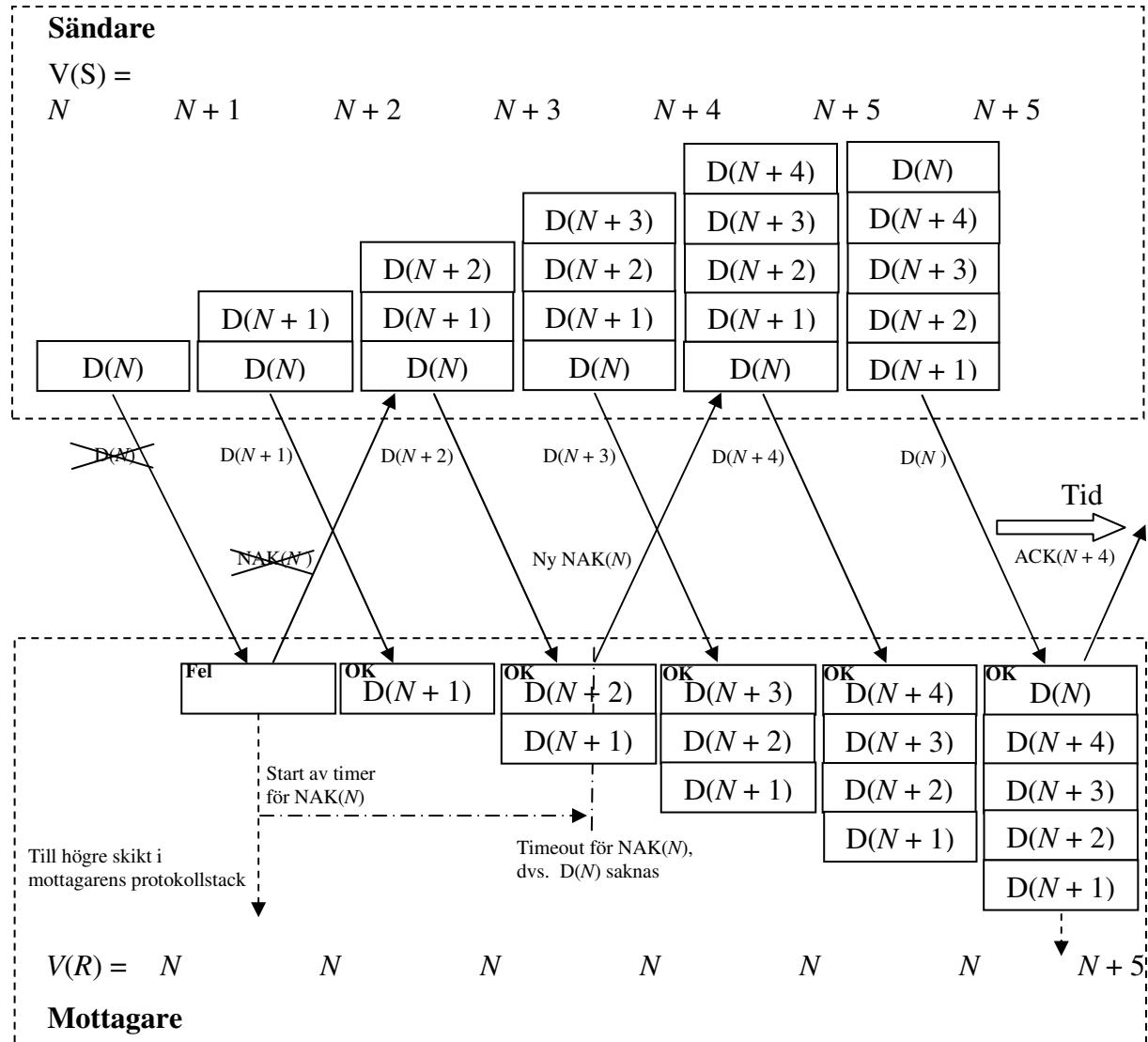
Figur 5. Selective repeat with implicit retransmission – felaktigt meddelande.



Figur 6. Selective repeat with implicit retransmission – felaktig kvittens.



Figur 7. Selective repeat with explicit request – felaktigt meddelande.



Figur 8. Selective repeat with explicit request – felaktig (negativ) kvittens.

Go-back-N

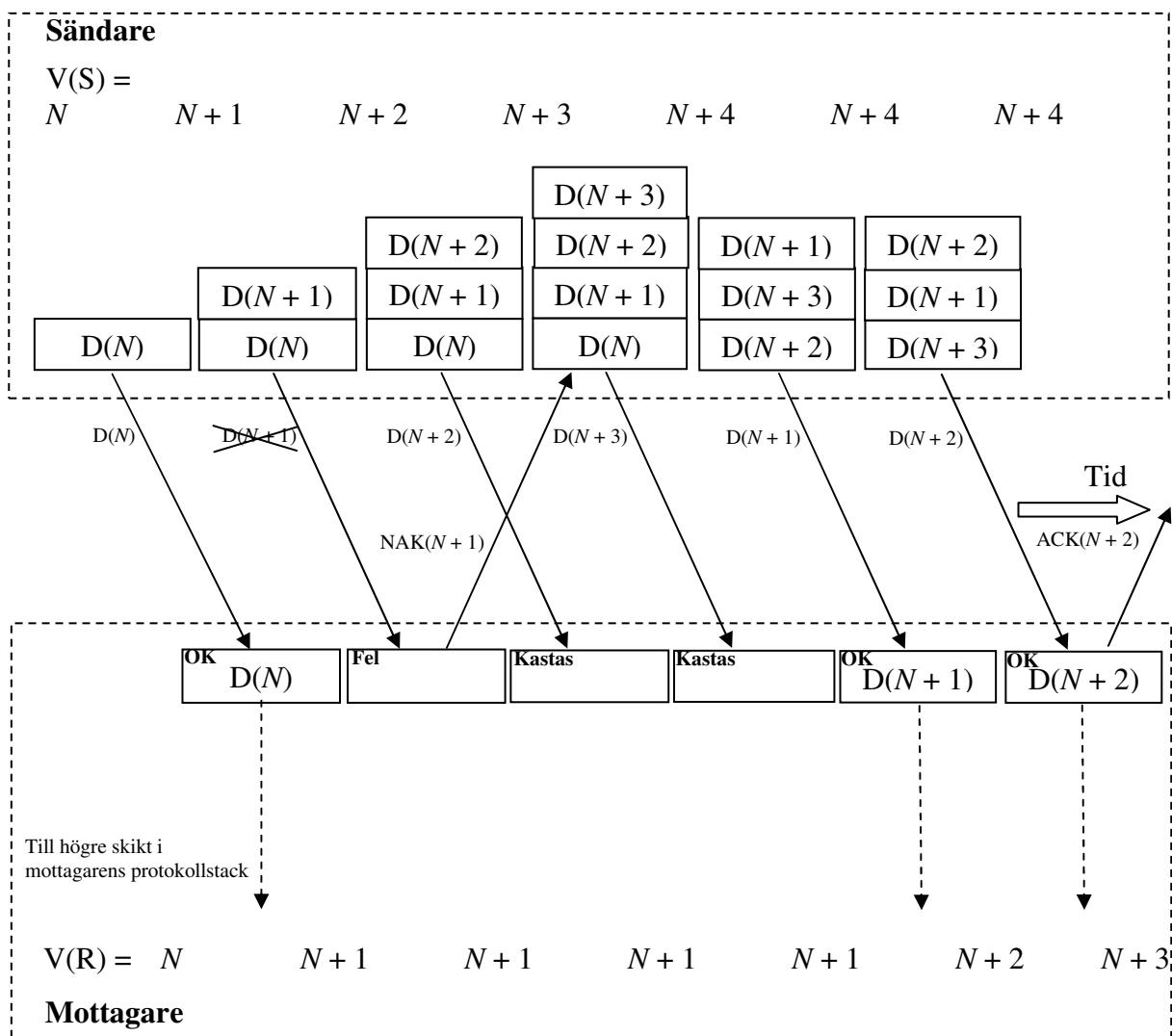
Den stora skillnaden mellan selective repeat och go-back-N är att den senare repeterar samtliga meddelanden från och med ett felaktigt. Detta medför att mottagaren inte behöver spara meddelanden efter ett felaktigt. Alla meddelanden sänds ju på nytt efter ett fel. Den stora fördelen med go-back-N är att det inte behövs en stor mottagningsbuffert. Det räcker faktiskt att mottagningsbufferten har plats för ett enda meddelande.

Några speciella egenskaper är följande:

- Både ACK och NAK används.
- $NAK(N)$ medför omsändning från och med meddelande N .

- $\text{ACK}(N)$ och $\text{NAK}(N + 1)$ ger implicit kvittering för alla meddelanden till och med meddelande N .
- Varje meddelande som sänds har en timer hos sändaren så att utebliven ACK/NAK ger omsändning.

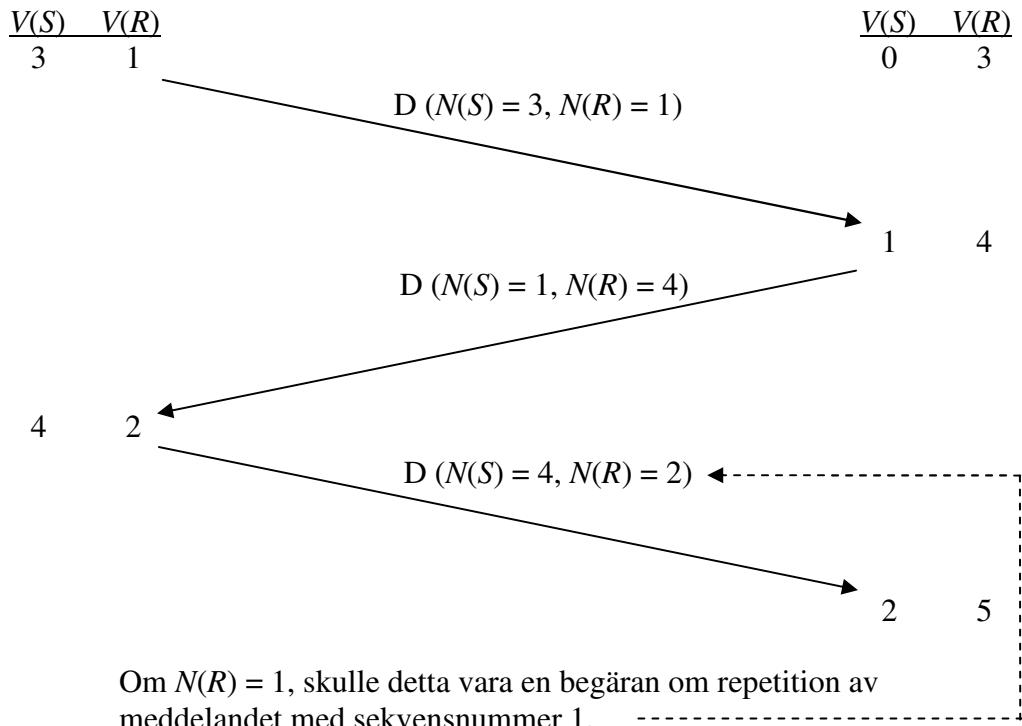
Principerna visas i figur 9.



Figur 9. Go-back-N.

Piggyback acknowledgement

Den tillgängliga bandbredden bör i största möjliga mån utnyttjas för överföring av nyttodata. Därför är det av intresse att inte skicka små meddelanden för kontroll. De kvittenser som mottagaren skickar, både ACK och NAK, kan med fördel bättas in i meddelanden för nyttodata. Denna teknik kallas **piggyback acknowledgement**. Kvitteringen görs genom att den som sänder skriver in två sekvensnummer, ett för det meddelande som skickas $N(S) = V(S)$ och att annat som anger sekvensnumret på nästa förväntade meddelande $N(R) = V(R)$. Observera att $N(R)$ är sändarens önskemål. Egentligen är det bättre att tala om två parter som kommunicerar eftersom bågge bärar in kvittenserna i vanliga meddelanden.



Figur 10. Exempel på piggybacking.

Checksummor

Paritetskontroll

Data som t.ex. tecken kan förses med kontrollbitar s.k. paritetsbitar. En enkel form av paritet är att sändaren beräkna värdet på en bit som av mottagaren används för att kontrollera data. Det finns också förslag på att använda flera bitar för kontroll per data. Blockparitet (blocksumma) innebär att varje data har sin egen paritetsbit. Detta kallas kodord. Sedan beräknas summan, t.ex. addition modulo 2, av flera kodord. Summan är den checksumma som skickas med från sändare till mottagare så att den senare kan kontrollera mottaget data.

För att bestämma värdet för en paritetsbit används antingen udda paritetsteknik eller jämn. Först beräknas antalet logiska ettor i data. Sedan sätts paritetsbiten så att antalet logiska ettor inklusive paritetsbiten blir ett udda tal vid udda paritetsteknik respektive ett jämnt tal vid jämn paritetsteknik.

Mottagaren beräknar antalet logiska ettor i kodordet (data inklusive paritetsbit). Då ska resultatet ge ett udda tal vid udda paritetsteknik och ett jämnt tal vid jämn paritetsteknik, annars är det fel. Problemet är att två bitfel tar ut varandra, dvs. de märks inte så mottagaren gör kontroll. Detta gäller alla jämna antal bitar fel. Därför anses paritetskontroll vara otillräcklig för bl.a. synkron kommunikation.

Hammingavstånd

Ett kodord är data inklusive paritetsbit (eller paritetsbitar). Avståndet mellan två kodord definieras som antalet positioner i kodorden där bitarna är olika. Hammingavståndet (eng. *Hamming distance, HD*) definieras i sin tur som det minsta av alla förekommande avstånd, dvs.

$$HD = \min(\text{alla avstånd i en mängd av kodord})$$

Vi tänker oss en mängd av data som är 0000, 0001 och 0111. Inga fler data förekommer under kommunikation. Vi bildar respektive kodord med udda paritetsteknik och får **10000**, **00001** och **00111**. (Bitarna som är markerade med fet stil är paritetsbitar.) Nästa steg är bestämma samtliga avstånd. I hårdvara och program kan den booleska (logiska) funktionen **exklusiv eller** (XOR) användas för att ta fram antalet bitpositioner som skiljer sig mellan två kodord. XOR kallas också för **addition modulo-2**.

Avståndet mellan 10000 och 00001 är 2 bitar.
Avståndet mellan 10000 och 00111 är 4 bitar.
Avståndet mellan 00001 och 00111 är 2 bitar.

Slutligen erhålls

$$HD = \min(2, 4, 2) = 2.$$

Nyttan med att bestämma Hammingavståndet är att sådana anger antalet bitfel som kan upptäckas samt antalet bitfel som kan korrigeras. Låt u och r vara positiva heltal. Om $HD = u + 1$, kan u st. bitfel upptäckas i ett felaktigt kodord. Om $HD = 2r + 1$, kan r st. bitfel korrigeras i ett felaktigt kodord.

Problem under synkron överföring

Vid synkron överföring är risken stor för fel eftersom det vanligtvis skickas mycket stora datamängder. Då är paritetskontroll inte tillräcklig. Sannolikheten för fel anges med **Bit Error Rate (BER)**. Om $BER = 10^{-4}$ innebär detta att man i medeltal får ett bitfel per 10.000 bitar under en bestämd tidsrymd. Felstorleken kan också anges per block (**block error rate**), dvs. medelvärdet för antalet felaktiga block under en bestämd tidsrymd. Bitfel kan inte bara drabba några enskilda bitar. Sekvenser av bitar kan också bli felaktiga. Då används begreppet **felskur (burst error)**. För att kunna kontrollera mottagna bitar under synkrona överföringar används t.ex. **blockparitet** och **Cyclic Redundancy Check (CRC)**.

Blockparitet

För att visa hur blockparitet (blocksumma) bestäms använder vi ett exempel. Först bestäms paritetsbiten för varje enskilt data (tecken). I nedanstående exempel används udda paritetsteknik. Paritetsbitarna har markerats med fet stil. Sedan beräknas blockpariteten (**Block Code Character, BCC**) som summan av alla kodord. Summan beräknas med addition modulo 2 för respektive bitkolumn, dvs. en kolumnsumma kan antingen bli 0 eller 1. Bitkolumnen med paritetsbitar läggs inte ihop. Istället bestäms paritetsbiten för BCC med udda paritetsteknik med avseende på de övriga bitarna i BCC. Denna paritetsbit har markerats med understrykning. Om vi adderar paritetsbitarna får vi $0 + 1 + 0 + 1 + 1 + 1 = 4$ som modulo 2 är 0. Detta stämmer inte med att antalet logiska ettor i BCC är fyra, dvs. att paritetsbiten ska vara 1. Man kan alltså inte addera paritetskolumnen för att bestämma paritetsbiten i BCC.

$$\begin{array}{ll}
 \mathbf{00000010} & = \text{STX} \\
 \mathbf{10101000} & 1:\text{a tecknet} \\
 \mathbf{00100000} & 2:\text{a tecknet} \\
 \mathbf{10101101} & 3:\text{e tecknet} \\
 \mathbf{11100011} & 4:\text{e tecknet} \\
 \oplus \underline{\mathbf{10000011}} & = \text{ETX} \\
 \text{BCC} = \underline{\mathbf{11000111}}
 \end{array}$$

Under kontroll av överfört data, gör mottagaren samma beräkningar som sändaren. Om $BCC_{\text{mottagare}} = BCC_{\text{sändare}}$, har överföringen gjorts utan fel. Annars har det blivit överförlingsfel.

En alternativ algoritm för att bestämma BCC är att sändaren gör äkta summering (ej modulo-2) av kodorden och ta hänsyn till ord längden, dvs. summan bestäms till modulo $2^{\text{ordlängden}}$. Sedan inverteras alla bitar (1-komplement). Resultatet är BCC. (Observera att det inte nödvändigtvis blir samma BCC med de två algoritmerna.) För kontroll gör mottagaren äkta

summering av alla kodord inklusive BCC och tar hänsyn till ord längden. Om data är felfritt, ska resultatet bestå endast av logiska ettor.

Cyclic redundancy check

En väl beprövad teknik är att använda **Cyclic Redundancy Check** (CRC) som checksumma under dataöverföringar. Vi låter D vara nyttodata med ord längden k bitar, G vara generatorpolynom med ord längden $n + 1$ bitar och R vara rest med ord längden n bitar. Antalet bitar n väljs så att $k > n$. Sändaren beräknar

$$\frac{D \cdot 2^n}{G} = Q + \frac{R}{G}$$

där Q är kvoten och R är en rest. Jämför med

$$\frac{13}{4} = 3 + \frac{1}{4}$$

Sändaren skickar

$$T = D \cdot 2^n + R$$

Kvoten Q är alltså inte av intresse. D multipliceras med 2^n för att lägga in n st. logiska nollor. De är på dessa logiska nollor som bitarna i R läggs till eftersom denna har ord längden n bitar. För att kontrollera inkommende data beräknar mottagaren följande:

$$\frac{T}{G} = \frac{D \cdot 2^n + R}{G} = \frac{D \cdot 2^n}{G} + \frac{R}{G} = Q + \frac{R}{G} + \frac{R}{G}$$

Detta kan till en första anblick verka underligt men om samtliga beräkningar görs modulo 2, blir resultatet lätt att analysera. Varje binärt tal b ger nämligen att $b \oplus b = 0$, där \oplus är addition modulo 2. Vi undersöker detta med följande:

	0	0	0	0		0	1	0	1		1	1	1	0		0	0	1	1	
\oplus	0	0	0	0		\oplus	0	1	0	1	\oplus	1	1	1	0	\oplus	0	0	1	1
	0	0	0	0		0	0	0	0		0	0	0	0		0	0	0	0	

Detta enkla test bevisar inte att $b \oplus b = 0$ men vi kan se det som en bekräftelse på att det inte är fel. Då får vi

$$\frac{T}{G} = Q \oplus \frac{R}{G} \oplus \frac{R}{G} = Q$$

Om T överförs utan fel, kommer mottagaren med beräkningen T/G få en rest som är 0. Annars, om T blir felaktig under överföring, är det stor risk att T/G få en rest som är skild från 0. Säkerheten i denna bedömning bestäms av G . Därför väljs G med stor omsorg i verkliga tillämpningar. Trots detta kan man aldrig vara helt säker på att hitta alla eventuella fel men näst intill alla fel.

Eftersom vi inte vet huruvida T blir felaktig eller ej under överföring, betecknas den bitsekvens som kommer fram till mottagaren med T' . Det är mottagarens uppgift att kontrollera om $T' = T$ eller ej. Antag att det har blivit fel så att $T' = T \oplus E'$, där $E' \neq 0$. Då ger mottagarens beräkning att

$$\frac{T'}{G} = \frac{T \oplus E'}{G} = Q \oplus \frac{E}{G}$$

där $E \neq 0$ är en rest. Säkerheten i detta beror på val av G .

Om $E' = 0$, dvs. inget fel under överföringen, får vi

$$\frac{T'}{G} = \frac{T \oplus E'}{G} = \frac{T}{G} = Q$$

Den kontroll som mottagaren gör är följande:

$$E = \begin{cases} 0, & \text{inga fel antas} \\ & \text{annars fel} \end{cases}$$

Generatorpolynom

Det är av väsentlig betydelse för säkerheten att generatorpolynomet välj med omsorg för att mottagaren ska kunna upptäcka de tänkbara felet. Nedanstående generatorpolynom är vanligt förekommande:

$$\text{CRC-12: } G = x^{12} + x^{11} + x^3 + x^2 + 1$$

$$\text{CRC-16: } G = x^{16} + x^{15} + x^2 + 1$$

(mest i WAN)

$$\text{CRC-CCITT: } G = x^{16} + x^{12} + x^5 + 1$$

(mest i WAN)

$$\text{CRC-32: } G = x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

(mest i LAN)

Med ett bra val av generatorpolynom kan mottagaren upptäcka följande feltyper:

- alla en-bits fel
- alla två-bitars fel
- alla udda antal bitar fel
- alla felskurar $< n + 1$ bitar
- de flesta felskurar $\geq n + 1$ bitar

Med generatorpolynomet CRC-16 som upptäcka alla felskurar < 17 bitar och nästan alla ≥ 17 bitar.

CRC implementeras i hårdvaran med skiftregister och adderare modulo 2 (XOR-grindar). Det förekommer också implementeringar med program.

Reed-Solomon

Till de vanligast förekommande checksummorna för datakommunikation och lagringsmedia hör CRC och **Reed-Solomon** (RS). Utökad CRC (CRC integrerad med en fejlrättande kod) och RS är så bra att de kan användas för att rätta bitfel. Sådana kallas **Error-correcting Codes** (ECC). Beteckningen RS(n, k) innebär att k antal data (s -bitars symboler) tilldelas $n - k$ antal checksummor (s -bitars symboler). Antalet data inklusive checksummorna är n . Låt det maximala antalet data som kan rättas vid fel vara t . Då gäller att $2t = n - k$.

Exempel och övningar

Exempel

Se det tidigare exemplet för Hammingkod.

1. Vi har följande mängd av kodord: 0110010010, 1011110011, 1000101111.
Inga andra kodord kan förekomma. Bestäm samtliga avstånd och HD .
2. Det är vanligt förekommande att en paritetsbit läggs till vid MSB i nyttodata. Detta gör man t.ex. i persondatorns serieport, COM-porten. MSB i ett av våra tal är biten längst till vänster. Därför får vi LSB längst till höger i varje tal. Vi sätter positionsnumret till 0 för LSB. Då blir den andra biten från höger nummer 1, den tredje biten från höger nummer 2, osv. Med jämna bitpositioner avses nr. 0, 2, 4, ... och med udda bitpositioner avses 1, 3, 5, ... Tabell 1 visar ett exempel.

	MSB							LSB
Data	0	1	1	0	1	0	1	1
Position	8	7	6	5	4	3	2	1
Jämna bit-positioner	8		6		4		2	
Udda bit-positioner			7		5		3	
							1	

Tabell 1. Jämna och udda bitpositioner.

Sedan ska vi använda en metoden som Tanenbaum har föreslagit för att skapa två paritetsbitar per kodord. Vi gör experimentet på talen 0000, 0001, ..., 1111. (Det är alltså 16 st. tal.) Gör så här:

- I) Lägg till en paritetsbit för alla bitar med jämna positionsnummer. Använd udda paritetsteknik. Det ger 10000, 00001, ..., 11111. (Du ska ta fram alla 16 kodorden.)
- II) Lägg till ytterligare en paritetsbit för bitar med udda positionsnummer. Använd udda paritetsteknik. Det ger dvs. 110000, 100001, ..., 111111. (Du ska ta fram alla 16 kodorden.)
- III) Jämför alla kodorden från steg II med varandra och ta fram antalet bitar som skiljer dessa åt, dvs. ta fram alla förekommande avstånd. Observera att du inte behöver jämföra två kodord med varandra mer än en gång. På så sätt reduceras arbetet väsentligt.
- IV) Bestäm Hammingavståndet (*HD*).

3. Låt *HD* = 4. Hur många bitfel kan då upptäckas i ett felaktigt kodord?
4. Låt *HD* = 5. Hur många bitfel kan då upptäckas och hur många av dessa kan rättas i ett felaktigt data?

Exempel

Se det tidigare exemplet för blockparitet (blocksumma).

5. Lägg till STX (0000010) och ETX (0000011). Använd udda paritetsteknik för de enskilda kodorden. Bestäm sedan blocksumman för följande meddelande:

0010101 1010011 0011111 0111111

Visa till slut det kompletta meddelandet.

Exempel

Vi ska bestämma det nyttomeddelande som ska sändas tillsammans med tillhörande checksumma (CRC). Nyttodata är 1011 och enligt protokollet ska generatorpolynomet $x^2 + 1$ användas. LSB är längst till höger i nyttodata.

Vi har $D = 1011$, dvs. antalet bitar $k = 4$.

Vi använder generatorpolynomet

$$G = x^2 + 1 = (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{\text{dec}} = 101_{\text{bin}}$$

dvs. antalet bitar är $n + 1 = 3$.

Då är $n < k$ vilket betyder att G räcker till för detta nyttodata. Vi ska alltså ha en checksumma (CRC, R) med två bitar, ty $n = 2$. Detta ger inskjutning av två nollor.

$$D \cdot 2^n = 1011 \cdot (2^2)_{10} = 101100$$

Sändaren beräknar $D \cdot 2^n / G$ enligt följande:

				1	0	0	1
1	0	1	1	0	1	1	0
		\oplus	1	0	1		
		(0)	0	0	1		
		\oplus	0	0	0		
		(0)	0	1	0		
		\oplus	0	0	0		
		(0)	1	0	0		
		\oplus	1	0	1		
		(0)	0	1	0		

Beräkningen ger en rest som är 01, dvs. $R = 01$ som är checksumman (CRC). Sändaren ska skicka $T = D \cdot 2^n \oplus R$. Därför beräknar vi denna summa (modulo 2).

	1	0	1	1	0	0
\oplus					0	1
	0	0	0	0		
	1	0	1	1	0	1

Sändaren skickar $T = 101101$.

CRC-algoritmen steg för steg

Det svåra i bestämningen av CRC är beräkningen av $D \cdot 2^n / G$ hos sändaren och motsvarande kontrollberäkning hos mottagaren. Därför visas här föregående exempel steg för steg:

1. Ställ upp en ”trappa” för beräkning av $D \cdot 2^n / G$ ($= 101100/101$), enligt CRC.

(På detta sätt fick elever i grundskolan lösa divisioner för en tid sedan. Författaren fick lära sig att dividera med denna metod.)

1	0	1	1	0	1	1	0	0

2. Bestäm den bit som ska skrivas på trappsteget ovanför.

Allra första gången är det biten längst till vänster i $D \cdot 2^n$ ($= 101100$). Skriv den lika många bitar in ovanför 101100 som det finns bitar i G ($= 101$), dvs. 3 bitar in.

					1			
1	0	1	1	0	1	1	0	0

3. Multiplicera G ($= 101$) med den bit som skrevs in på trappsteget ovanför, dvs. $101 \cdot 1 = 101$.

Denna produkt (101) skrivs under 101100 med början från vänster.

					1			
1	0	1	1	0	1	1	0	0
			1	0	1			

4. Gör addition modulo-2 (XOR).

Additionen ska beräknas mellan de två 3-bitars talen som nu står ovanför varandra. (I jämförelse med divisionsmetoden som elever i grundskolan för en tid sedan använde gjordes subtraktion istället för addition modulo-2. För övrigt är metoderna mycket lika.) Resultatet blir tydligent 000.

					1			
0	1	1	0	1	1	0	0	
		⊕	1	0	1			
			0	0	0			

5. Stryk den vänstra biten i resultatet (000) och för ner en ny bit.

						1			
1	0	1	1	0	1	1	0	0	
		\oplus	1	0	1		▼		
			(0)	0	0	1			

6. Den bit (0) som nu står till vänster i resultatet (001) skrivs in på trappsteget ovanför.

						1	0		
1	0	1	1	0	1	1	0	0	
		\oplus	1	0	1				
			0	0	1				

7. Multiplisera $G (= 101)$ med den bit som skrevs in på trappsteget ovanför, dvs. $101 \cdot 0 = 000$.

Denna produkt (000) skrivs under 001.

					1	0			
1	0	1	1	0	1	1	0	0	
			1	0	1				
				0	0	1			
				0	0	0			

8. Gör addition modulo-2 (XOR).

Additionen ska beräknas mellan de två 3-bitars talen som nu står ovanför varandra. Resultatet blir tydlig 001.

					1	0			
1	0	1	1	0	1	1	0	0	
			1	0	1				
				0	0	1			
			\oplus	0	0	0			
				0	0	1			

9. Stryk den vänstra biten i resultatet (001) och för ner en ny bit.

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

10. Den bit (0) som nu står till vänster i resultatet (010) skrivs in på trappsteget ovanför.

					1	0	0	
1	0	1	1	0	1	1	0	0
			1	0	1			
				0	0	1		
				0	0	0		
				0'	1	0		

11. Multiplicera G ($= 101$) med den bit som skrevs in på trappsteget ovanför, dvs. $101 \cdot 0 = 000$.

Denna produkt (000) skrivs under 010.

					1	0	0	
1	0	1	1	0	1	1	0	0
			1	0	1			
				0	0	1		
				0	0	0		
					0	1	0	
					0	0	0	

12. Nu upprepas algoritmen på liknande sätt tills det blir slut på bitar att föra ned, det finns ”kvot” på trappsteget ovanför och den sista additionen modulo-2 är beräknad. CRC (som också kallas R) är det två restbitarna 01.

					1	0	0	1
1	0	1	1	0	1	1	0	0
		⊕	1	0	1			
			(0)	0	0	1		
			⊕	0	0	0		
				(0)	0	1	0	
				⊕	0	0	0	
					(0)	1	0	0
					⊕	1	0	1
					(0)	0	1	

Exempel

En mottagare får $T' = 101101$ (LSB längst till höger) och ska enligt protokollet använda generatorpolynomet $x^2 + 1$. Det är alltså samma generatorpolynom som i det föregående exemplet. Vi ska bestämma huruvida T' är lika med det T som sändaren skickade ut eller ej. Därför beräknar mottagaren T'/G .

					1	0	0	1
1	0	1	1	0	1	1	0	1
		\oplus	1	0	1			
			(0)	0	0	1		
			\oplus	0	0	0		
				(0)	0	1	0	
					\oplus	0	0	
						(0)	1	0
							\oplus	1
							(0)	0

Beräkningen ger en rest som är 00, dvs. $E = 00$. Detta betyder felfri överföring. (Om $E \neq 00$, hade det blivit fel under överföringen.)

Exempel

Vilket nyttodata får mottagaren i det föregående exemplet?

Vi använder generatorpolynomet

$$G = x^2 + 1 = (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{\text{dec}} = 101_{\text{bin}}$$

dvs. antalet bitar är $n + 1 = 3$. Detta ger $n = 2$ och detta är lika med antal bitar för R (CRC). Detta betyder att de två sista bitarna (längst till höger) i $T' = 101101$ är R . De bitar som återstår är nyttodata, dvs. $D = 1011$.

Kommentar: I detta fall var $E = 00$, dvs. inget fel under överföringen. Då kan man lita på $D = 1011$. Om vi istället hade fått $E \neq 00$, skulle detta ge ett bestämt D fastän det skulle vi inte kunna lita på.

6. Bestäm CRC-koden och sätt ihop det meddelande som ska sändas. Nyttodata är 10101. (LSB är längst till höger.) Använd generatorpolynomet $x^3 + x + 1$.
7. En mottagare får 11101101 där LSB är längst till höger. Avgör huruvida det har skett fel under överföringen eller ej. Generatorpolynomet är enligt protokollet $x^3 + x + 1$. Bestäm också nyttodata (även om det inte skulle gå att lita på).

8. En mottagare tar emot bitsekvensen 11010111, där LSB finns längst till höger. I bitsekvensen finns kontrollbitar för CRC som har tagits fram med generatorpolynomet $x^3 + x + 1$. Utför felkontroll av bitsekvensen och avgör om det är felfritt eller ej. Visa också vad som är nyttodata i bitsekvensen.
9. En mottagare tar emot bitsekvensen 1101101, där LSB finns längst till höger. I bitsekvensen finns kontrollbitar för CRC som har tagits fram med generatorpolynomet $x^3 + 1$. Utför felkontroll av data och avgör om det är felfritt eller ej. Visa också vad som är nyttodata i data.
10. Bestäm CRC-koden med generatorpolynomet $x^4 + x^3 + x + 1$ för följande nyttodata: 011010. LSB är längst till höger. Sätt sedan samman nyttodata och CRC-koden till det data som skickas.
11. Bestäm CRC-koden med generatorpolynomet $x^3 + 1$ för följande nyttodata: 01010. LSB är längst till höger. Sätt sedan samman nyttodata och CRC-koden till det data som skickas.
12. Reed-Solomon används för att bestämma checksummor. Antalet data (s-bitars symboler) som omfattas av checksummorna (s-bitars symboler) är 223 st. Antalet symboler inklusive checksummorna är 255 st. Hur många data (s-bitars symboler) kan rättas vid fel?
13. Reed-Solomon används för att bestämma checksummor. Antalet s-bitars symboler inklusive checksummorna är 255 st. Bestäm antalet checksummor (s-bitars symboler) för att kunna felsätta 32 data (s-bitars symboler).
14. Reed-Solomon används för att bestämma checksummorna. Antalet data (s-bitars symboler) exklusive checksummorna är 112 st. Antalet s-bitars symboler inklusive checksummorna är 128 st. Hur många data (s-bitars symboler) kan rättas vid fel?

Lösningar

1. Avstånd(**0110010010**, 1011110011) = 5
Avstånd(**0110010010**, 1000101111) = 8
Avstånd(10**11110011**, 1000101111) = 5
 $HD = \min(\text{alla avstånd}) = \min(5, 8, 5) = 5$
2. Kodorden blir 110000, 100001, 010010, 000011, ... De förekommande avstånd är 2 och 4.
Detta ger $HD = \min(\text{alla avstånd}) = \min(2, 4) = 2$.
3. Tre bitfel.
4. Upptäcka fem bitfel och rätta två av dessa.
5. 00000010 00010101 11010011 00011111 10111111 10000011 01100111
6. $T = 10101101$
7. $E = 101$, dvs. överföringsfel. $D = 11101$ men det kan inte användas.
8. $E = 101$, dvs. överföringsfel. $D = 11010$ men det kan inte användas.
9. $E = 001$, dvs. överföringsfel. $D = 1101$ men det kan inte användas.
10. $T = 0110101011$
11. $T = 01010011$
12. 16
13. 64
14. 8

Flödesreglering

Om mottagarens buffert är fyllt, kan den inte ta emot fler meddelanden. Då måste mottagaren kunna meddela detta till sändaren. Det finns olika tekniker för att göra detta. Mottagaren kan t.ex. skicka meddelanden som anger att det är nödvändig med paus eller att överföringen kan fortsätta. Detta kallas **programstyrda flödesregleringar**. Man kan också använda separata ledare för flödesreglering. Detta kallas **hårdvarustyrda flödesregleringar**. Val av omsändningsmetod för med sig att olika sändare och mottagare har bestämd kapacitet beträffande antalet ramar som kan skickas och tas emot innan nästa kvittering. Detta kallas **fönstertechnik**.

Programstyrda flödesregleringar

Mottagaren styr flödet genom att skicka meddelanden till sändaren. Dessa meddelanden kan vara mycket korta och det väsentliga är det kontrolltecken som överförs. XOFF är den generella beteckningen för kontrolltecknet som kan stänga av flödet. Ett praktiskt exempel är DC3 enligt en ASCII-tabell. Motsvarande generella beteckning för att fortsätta överföringen kallas XON som kan realiseras med DC1. Dessa tecken är vanligt förekommande i kommunikationen med enkla skrivare som t.ex. matrisskrivare och radskrivare. Programstyrda flödesregleringar passar bäst för teckenorienterad, asynkron kommunikation och kan användas för full duplex, dvs. sändning i bågge riktningarna samtidigt.

Hårdvarustyrda flödesregleringar

Eftersom hårdvarustyrda flödesregleringar använder separata ledare för kontroll, lämpar sig detta för RS-232D. Mottagaren skickar signaler som generellt kallas **ready** och **busy**. Det är upp till konstruktören att välja lämpliga par av kontrollsinyaler i RS-232D för dessa. Man kan t.ex. använda RTS/CTS och DTR/DSR. Denna typ av flödesreglering kan användas för asynkron kommunikation. Det går att realisera denna flödesreglering för simplex, halv duplex och full duplex.

Fönstertechnik

För varje omsändningsmetod finns det beskrivet vilka storlekar på sändnings- och mottagningsbuffertar som passar för ändamålet. Dessa bestäms av antalet sekvensnummer som ska användas. Anledningen är att meddelanden måste sparas för att kunna repeteras på sändarsidan och kunna ordnas i rätt följd på mottagarsidan. Dessutom får det inte råda någon tveksamhet beträffande sekvensnumren. Dessa återanvänds så att de bildar naturliga talserier modulo 2^n . Det får alltså inte inträffa att t.ex. meddelanden med samma sekvensnummer kan förväxlas. Det maximala antalet meddelanden som sändaren tillåts skicka innan kvittering kallas **sändningsfönster**. På mottagarsidan finns också ett fönster, ett s.k. **mottagningsfönster**. Detta är alltid lika med antalet mottagningsbuffertar, dvs. lika med mottagningskapaciteten. Låt $k = 2^n$, där n är antalet bitar för sekvensnummer i meddelandet (ramen, paketet, segmentet etc.). Då gäller mottagningsfönstren enligt tabell 2.

Omsändningsmetod	Sändningsfönster	Mottagningsfönster
Idle repeat request	1	1
Selective repeat	$k/2$	$k/2$
Go-back-N	$k - 1$	1

Tabell 2. Fönster.

Praktiska exempel

Transportprotokollet TCP ger förbindelseorienterad kommunikation över Internet och intranets. TCP använder en modifierad typ av **go-back-N** för **full duplex**, dvs. **fönsterteknik** och **timers**. Modifieringen innebär att endast felaktiga segmentet sänds på nytt. Däremot sänds inte efterföljande korrekta segment på nytt. Jämför denna modifiering med principen för **go-back-N**. Det finns ett förslag (RFC 2018) att istället använda en version av **selective repeat** i TCP. Denna version kallas **selective acknowledgment**. I TCP begränsas maximalt utskickade TCP-segment av ett **trafikstockningsfönster** och ett **mottagningsfönster**. Vanligtvis är mottagningsfönstret inte flaskhalsen. Istället är det trafiken på Internet som hindrar TCP att sända ut alltför många TCP-segment åt gången.

Linjeproceduren **High-level Data Link Control** (HDLC) finns i flera versioner. HDLC ger bitorienterad kommunikation i **full duplex** för både punkt till punkt-förbindelser och flerpunktförbindelser. I HDLC kan omsändningsmetoden väljas. Det finns **selective repeat** och **idle repeat request**, dvs. **fönsterteknik** och **timers**.

En router på ARPANET kallas **Interface Message Processor** (IMP). Kommunikationen mellan IMP:er görs i åtta fysiska kanaler. Varje kanal använder **idle repeat request**, dvs. **fönsterteknik** som har sändningsfönster = mottagningsfönster = 1. För att skydda start- och stoppflaggor görs **byte stuffing** enligt DLE-versionen. Vid sändning används första lediga kanal, dvs. alla åtta fysiska kanaler delar på en gemensam sändningskö. Om man betraktar kommunikationen från IMP till IMP blir den totala funktionen lika med **selective repeat with implicit retransmission** vars sändningsfönster = 8. Om det är fråga om kommunikation via satelliter, används 16 fysiska kanaler, dvs. den totala funktionen över IMP-satellit är lika med **selective repeat with implicit retransmission** vars sändningsfönster = 16.

Exempel och övningar

Exempel

Fältet för sekvensnummer i en ram har tre bitar. Vi ska bestämma sändningsfönster, mottagningsfönster och sekvensnummer för följande omsändningsmetoder:

- Idle repeat request
- Selective repeat
- Go-back-N

Vi har $n = 3$ som ger $k = 2^n = 2^3 = 8$ och får följande svar:

- a. Sändningsfönster = mottagningsfönster = 1.
Eftersom kvittering görs efter varje ram, behövs endast två sekvensnummer. Vi kan använda t.ex. 000 och 001.
- b. Sändningsfönster = mottagningsfönster = $k/2 = 8/2 = 4$.
Denna omsändningsmetod bygger på att alla sekvensnummer används, dvs. 000, 001, ..., 111.
- c. Sändningsfönster = $k - 1 = 8 - 1 = 7$.
Mottagningsfönster = 1.
Denna omsändningsmetod bygger på att alla sekvensnummer används, dvs. 000, 001, ..., 111.

Exempel

Skulle vi ha svarat annorlunda i det förra exemplet för implicit retransmission och explicit request i a och b?

Svaret är enkelt. Beträffande sändningsfönster, mottagningsfönster och sekvensnummer är det ingen skillnad mellan de två versionerna av idle repeat request respektive selective repeat.

1. Fältet för sekvensnummer i en ram har två bitar. Omsändningsmetod är idle repeat request. Bestäm fönster och sekvensnummer.
 - a. Sändningsfönster
 - b. Mottagningsfönster
 - c. Sekvensnummer
 - d. Är det någon skillnad mellan implicit retransmission och explicit request i a, b och c?
2. Fältet för sekvensnummer i en ram har fyra bitar. Omsändningsmetod är selective repeat. Bestäm fönster och sekvensnummer.
 - a. Sändningsfönster
 - b. Mottagningsfönster
 - c. Sekvensnummer
 - d. Är det någon skillnad mellan implicit retransmission och explicit request i a, b och c?
3. Fältet för sekvensnummer i en ram har fem bitar. Omsändningsmetod är go-back-N. Bestäm fönster och sekvensnummer.
 - a. Sändningsfönster
 - b. Mottagningsfönster
 - c. Sekvensnummer

Lösningar

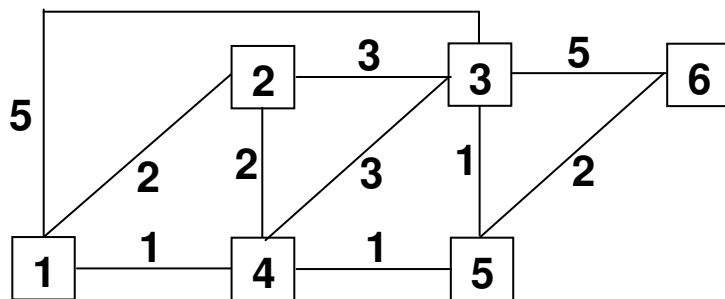
- | | | | |
|----------|------|----------------|--------|
| 1. a. 1 | b. 1 | c. 00 och 01 | d. Nej |
| 2. a. 8 | b. 8 | c. 0000–1111 | d. Nej |
| 3. a. 31 | b. 1 | c. 00000–11111 | |

Övningar på LS

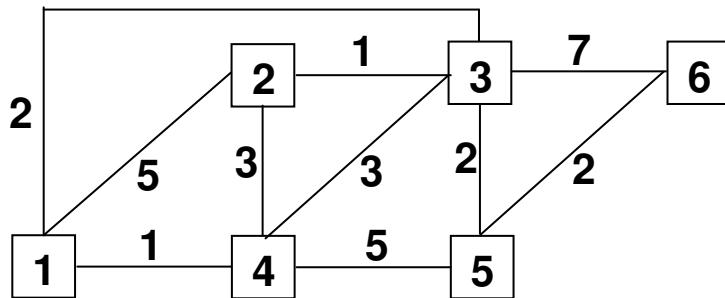
Följande övningar är avsedda för routingalgoritmen Link State (LS) med Dijkstras algoritm.

Uppgifter

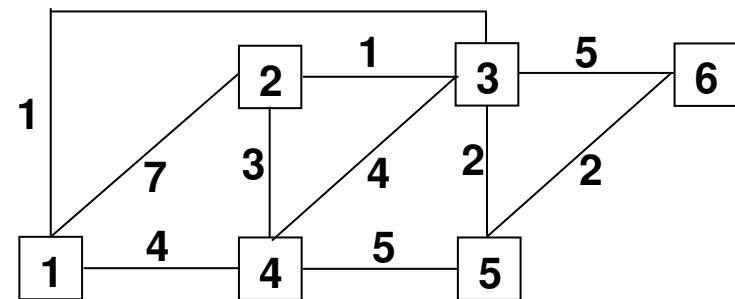
- Bestäm de bästa vägarna för nedanstående nät sett från router 1 till samtliga routrar i nätet. Använd LS. Kostnaderna antas vara symmetriska på vägarna.



- Bestäm de bästa vägarna för nedanstående nät sett från router 1 till samtliga routrar i nätet. Använd LS. Kostnaderna antas vara symmetriska på vägarna.

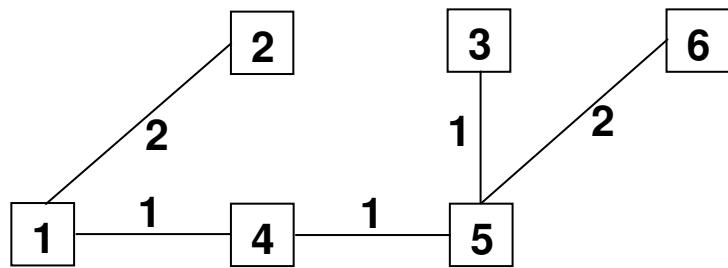


- Bestäm de bästa vägarna för nedanstående nät sett från router 1 till samtliga routrar i nätet. Använd LS. Kostnaderna antas vara symmetriska på vägarna.

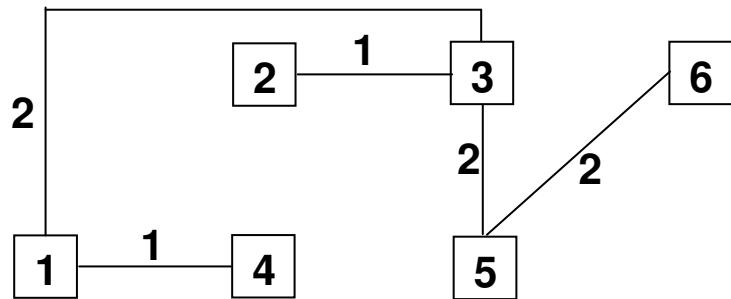


Lösningar

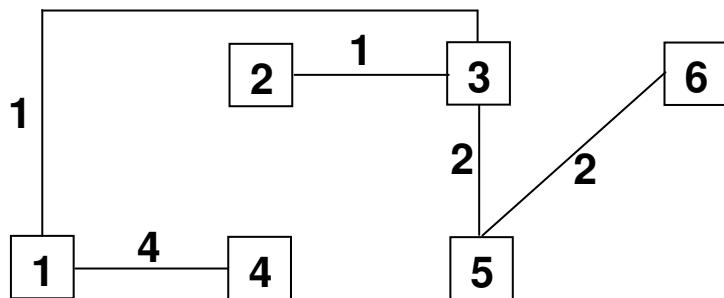
1.



2.



3.



Lokala nät

Typiskt för lokala nät

Komponenterna i ett **lokalt nät** (**Local Area Network**, LAN) är t.ex. persondatorer med nätverkskort, kabelförbindelser, sammankopplande enheter och skrivare. Grunden för lokala nät är utan tvekan de relativt billiga persondatorerna. Det var på 1980-talet som man började masstillverka billiga mikroprocessorer. Därmed kunde man bygga ut terminalerna så att även dessa kunde genomföra beräkningar. Tidigare hade beräkningar gjorts i den centrala datorn och användarna kommunicerade med denna via ointelligenta terminaler men det fanns även intelligenta terminaler. Med persondatorer kunde man bygga en helt ny typ av nät där beräkningarna kunde göras i godtyckliga datorer. Man kom att dela in persondatorer i servrar och arbetsstationer. Servern kom att ta över vissa av den centrala datorns uppgifter som t.ex. lagring av centrala program och gemensamma filer. Observera att man kan klara sig bra utan en enda server i ett lokalt nät eftersom samtliga persondatorer har kapacitet att lagra program och filer. Trots detta är det praktiskt att använda servrar. Man kan definiera en server som en persondator med speciella uppgifter i det lokala nätet. Arbetsstationer är opålitliga eftersom de kan stängas av och inte finns tillgängliga. En server har fördelen att nästan alltid vara i drift. Servrar brukar användas för en eller flera uppgifter, t.ex. hantera användarkonton och säkerhet, hantera utskrifter, tillhandahålla applikationsprogram, tillhandahålla webbsidor, köra brandväggsprogram och tillhandahålla databaser.

Ytterst är servrar program som tillhandahåller tjänster. Det är dessa tjänster som klienterna kan anropa. Av praktiska skäl kallas man även datorerna som innehåller serverprogram för servrar. Däremot kallas datorn som innehåller klienter för klient. Sådana datorer kallas hellre arbetsstationer eller stationer.

En av pionjärerna inom lokala nät var Apple. Deras persondatorer som kallas **Macintosh** var bland de första att vara fullt utrustade för att anslutas till lokala nät. Idén var att kunna dela på dyrbara resurser. Istället för att varje Macintosh var ansluten till sin egen matrisskrivare, anslöts persondatorerna till en dyr gemensam laserskrivare. Fildelning fungerade utmärkt i Apples nät. Varje Macintosh kunde ha mappar (kataloger) utdelade för nätverksåtkomst. Apples nät kallas **LocalTalk** och det protokoll (accessmetod) som används kallas **AppleTalk**. Numera säljs Macintosh-datorer fullt utrustade för det lokala nätet Ethernet. Trots detta finns det långsamma LocalTalk kvar i persondatorerna.

Några typiska egenskaper för ett lokalt nät är följande:

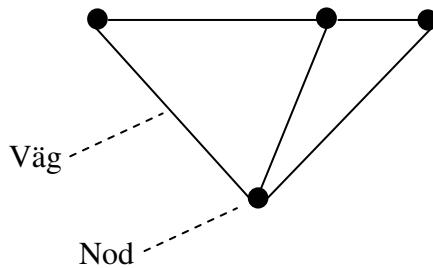
- Finns på ett geografiskt begränsat område, t.ex. i en kontorsbyggnad eller på ett universitetsområde. Observera att flera lokala nät kan kopplas ihop till ett **site wide LAN** med ett **ryggradsnät (backbone)**.
- Höga bithastigheter, t.ex. 10 Mbps, 100 Mbps och 1 Gbps.
- Trafiken mellan två noder avlyssnas av de övriga, åtminstone de som är anslutna till samma kabelsegment.
- **Broadcast-meddelanden** kan användas, dvs. alla i det lokala nätet kan ta emot samma meddelande.

- Det lokala nätet ägs vanligtvis av företaget eller organisationen, dvs. ingen myndighet bestämmer hur det ska ”se ut” i detta.

Lokala nät är alltså nät som är begränsade till sin geografiska utbredning. Om nätet täcker stora områden, talas det om **Metropolitan Area Network** (MAN) och **Wide Area Network** (WAN). MAN är stadsnät och ännu större nät kallas WAN. De senare kan till och med vara globala.

Topologier

Med **topologi** avses nätverksstruktur, dvs. hur enheter i nätet har kopplats samman. Generella begrepp för att beskriva nätverksstrukturer är **noder** och **vägar**. Exempel på noder i ett lokalt nät är persondatorer, skrivare och sammankopplande enheter. Vägarna i lokala nät kan vara trådbundna förbindelser (med kablar som har antingen metalledare eller optiska fibrer) och trådlösa förbindelser (med radio eller IR). Observera att vägar kan vara antingen enkelriktade eller dubbelriktade.



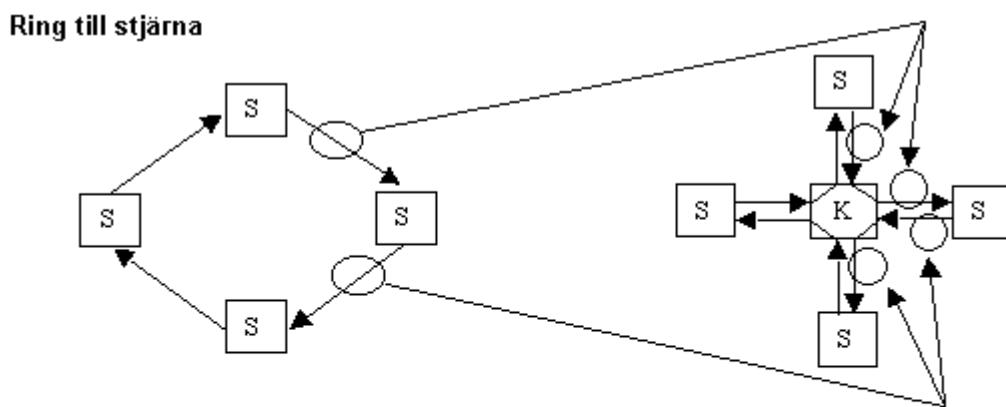
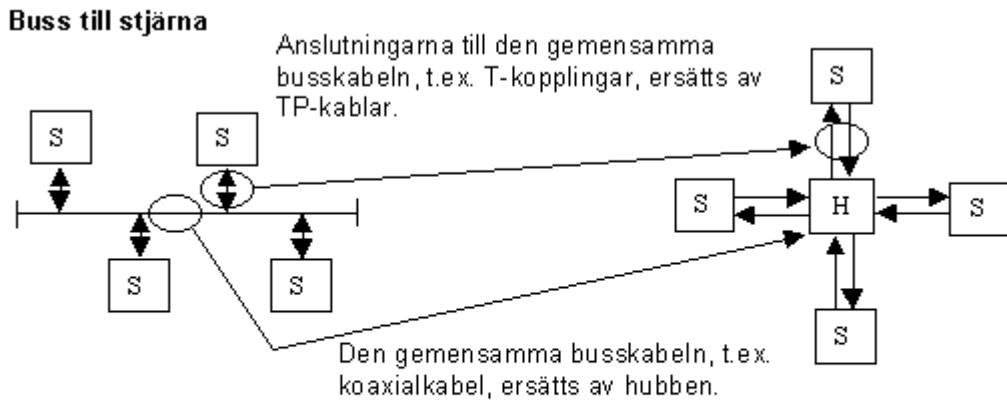
Figur 1. Nätverksstrukturer.

Lokala nät brukar beskrivas med två huvudtyper av topologier, **logiska** och **fysiska**. Den logiska topologin anger hur meddelanden tar sig från nod till nod. I lokala nät används för det mesta antingen **buss** eller **ring**. På senare tid har det kommit lokala nät som förutsätter att det är **stjärnor** med **switchar** som nav. Detta ger **punkt till punkt-förbindelser**. Det typiska för en buss är att alla noder på denna hör samma meddelande (ungefär) samtidigt även om inte alla är mottagare. Detta kallas **flerpunktsförbindelser (multidrop)**. I en ring skickas meddelanden från nod till nod, dvs. alla noder hör inte samma del av ett meddelande samtidigt. Varje nod skickar meddelanden vidare till sin granne i trafikrikningen, dvs. inte till alla andra i ringen. Liksom på en logisk buss behöver nödvändigtvis inte alla noder vara mottagare till ett meddelande. Däremot är alla noder i en ring med i överföringen av ett meddelande. Därför ger även ringar flerpunktsförbindelser.

Den fysiska topologin kan vara lättare att förstå. Den beskriver nämligen hur kablarna dras samt hur radio- och ljussignaler skickas från nod till nod. Förr var det vanligt förekommande att den fysiska topologin var lika med den logiska. Numera används vanligtvis inte fysiska bussar och ringar. Man har funnit det vara praktiskt att dra kablarna till en sammankopplande enhet. Kablarna från noderna liknas vid ekrarna i ett cykelhjul. Den sammankopplande enheten kallas följaktligen nav (eng. *hub*). Denna typ av fysisk topologi kallas **stjärna**. I lokala nät används huvudsakligen buss, ring och stjärna som fysisk topologi, och som sagt, på senare tid domineras stjärnan. (Det finns andra topologier än buss, ring och stjärna fastän i lokala nät är dessa sällsynta.) I figur 2 visas hur logisk buss respektive ring realiseras med fysiska stjärnor.

Naven benämns olika i stjärnor som realiseras bussar respektive ringar. På bussar används **hubbar**. I det enklaste fallet är hubben en flerportsrepeterare som slaviskt skickar meddelanden vidare. På senare tid har hubbar ersatts av **switchar**, dvs. växlar som kan avläsa mottagarens hårdvaruadress (**Medium Access Control Address**, MAC-adress). Switcharna skickar meddelanden endast på den port och kabel där mottagaren förväntas finnas. Detta kallas switching. Switchar har förmågan att minska den totala trafikbelastningen på en stjärna

eftersom de gör adressering på länknivå. I ringar kallas navet **koncentrator**, **kabelkoncentrator** eller **ringkoncentrator**. En sådan sammankopplande enhet har till uppgift att koppla så att den logiska ringen förblir intakt. Till detta hör förmågan att känna av om enheterna i noderna är i drift och att de fungerar som de ska i ringen. Ringkoncentratorn ska koppla bort avstängda och trasiga noder för att bibehålla ringen.



S = station
H = hubb
K = koncentrator, ringkoncentrator, kabelkoncentrator

Figur 2. Realiseringar med stjärnor.

Bryggor och switchar

En switch ersätter hubbens funktion som nav i stjärna och har dessutom förmåga att göra adressering (eng. *switching*). Detta innebär att mottagaradressen i en ram avgör vilken utport som switchen väljer att öppna. Switchen är alltså ett nav med bryggfunktion (eng. *bridging*). Både switching och bridging handlar nämligen om att filtrera ramar (eng. *frame filtering*).

Bryggor och switchar har tabeller (eng. *forwarding databases, routing directories*) som anger utport för respektive mottagares MAC-adress. På logiska (och fysiska) bussar är det vanligt förekommande att bryggor och switchar är självlärande. Genom att studera en rams

avsändaradress får bryggan/switchen vetskap om rätt val av utport för denna adress. Principen kallas **backward learning**.

Det kan uppstå tillfället då mottagarens MAC-adress inte finns i tabellen. Då tillämpas **flooding**, dvs. ramen skicka ut på samtliga portar i förhoppning om att mottagaren ska finnas någonstans i nätet. Däremot bör ramen inte skickas tillbaka på importen eftersom det kan uppstå en kopia av ramen.

Om mottagarens adress finns i tabellen och den finns på samma port som sändaren, skickar bryggan/switchen inte ramen vidare. I sådana fall får mottagaren ändå ramen. Det är endast när mottagaren befinner sig på en annan port än sändaren som bryggan/switchen behöver skicka ramen vidare.

För att hålla tabellerna aktuella raderas dessa med jämma mellanrum. Ett lämpligt tidsintervall för **återställning** (reset) av en tabell rör sig om några minuter. Detta förfarande skyddar tabellerna mot inaktuell information och mot att de blir överfulla.

Fjärrbryggan har en speciell uppgift som är att koppla samman del-LAN som befinner sig på olika platser. Det förekommer att fjärrbryggor är försedda med modem för kommunikation över mellanliggande nät som t.ex. telefonnät.

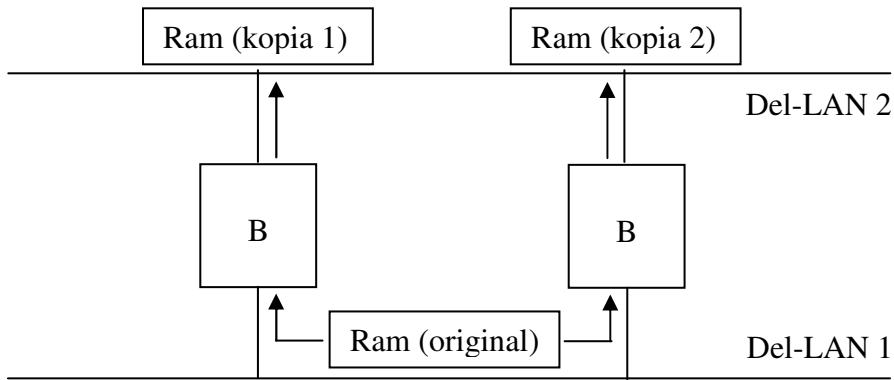
Bryggor/switchar har potential för att kunna koppla ihop lokala nät av olika typ. Detta betyder nödvändigtvis inte att man alltid använder bryggor/switchar för ändamålet. Det är nämligen tekniskt komplicerat att överföra ramar mellan lokala nät av olika typ. Följande problem måste lösas:

- Olika ramformat.
- Olika maximala ramlängder.
- Olika bit hastigheter.
- Om näten använder olika krypteringar eller mottagande nät inte använder kryptering, görs dekryptering. Detta är en säkerhetsrisk eftersom dekrypteringen inte görs hos mottagaren.
- Olika Quality of Service (QoS).

Det kan vara av fördel att använda en router för att konvertera mellan olika lokala nät. En sådan måste naturligtvis vara avsedd för att konvertera mellan de avsedda typerna. En router går upp till nätverksskiktet eftersom paketet tas ut ur ramen. Då är det naturligt att lägga in paketet i en ny ram som passar i det mottagande nätet.

Ett problem med att använda fler än en brygga mellan två bussar är att det skapas flera vägar mellan näten. Om flera bryggor vidarebefordrar samma ram från en buss till en annan, blir det flera kopior av samma ram på mottagande buss. Om dessa kopior skickas vidare av andra bryggor, kan nätet bli fyllt av ramkopior som lamslår trafiken. Hursomhelst kommer mottagaren att få flera exemplar av samma ram. Problemet illustreras i figur 3. Bryggorna (B) uppfattar att mottagaren finns i del-LAN 2. Därför skickar bryggorna ramen vidare till del-

LAN 2. Detta resulterar i två kopior av ramen. Observera att ramkopiorna inte skickas ut exakt samtidigt. Då blir det kollision. Ramkopiorna skickas ut efter varandra sett i tiden.



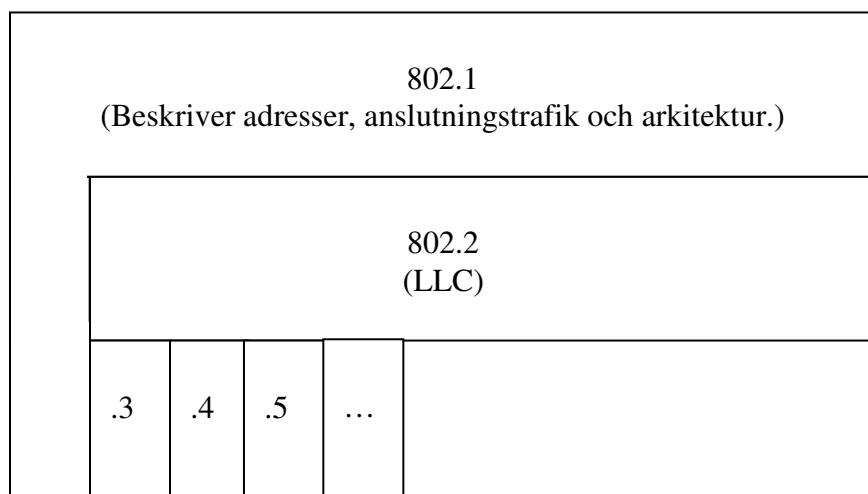
Figur 3. Ramkopior.

Algoritmen **Spanning tree** används i bryggorna för att ta bort alternativa vägar. Målet är att det ska finnas endast en logisk väg mellan varje par av sändare och mottagare i trädstrukturen trots att det kan finnas flera fysiska. Algoritmen förhindrar att det kan bli flera kopior av samma ram på en buss (delbuss i trädstrukturen).

Principer för accessmetoder

Med **accessmetod** (**åtkomstmetod**, MAC) avses tekniken med vilken noderna kommer åt transmissionsmediet och hur sändningsrätten fördelas. I **IEEE 802-serien** förutsätts att respektive MAC samarbetar med **Logical Link Control** (LLC) i datalänkskiktet. MAC och LLC är p.g.a. detta centrala begrepp för lokala nät. (IEEE 802-serien motsvaras av **ISO 8802/3**.)

En överblick av IEEE 802-serien ges av figur 4. De olika näten finns nederst som t.ex. Ethernet-standarden (802.3), Token bus (802.4), Token ring (802.5), MAN med fiberteknik (802.6), bredband i kabel-TV-nät (802.7), fiberoptik i LAN (802.8), integrering av ljud och bild i LAN (802.9), datasäkerhet i LAN (802.10), trådlösa LAN (802.11) och nätet 100VG-AnyLAN (802.12).

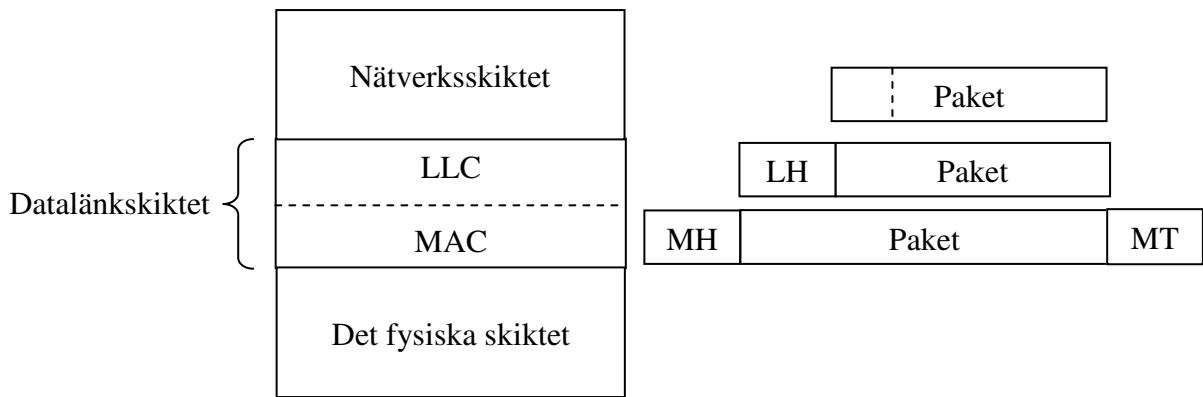


Figur 4. Strukturen i IEEE 802-serien.

Det kan vara av intresse att se hur IEEE 802-serien sammanfaller med OSI-modellen. Detta visas i figur 5. De lokala näten är linjeprocedurer, dvs. de hör till länkskiktet. Det är också i detta skikt som ”bryggning” görs med de högre protokollen med hjälp av LLC. På så sätt görs de högre protokollen oberoende av den MAC-typ som används. Denna ”isolation” gör det relativt enkelt att byta mellan olika MAC:ar.

I figur 5 visas hur ett paket från nätverksskiktet läggs in i LLC-ramens info-fält. Förutom paketet består LLC-ramen av LLC Header (LH). Fälten i LH visas i figur 7. De kallas **Source Service Access Point** (SSAP), **Destination Service Access Point** (DSAP) och Control. SSAP och DSAP fungerar på liknande sätt som NSAP i OSI-modellen. Se avsnittet **Referensmodeller**. MAC-ramen består av MAC Header (MH), LLC-ramen och en MAC Trailer (MT). MT är lika med ramens checksumma.

Flera typer av lokala nät beskrivs ingående i IEEE 802-serien både beträffande datalänkskiktet och det fysiska skivet. Därför kan man påstå att ett lokalt nät omfattar OSI-skikt 1 och 2 med reservation för LLC-delen. (Originalversionen av Ethernet använder till skillnad från IEEE 802.3 inte LLC-delen.)



Figur 5. IEEE 802-serien i OSI-modellen.

LLC ger också val av **Quality of Service** (QoS) För detta ska vara möjligt tillhandahåller LLC tjänster som är oberoende av aktuell MAC. De tre LLC-tjänsterna är följande:

1. Okvitterad, förbindelselös tjänst
2. Kvitterad, förbindelselös tjänst
3. Kvitterad, förbindelseorienterad tjänst

Dessutom finns det en fjärde specialtjänst som kallas **obtain reply**. Denna används för att hämta meddelanden från buffertar i andra noder utan att upprätta förbindelse. Obtain reply används också för att uppdatera den egna meddelandebufferten.

LLC är en kopia av **Link Access Procedure, Balanced** (LAPB) som är en delmängd av den bitorienterade linjeproceduren **High-level Data Link Control** (HDLC).

En stor uppgift för MAC är adresseringen. Detta kan kallas för hårdvaruadressering eftersom MAC-adresserna vanligtvis finns i hårdvara som t.ex. nätverkskort. Det är vanligtvis fråga om 12 hexadecimala tecken per MAC-adress. Dessa är unika, dvs. en MAC-adress får endast förekomma i en upplaga som t.ex. på ett enda nätverkskort för Ethernet, åtminstone inom ett och samma LAN. (Däremot kan samma MAC-adress förekomma på olika håll i världen eftersom $2^{12 \cdot 4} = 2^{48} \approx 2,82 \cdot 10^{14}$ adresser annars inte skulle räcka till för det totala behovet.) MAC-adresserna kallas även LAN-adresser och ibland används namnet på nätverkstypen. I Ethernet kan man alltså tala om Ethernet-adresser. Det är det amerikanska ingenjörsförbundet IEEE som delar ut MAC-adresser. En tillverkare får köpa en serie av sådana för sina nätverkskort och sammankopplande enheter. (Broadcast-adressen är vanligtvis FF-FF-FF-FF-FF-FF.)

En indelning av accessmetoder är att ange den logiska topologin för respektive metod. I följande avsnitt beskrivs kortfattat några accessmetoder som lämpar sig för bussar och ringar. Det finns också accessmetoder som förutsätter stjärna. Vi ska även beskriva ett par sådana.

Contention utan bokning

Det finns en rad accessmetoder som bygger på principen **contention**, dvs. först ut får rätten att sända. Om kollision skulle inträffa, avbryter de som sänder för att efter slumpmässiga tider göra nya försök att sända. Idén kommer från Hawaii på 1970-talet då en forskare undersökte lämpliga accessmetoder för ett radionät med placering på de många öarna. Sådana accessmetoder ledde senare och i ett annat sammanhang fram till den version som kallas **Carrier Sense Multiple Access with Collision Detection** (CSMA/CD). Ethernet bygger på CSMA/CD. Denna accessmetod beskrivs delvis av sitt namn. CS står för att alla måste lyssna om det är ledigt på bussen innan något sänds. När någon nod sänder, lyssnar även denna för att kunna uppfatta eventuella kollisioner. MA innebär att alla noder delar på samma kanal, det gemensamma transmissionsmediet. CD är den del av accessmetoden som upptäcker och hanterar kollisioner.

Topologi: logisk buss på fysisk buss eller i stjärna.

Exempel: Ethernet.

Contention med bokning

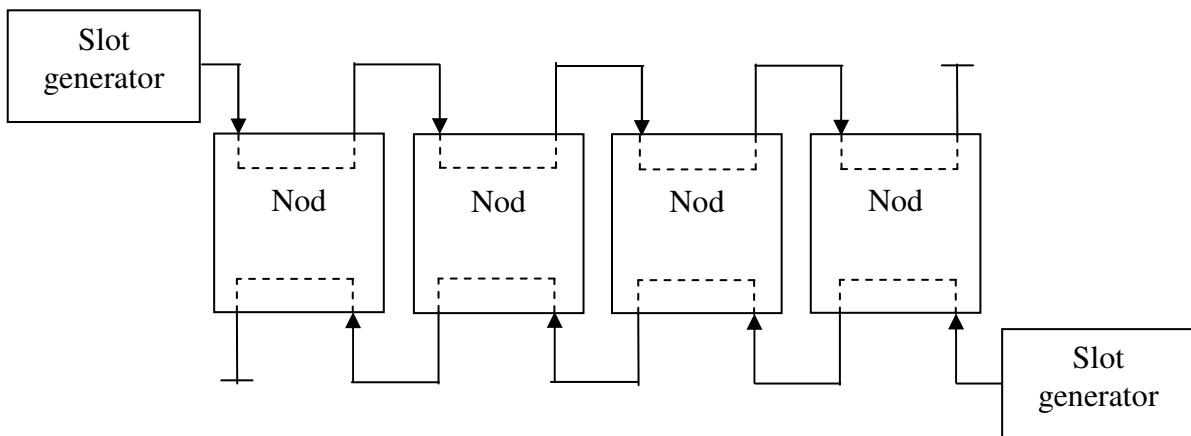
Av namnet framgår att **Carrier Sense Multiple Access with Collision Avoidance** (CSMA/CA) hör till principen contention. Skillnaden mellan CSMA/CA och CSMA/CD är att den förra använder bokningar. En nod som vill sända lyssnar om det är ledigt innan den skickar ut ett bokningsmeddelande (preliminär bokning, ett bestämt bitmönster). Om det inte blir kollision för detta bokningsmeddelande, har noden gjort definitiv bokning som innebär att den under en bestämd tid får skicka nyttomeddelanden. CSMA/CA ger mindre antal kollisioner än med CSMA/CD.

Topologi: logisk buss på fysisk buss eller i stjärna.

Exempel: AppleTalk (LokalTalk) och IEEE 802.11 (Wi-Fi, ett trådlöst LAN).

Dubbelbuss med decentraliserad kontroll

Noderna ansluts till två bussar som används för trafik i var sin riktning. Transmissionskanalen delas upp med TDM. För att kunna boka för sändning används en kontrollbit i varje slot. Bokningen görs i en slot som skickas på den motsatta bussen, dvs. sändningarna på en buss kontrolleras med hjälp av kontrollbitar i de ramar som skickas på den andra bussen och vice versa. Information om bokningar som en nod gör skickas alltså till noderna som finns före denna. Innan en nod får sända på en buss måste den släppa förbi lika många tomma slots som efterkommande noder har bokat. Varje nod håller kontroll på antalet bokade slots (andra noders bokningar) och räknar ned för varje tom slot som släpps förbi. Detta kräver en räknare för varje buss.



Figur 6. Dubbelbuss.

För att göra bokningsfunktionen mer användbar, kan prioritetsnivåer införas. Vanligtvis kan bokningar göras med fyra prioritetsnivåer. Detta kräver förstås en kontrollbit per prioritetsnivå i varje slot. Då behöver noderna en räknare per prioritetsnivå och buss.

När en nod vill sända, används en räknare som räknar ned. Det finns en sådan för varje prioritetsnivå och buss. Räknaren för nedräkning tilldelas samma värde som motsvarande bokningsräknare. När räknaren har kommit ned till 0, används nästa tomma slot för sändning. Eftersom noderna inte känner till varandras positioner på dubbelbussen, måste ett och samma meddelande skickas ut på bågge bussarna. Detta behöver nödvändigtvis inte ske exakt samtidigt eftersom de bågge bussarna kan vara i otakt som kan orsakas av t.ex. omsändningar.

I ena änden av respektive buss finns en enhet som genererar slots (slot generator), dvs. en enhet som markerar början på varje slot. Det kan vara till fördel att placera de båda slot generators nära varandra eftersom det underlättar för administratörerna. Dessutom kan man samköra enheterna så att de synkroniseras. Ytterligare en finess är att använda en och samma slot generator för båda bussarna. Om bågge slot generators placeras nära varandra eller om det är en och samma enhet för bågge bussarna, kommer den fysiska topologin att se ut som en ring. Däremot kommer bussarna inte att förändras, dvs. de förblir bussar.

Topologi: logisk buss (dubbelbuss) på fysisk buss.

Exempel: Distributed Queue Dual Bus (DQDB).

Stjärna med centralisering av kontroll

Navet i en stjärna kan tilldelas kontrollfunktioner som t.ex. polling av noder som är anslutna till dess portar. Sådana nav kallas koncentratorer men observera att koncentrator är ett tekniskt begrepp som förekommer i andra sammanhang. En koncentrator ”pollar” sina portar i tur och ordning för att hitta noder som vill sända.

Det är också möjligt att använda bokningsförfarande med prioritetsnivåer. Vanligtvis används två prioritetsnivåer.

Koncentratorer kan kopplas till varandra. Polling av portar görs med början i den första koncentratorn. När turen har kommit till en port där en underliggande koncentrator ansluts, görs polling av den underliggande koncentratorns portar innan polling fortsätter i den första koncentratorn. Man kan tänka sig såväl flera underliggande koncentratorer till den första koncentratorn likaväl som flera koncentratorer på djupet. Denna typ av nät förutsätter att det är fysiska stjärnor.

En nod som vill sända får vid polling klartecken att skicka en ram till koncentratorn som buffrar ramen. Sedan vidarebefordrar koncentratorn ramen till den port där mottagaren befinner sig. Om mottagaren inte finns ansluten direkt till denna port, finns den längre ned trädstrukturen. När en nod sänder, har den ensamrätt till stjärnan. De övriga noderna har beordrats att vila.

Algoritmen som hanterar polling och bokning kallas **round-robin scheduling**.

Topologi: punkt till punkt-förbindelse i stjärna.

Exempel: 100VG-AnyLAN.

Stjärna med växlande nav

På senare tid har det kommit mycket snabba lokala nät som enbart använder switchar istället för hubbar som nav. Det nödvändigt att använda switchar i stjärnor för att erhålla växlande nav. Switchar har nämligen förmåga att välja utport. Stjärnor med switchar kan på så sätt ge punkt till punkt-förbindelser. Detta utnyttjas i lokala nät som kan sända med full duplex. Detta är möjligt tack vare punkt till punkt-förbindelser över switchar som dessutom har förmåga att vid behov kunna buffra ramar (store-and-forward). I Ethernet-versioner som sänder och tar emot på olika TP-ledarpar är det inte nödvändigt med store-and-forward. Sådana switchar kan arbeta efter principen som kallas cut-through. Om switchen också ska användas för neddelning av bithastighet, t.ex. från 100 till 10 Mbps, behövs store-and-forward.

Topologi: punkt till punkt-förbindelse i stjärna.

Exempel: X-versionerna av Ethernet (med CSMA/CD bortkopplad) som t.ex. 1000BASE-SX, 1000BASE-LX och 1000BASE-CX.

Stafettnät i ring

I stafettnät används en speciell ram för att tilldela rätten att sända. Denna kallas **token** och är att jämföra med en stafettpinne. Den nod som har token har rätt att sända. Egentligen brukar det vara så att token är en minimal ram som vid behov kan kompletteras med kontrollinformation och nyttodata. Token skickas runt till noderna enligt den naturliga ordning som finns i en fysisk ring. I fysiska stjärnor erhålls en logisk ringtopologi. Access-metoden fungerar alltså i både fysiska ringar och fysiska stjärnor. I både token och den data-ram som den blir vid sändning finns kontrollbitar för bokning. Dataramar kan delas in i två kategorier: användardata och ringkontroll. Det behövs nämligen flertalet kontrollramar för att få igång nätet och för att upprätthålla dess funktioner.

Topologi: logisk ring i fysisk ring eller stjärna.

Exempel: Token ring (enkelring) och Fiber Distributed Data Interface (FDDI, dubbel- och enkelring).

Stafettnät på buss

Ett stafettnät i ring har en naturlig ordning för att skicka runt token mellan noderna (datorerna). Denna sändning kan göras även på en logisk buss med hjälp av en **sändningslista** som anger ordningen. Eftersom noderna kan tilldelas en maximal tid för att sända (maximal tid att behålla token), kan en maximal cirkulationstid garanteras. Det finns möjlighet att införa bokningsförfarande med vanligtvis fyra prioritetsnivåer.

Topologi: logisk buss med sändningslista på fysisk buss och i stjärna.

Exempel: Token bus och Profibus.

Ethernet

Ethernet II och IEEE 802.3

Bob Metcalfe och David Boggs är de som konstruerade Ethernet. Redan under det tidiga 1970-talet arbetade Bob med ARPAnet vid MIT som doktorand från Harvard University. På så sätt var Bob välunderättad om arbetet med Internet men det var det trådlösa ALOHAnet som senare inspirerade Bob och David till att påbörja arbetet med Ethernet. ALOHAnet utarbetades av Norm Abramson vid University of Hawaii.

Norm och hans kollegor undersökte olika accessmetoder (ALOHA-familjen). Idén var att kunna kommunicera mellan Hawaiis många ör med hjälp av radio på endast två kanaler. Bortsett från typiska radioproblem är det liknande problematik med trådlösa nät som med kabelbundna, i synnerhet för logiska/fysiska busstopologier. Norm hade gjort en noggrann utredning av detta.

Bob arbetade vid Xerox Parc (företagets forskningscentrum i Palo Alto) då experimenten med Ethernet påbörjades. Han var inte bara inspirerad av ALOHAnet utan också av persondatorerna från Alto Computers. Bob såg tydligt framför sig behovet av att kunna koppla ihop persondatorer i nät.

Ingredienserna i arbetet med Ethernet var alltså **ARPAnet**, **ALOHAnet** och **persondatorer**. Egentligen var idén med nät för datorer inte ny. Principerna för ringnät hade presenterats av svensken Söderblom i ett patent på 1950-talet. Däremot dröjde det fram till hösten 1985 innan IBM lanserade sitt Token ring (IEEE 802.5). Det var nämligen utvecklingen av kiselchip (integrerade kretsar med kisel) som medförde att det betydligt mer komplexa nätet Token ring kunde tillverkas på ett praktiskt sätt.

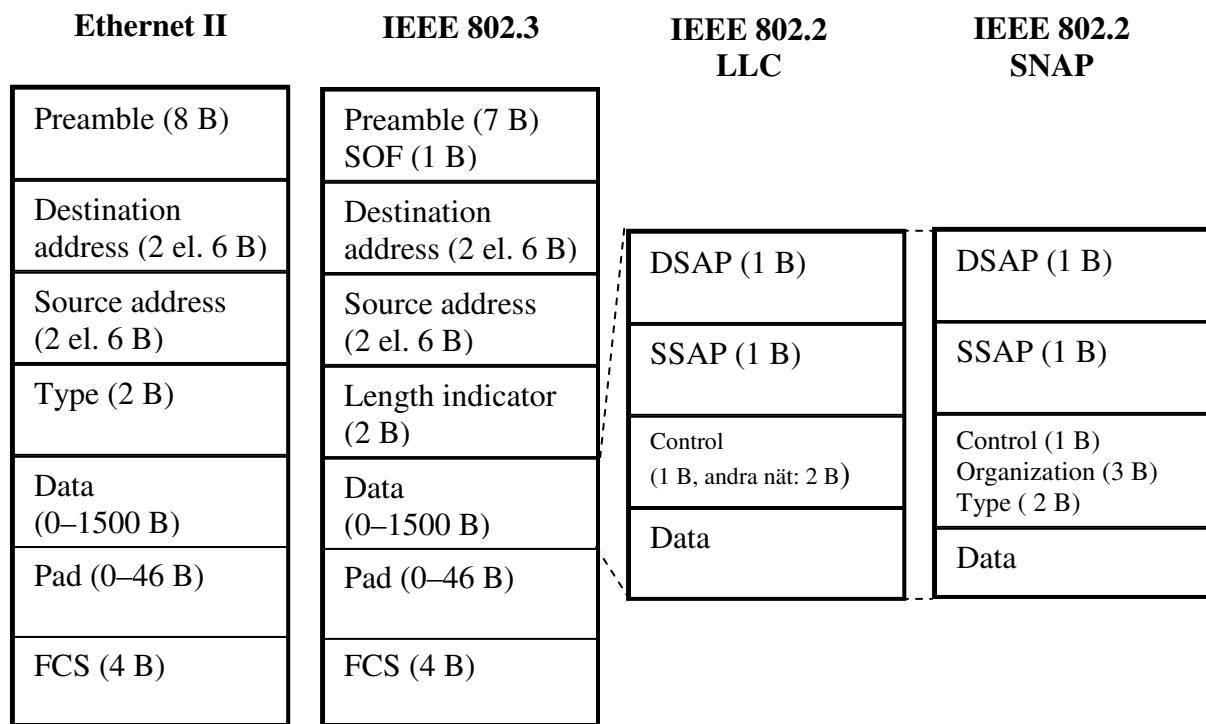
Den första versionen av Ethernet utvecklades redan under mitten av 1970-talet. Under år 1980 bildade Bob företaget 3Com som har blivit en av de största tillverkarna på marknaden av nätverkskort och sammankopplande enheter. Samma år (1980) publicerade Digital, Intel och Xerox (DIX) den första versionen av Ethernet. Redan 1983 kom den andra versionen av Ethernet från Digital. Den brukar kallas Ethernet II och används av bl.a. DECnet och gamla Internet-versioner. Då började det amerikanska ingenjörsförbundet IEEE att intressera sig för produkten. Detta ledde fram till standarden **IEEE 802.3**.

IEEE 802.3 skiljer sig något från Ethernet II beträffande ramformatet och dessutom omfattar standarden användande av LLC. Ramformatet visas i figur 7. Observera att ramen för IEEE 802.3 omfattar fält för LLC eller som alternativ **Data Link Control** (DLC) för IBM-baserade nät. Användandet av LLC ger som resultat att en LLC-ram placeras inne i IEEE-ramens info-fält.

En annan version av LLC är **Subnet Access Protocol** (SNAP). I SNAP har fältet Type (i Ethernet II) flyttats till SNAP header. SNAP används av moderna Internet-versioner och Apple. I Internet är SDAP och SSAP satta till AA_{hex}, Control till 03_{hex} och Organization till 000000_{hex}.

NetWare är en server från Novell. I denna finns möjlighet att använda alla typer av Ethernet-ramar, även ramtypen med SNAP. Eftersom Netware-nätet tidigare var baserat på transport-

och nätprotokollet **Internet Packet Exchange** (IPX), finns också en ramversion som kallas **802.3 raw packet**. Denna innehåller fält för IPX. Numera används vanligtvis TCP/IP i NetWare-näten.



Figur 7. Jämförelse av ramformaten.

Preamble är ett synkroniseringsfält, dvs. mottagarna ska bli förberedda på den inkommende ramen. Det är ett åtta oktetter långt fält, dvs. 8 B långt. I Ethernet II består varje oktett av 10101010 vardera. Däremot är det endast sju sådana oktetter i 802.3. I den åttonde oktetten görs en variation som ska ange att i nästa oktett börjar destination address. Variationen görs i den åttonde biten genom att den inte är logiska nolla, dvs. den åttonde oktetten är 10101011. Detta kallas **Start of Frame** (SOF).

Destination och source address är fält med plats för två eller sex oktetter vardera. I normala fall används sex oktetter. Det ger plats för 12 hexadecimala tecken. Detta överensstämmer med antalet tecken i MAC-adresserna (hårdvaru-, nod- och Ethernet-adresserna).

Type är ett fält som anger nätverksprotokoll. I 802.3-ramen används istället en length indicator för storleken (antalet oktetter) på data. I Ethernet II anges istället meddelandets längd i ett särskild fält.

De ovanstående fälten sammanfattas med MH i figur 5, dvs. det är kontrollinformationen i början av ramen (header).

Pad-fältet används för att garantera att ramarna inte blir mindre än den minsta tillåtna ramstorleken. Det är av betydelse för överföringssäkerheten i Ethernet. I 802.3-ramen finns inte endast data utan en hel LLC-ram. Denna består av en LLC-header efterföljt av ett paket (data från nätverksskiktet).

Sist i ramen finns **Frame Check Sequence** (FCS). Denna checksumma bestäms med metoden **Cyclic Redundancy Check** (CRC). Mottagaren använder samma algoritm för att kontrollera mottagen ram. FCS kallas MT i figur 5, dvs. kontrollinformationen i slutet av ramen (trailer). I fall av överföringsfel kasseras ramen omedelbart. Mottagaren begär i sådan situation **ej** omsändning. Detta innebär att Ethernet är ett **ramförmedlande** (eller paketförmedlande) nät till skillnad från kretskopplade nät. I protokoll på högre nivåer kallas detta förbindelselöst. I viss litteratur står det av den anledningen att **Ethernet är förbindelselöst**. Om man använder TCP/IP över Internet, kommer TCP att hantera omsändningar. Motsvarande gäller också för SPX/IPX, där SPX är förbindelseorienterat. Däremot kommer varken UDP/IP eller IPX (utan SPX) att ge omsändningar. Beträffande ramkollisioner är det sändarens uppgift att reagera på jam. Detta beskrivs nedan.

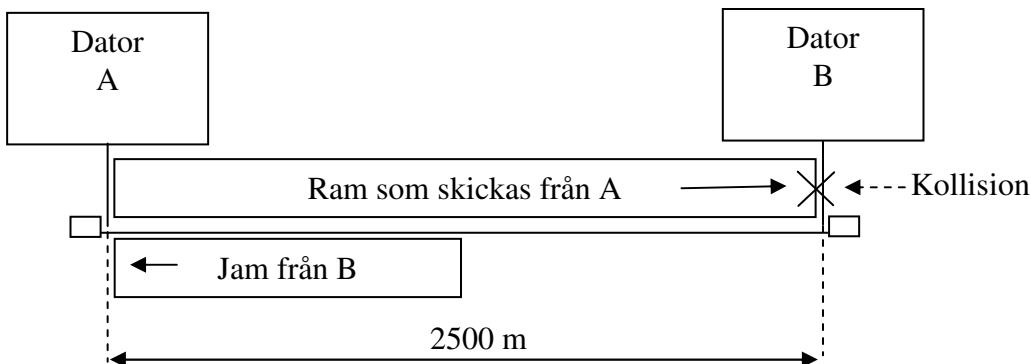
Den variant av ALHO-familjen som används i Ethernet kallas **Carrier Sense Multiple Access with Collision Detection** (CSMA/CD). CS innebär att alla enheter (datorer, hubbar, switchar etc.) lyssnar på trafiken. Så gör även en sändande enhet. Den behöver medhörning för att upptäcka kollisioner. (I stjärnnäten är det naven som upptäcker kollisioner.) Sändarna lyssnar i synnerhet innan sändningsförsök för att undvika onödiga kollisioner. En ram kan sändas tidigast tiden **interframe gap** efter den senaste ramen förutsatt att bussen är ledig. MA betyder att alla enheter har åtkomst av ”det gemensamma mediet”. I Ethernet har enheterna lika rätt att sända på ”det gemensamma mediet”, dvs. enheterna sänder utan prioritetsordning. I fysiska bussar motsvaras ”det gemensamma mediet” av koaxialkablarna och i fysiska stjärnor av hubbarna. (Då switchar används som nav i stjärnnäten har sändaren och mottagaren/mottagarna ensamrätt till ”det gemensamma mediet”.) CD är den del i accessmetoden som upptäcker och rättar till kollisioner på den logiska bussen. Efter krock görs nya sändningsförsök vid slumpade tider efter en bestämd algoritm.

Väntetiden innan omsändning är lika med R -slot time, där heltalet R slumpas i intervallet $[0, 2^K]$ och $K = \min(\text{sändningsförsök nr}, 10)$. Vid det första omsändningsförsöket efter krock lyssnar noden om det är ledigt och sänder omedelbart ($R = 0$) eller efter en hel slot time ($R = 1$). Om det då blir kollision, växer intervallet för R . Eftersom $K = \min(\text{sändning nr}, 10)$, blir K aldrig större än 10. Detta kallas **backoff limit**. Algoritmen för att bestämma väntetid kallas **(truncated) binary exponential backoff**. Den nod som upptäcker kollision skickar ut en störsignal som kallas **jam** (32 bitar). När sändande nod hör störsignalen, avbryter den genast sändningen och väntar den slumpade tiden R -slot time. På så sätt hoppas man att noderna vars sändningar har kolliderat inte ska göra nya försök exakt samtidigt.

Omsändningsförsök nr	K	R
1	1	0, 1
2	2	0, 1, 2, 3
...
10	10 (backoff limit)	0, 1, 2, ..., 1023
11	10	0, 1, 2, ..., 1023
...
16 (max)	10	0, 1, 2, ..., 1023

Tabell 1. Variabler för binary exponential backoff.

Slot time $51,2 \mu\text{s}$ grundar sig på att noderna på var sin ände av en maximalt lång fysisk buss ska hinna uppfatta jam vid kollision. Den maximala längden är $5 \cdot 500 = 2500 \text{ m}$. Segmentlängden 500 m erhålls med tjock koaxialkabel. Anta att dator A i figur 8 skickar en ram. Något senare gör dator B detsamma så att det blir kollision nära dator B. Detta innebär att den första biten i ramen som A sänder har utbrett sig 2500 m. (Det måste vara fråga om någon av de allra första bitarna i ramen. Annars skulle dator B ha upptäckt ramen från dator A innan sitt sändningsförsök.) Dator B sänder ut ett jam som måste hinna fram till dator A innan den sista biten i ramen har skickats ut. Annars kommer inte dator A att förstå att det har blivit kollision under sändningen och att ramen ska upprepas.



Figur 8. Bestämning av slot time.

Slot time är den tid som det **måste** ta att skicka en bit i en ram 2500 m inklusive att skicka en bit i ett jam samma sträcka. Dessutom tillkommer en säkerhetsmarginal.

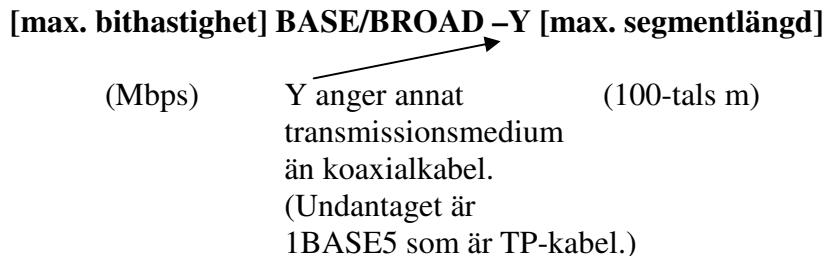
$$\text{slot time} = 2 \cdot (\text{utbredningstiden}) + (\text{säkerhetsmarginalen})$$

Slot time har satts till $51,2 \mu\text{s}$ i IEEE 802.3. Om den maximala bithastigheten 10 Mbps, kan den **minimala ramstorleken** beräknas. Vi får

$$\text{minimal ramstorlek} = (\text{slot time}) \cdot (\text{maximal bithastighet}) = 51,2 \cdot 10^{-6} \cdot 10 \cdot 10^6 = 512 \text{ b}$$

Om vi bestämmer den minsta ramstorleken genom att använda minimala fält och inget meddelande (data) i figur 7, får vi $(8 + 2 + 2 + 2 + 0 + 0 + 4) \cdot 8 = 144 \text{ b}$. Då fattas det $512 - 144 = 368 \text{ b}$ för att det ska vara en godkänd (minimal) ram. Detta är anledningen till pad-fältet (utfyllnaden). Enligt figur 7 ska pad vara maximalt 46 B = 368 b. Det är precis det antal bitar som saknas för att ramen kan godkännas. Pad används för att fylla ut ramar så att de blir minst 512 b = 64 B.

Det finns åtskilliga versioner av Ethernet för en och samma maximal bithastighet. Den stora skillnaden är olika transmissionsmedia. För att underlätta finns det en speciell kabelbeteckning.



Figur 9. Kabelbeteckning.

Exempel på kabelbeteckningar:

- 1BASE5 (StarLAN, telefonråde)
- 10BASE2 (tunn koaxialkabel, RG58, maximalt 185 m)
- 10BASE5 (tjock koaxialkabel, RG11)
- 10BASE-T (TP-kabel)
- 10BASE-FP (fiber passive)
- 10BASE-FL (fiber link)
- 10BROAD36 (bredband, radio i koaxialkabel, 2·1800 m)

Observera att den maximala segmentlängden enligt 10BASE2 är 200 m men att den praktiskt användbara är högst 185 m.

5-4-3-regeln innebär att maximalt **fem** kabelsegment får kopplas ihop. Då behövs **fyra** repeterare (förstärkare). Man får ansluta datorer och annan utrustning på maximalt **tre** av de fem kabelsegmenten. Det är tänkt att de två kabelsegmenten som inte ansluter utrustning endast ska förlänga bussen. Även optisk fiberkabel kan användas för de två länkande segmenten. Sådana länkar kallas **Fiber Optic Interrepeater Link** (FOIRL). Med tunn koaxialkabel erhålls maximalt $5 \cdot 185 = 925$ m och med tjock koaxialkabel maximalt $5 \cdot 500 = 2500$ m.

Tillverkaren Cisco har speciell teknik för långa länkar (3.500'–5.000' eller 1.067–1.524 m). Bithastigheterna 5–15 Mbps utlovas över TP-kablar (kat. 1–3). Detta kallas Cisco **Long-Reach Ethernet** (RLE). Tanken är att kunna använda gamla typer av TP-kablar för överföring av ljud, video och data för t.ex. Internet-anslutning, video-strömmar (streaming video) och IP-telefoni.

Isochronous Ethernet (IsoEthernet) är en utbyggnad av 10BASE-T som förutom datatrafik med 10 Mbps omfattar en multimediasanal med bithastigheten 6,144 Mbps. IsoEthernet kan användas på nät med 10BASE-T. De datorer som (också) ska kommunicera med 6,144 Mbps behöver speciella nätverkskort. I optiska fibersystem för telefoni kan bithastigheten jämföras med 96 kanaler med vardera 64 kbps, dvs. $96 \cdot 64 = 6144$ kbps = 6,144 Mbps. IsoEthernet beskrivs av standarden IEEE 802.9. Koden som används kallas 4b5b. Detta innebär att fyra bitar kodas med fem bitar som på Fiber Distributed Data Interface (FDDI). 4b5b ger effektivare kodning än med Manchester.

Ethernet är det lokala nät som utvecklas i snabbast takt. Standarden IEEE 802.3 som beskriver accessmetoden CSMA/CD användes under lång tid och används fortfarande. På senare tid har det tillkommit versioner för höga bithastigheter som t.ex. 100 Mbps och 1 Gbps.

Vi summerar IEEE 802.3 med följande:

Slot time = 51,2 μ s

Minsta ramstorlek = 512 b = 64 B

Interframe gap = 9,6 μ s

(Interframe gap är alltid lika med 96 bitars tid oavsett bithastighet.)

Maximal bithastighet = 10 Mbps

Maximal kabellängd på fysisk buss (med fem segment,

$5 \cdot 500\text{m} = 2500\text{ m}$

Maximal längd på en droppkabel (till tjock koaxialkabel) = 50 m

Maximal längd på en TP-kabel = 100 m

Kod (för kopparkablar): Manchester

(Se någon grundläggande kurs i datakommunikation beträffande Manchester-koden.)

Fast Ethernet

Fast Ethernet beskrivs av standarden IEEE 802.3u med accessmetoden CSMA/CD, dvs. i grunden detsamma som IEEE 802.3. Det finns en version av Fast Ethernet som avviker från CSMA/CD. Denna kallas **full duplex**. I Fast Ethernet används följande parametrar:

Slot time = 5,12 μ s
Minsta ramstorlek = 512 b = 64 B
Interframe gap = 0,96 μ s = 960 ns
Maximal bithastighet = 100 Mbps
Maximal längd på en TP-kabel = 100 m

Kodningen beror på val av transmissionsmedium. För 100BASE-T4 används koden 8b6t. (Se någon grundläggande kurs i datakommunikation beträffande koden.) Detta innebär att åtta bitar kodas med tre spänningsnivåer fördelade på sex tidsdelar. Med TP-kabel (lägst kategori 3 UTP) som har fyra ledarpar sänds det på tre ledarpar (sändaren använder det fjärde ledarparet för att kollisionsdetektering) och de övriga noderna lyssnar på tre ledarpar. Ett ledarpar används enbart för sändning, ett ledarpar enbart för mottagning och två ledarpar används i bågge riktningarna. Hubben/switchen gör korskoppling mellan sändningsparet och mottagningsparet så att de som lyssnar får ramar på avsedda (tre) ledarpar.

För 100BASE-TX används koden 4b5b (fyra bitar kodas som fem) i kombination med non-return-to-zero-inverted (NRZI). (Se någon grundläggande kurs i datakommunikation beträffande koderna.) Med TP-kabel (lägst kategori 5 UTP eller lägre kategori av bra kvalité, s.k. voice grade) som har fyra ledarpar sänds på ett ledarpar och de övriga noderna samt sändaren lyssnar på ett annat ledarpar. Korskopplingen görs i hubben/switchen.

För 100BASE-FX används koden 4b5b i kombination med NRZI. Det är fråga om optisk fiberkabel (multimod). I denna kabel används en fiber för sändning och en annan för mottagning.

Det finns ytterligare en version för 100 Mbps som kallas 100BASE-T2. Tanken var att kunna använda befintliga kabelsystem med kategori 3 UTP och använda endast två ledarpar. Dessvärre kräver denna version dyra DSP-processorer (digital signal processing) i hårdvaran. Eftersom nya kabelinstallationer sällan använder TP-kablar av sämre kvalité än kategori 5 UTP, har marknaden numera inget intresse för denna version av Ethernet.

Gemensam beteckning för 100BASE-T2, 100BASE-T4 och 100BASE-TX är 100BASE-T och gemensam beteckning för 100BASE-TX och 100BASE-FX är 100BASE-X. Vissa nätverkskort för dessa kabeltyper kan växla mellan halv och full duplex. Se kommande förklaring angående detta.

Gigabit Ethernet

Gigabit Ethernet beskrivs av standarden IEEE 802.3z och speciellt för TP-kabel av IEEE 802.3ab. Accessmetoden är modifierad eftersom den förutom förbindelser av typen multidrop (halv duplex med CSMA/CD) också tillåter förbindelser av typen punkt till punkt med full duplex. Punkt till punkt-förbindelse innebär att bussprincipen har förändrats, dvs. kommunikation medges endast mellan två parter och de övriga noderna hör ej denna. CSMA/CD gäller inte på samma sätt för full duplex som för halv duplex. För att detta ska fungera krävs speciella switchar. För halv duplex kan vanliga hubbar och switchar användas. I Gigabit Ethernet används följande parametrar:

Slot time = 4,096 µs
Minsta ramstorlek = 512 b = 64 B
Interframe gap = 96 ns
Maximal bithastighet = 1 Gbps
Kod: 8b10b (åtta bitar kodas med 10 bitar)

1000BASE-T (eller 1000BASE-TX) innebär UTP-kabel av tillräckligt bra kategori (lägst 5). I denna version används fyra par ledare och längden är som vanligt maximalt 100 m. 1000BASE-CX är STP (150 Ohm, max. 25 m) med DB-9-kontakter. I denna version används två par ledare. Det går också utmärkt att använda optiska fiberkablar.

1000BASE-LX är en sådan kabelbeteckning som innebär optisk fiberkabel för **monomod**. L-beteckningen anger **longwave** med våglängden 1.300 nm. Den maximala kabellängden beror av diametern och källorna anger olika värden (3–10 km). Cisco anger maximalt 3 km långa monomodkablar.

Även optisk fiberkabel för **multimod** kan användas. Maximal längd på en sådan kabel beror av diametern och tillverkarna anger olika värden (440–550 m). 1000BASE-SX betyder optisk fiberkabel för multimod och S-beteckningen anger **shortwave** med våglängden 780 nm (eller 850 nm, enligt somliga källor). Cisco anger maximalt 500 m långa multimodkablar med våglängden 780 nm.

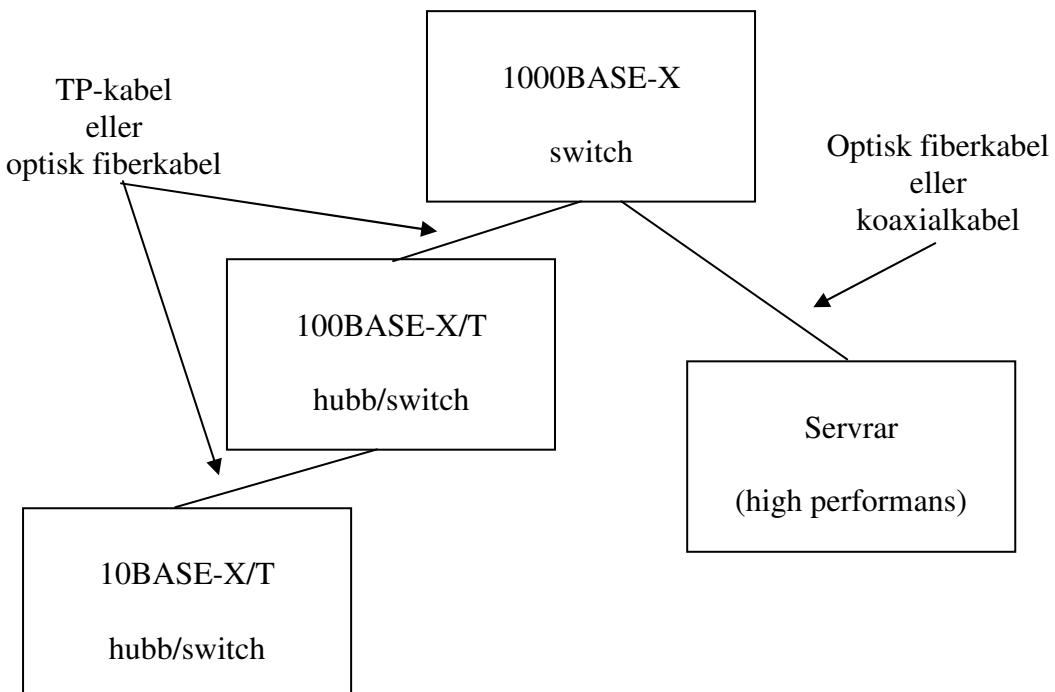
Halv och full duplex

När vi talar om IEEE 802.3 (CSMA/CD) och dess versioner för höga bithastigheter, avses **halv duplex**, dvs. endast en nod åt gången kan sända över stjärnnätet. De övriga noderna lyssnar. Det finns teknik som frångår principerna i CSMA/CD. Denna teknik kallas Ethernet med **full duplex** och betecknas IEEE 802.3x. Populärnamn för tekniken är **EtherChannel**. Full duplex ger sändning i två riktningar samtidigt. För att detta ska fungera krävs att accessmetoden CSMA/CD åsidosätts, dvs. en speciell typ av switch. Dessutom måste switchen kunna buffra data eftersom det kan vara upptaget på önskad utport och att det kan vara fråga om nedväxling till en lägre bithastighet. Ett undantag är Ethernet-versioner som använder ett ledarpar för sändning respektive mottagning. En switch i ett sådant nät kan fungera med cut-through.

Förberedda för full duplex är t.ex. 100BASE-TX, 100BASE-FX, 1000BASE-TX, 1000BASE-LX och 1000BASE-SX. EtherChannel för 100 Mbps och 1 Gbps medger upp till åtta länkar per kanal, enligt Cisco.

En speciell händelse som beskrivs i IEEE 802.3x är att en dator som samtalar kanske inte hinner ta emot fler ramar för studen. Då måste den datorn kunna reglera flödet, s.k. **flödeskontroll**. Detta görs genom att den mättade datorn sänder en **pausram** med begäran om sändningsuppehåll under specificerad tid. Den andra parten inställer vid mottagande av pausramen sin sändning under den önskade tidsperioden. Om den mättade datorn hinner processa data innan paustiden går ut, kan den återstarta den andra partens sändningar genom att skicka över en **startram** (a time-to-wait of zero).

Figur 10 visar ett förslag på hur de snabba Ethernet-versionerna kan tänkas att samarbeta i ett stjärnnät. Arbetsstationerna kan anslutas till någon av hubbarna/switcharna.



Figur 10. Stjärnnät med snabba Ethernet.

10 gigabit Ethernet

Det finns sedan en tid tillbaka en Ethernet-version för 10 Gbps. Denna använder full duplex. Punkt till punkt-förbindelser på wide area networks med ”10 gigabit Ethernet” beskrivs av IEEE 802.3ae. (Observera att IEEE 802.3ae inte uteslutande beskriver just 10 Gbps.) Speciella standarder för kablar till ”10 gigabit Ethernet”: 802.3an (TP-kabel), 802.3ak (twinaxialkabel), 802.3ae (optisk fiberkabel) och 802.3ap (bakplan med metallledare för exempelvis roustrar).

40 och 100 gigabit Ethernet

Den senaste Ethernet-versionen är för hela 100 Gbps. Denna använder full duplex. Punkt till punkt-förbindelser med 40 Gbps och 100 Gbps Ethernet beskrivs av IEEE 802.3ba.

Fältbussar

Inledning

”Fältbussar kallas sådana datanät som är anpassade för att överföra korta meddelanden i snabb takt över relativt begränsade avstånd. Användningen gäller i första hand överföring av binär eller digital information från givare till styrsystem och vidare till styrdon i automatiserade system. Exempelvis kan kommunikationen mellan en robot och ett antal magnetventiler och ändlägesgivare i kringutrustningen till roboten ske med hjälp av en fältbuss. Det förekommer även tillämpningar av fältbussar i processindustriella system, i fordon och i system för lokalövervakning.

Bland fördelarna med fältbussar märks:

- Färre kablar
- Högre driftssäkerhet
- Enklare inkoppling
- Enklare felsökning
- Kommunikation även mot överordnade datasystem
- Kan vara billiga

Ett problem med fältbussar är att industrin hittills inte kunnat ena sig om någon standard. Det beror bland annat på att teknikutvecklingen på området, för närvarande, går snabbt framåt. Det är också betydande skillnader i prestandabehov mellan olika tillämpningar. T.ex. kräver tillämpningar i bilar hög snabbhet och hög tillförlitlighet. I automatiserade maskinsystem krävs särskilt flexibilitet så att man enkelt kan ansluta nya utrustningar. Vid användning för övervakning av lokaler krävs bl.a. möjlighet att ansluta långa kablar, men i gengäld ställs låga krav på hastighet.

Några av de fältbusstyper som konkurrerar är:

- Profibus
- Interbus-S
- CAN
- FIP
- AS-i
- Bitbus”³

Vidare gör vi följande jämförelse: Ett sätt att avkänna givare och kontrollera styrdon är att använda en **Programmable Logic Controller** (PLC). Vanligtvis ansluts varje givare och styrdon till en PLC med var sin kabel. Med en fältbuss kan vi istället ansluta både givare och styrdon med en gemensam kabel, dvs. en buss, till en persondator som kontrollerar dataflödet. Detta ger besparingar och enklare kabeldragningar. Vanligtvis används seriell kommunikation

³ Gilbert Ossbahr, avsnittet *Fältbussar* i kompendiet *CIM - 96, Rapporter från kursen ”Flexibla datorintegrerade system”*, 1996. Se avsnittet **Referenser**.

i nät, så även på fältbussar. Ytterligare en fördel med fältbussar är de inte är begränsade till att enbart ansluta givare och styrdon till styrenheter. Vi kan också koppla ihop t.ex. PLC:er, NC-utrustningar, motorer, övervakningskameror, databaser och olika typer av datorer. Med PC-baserade styrkoncept får vi möjlighet till centraliserad programmering och övervakning även om styrningen på fältnivån kan göras i decentraliseraade enheter som t.ex. PLC:er.

Fältbussarna namnges ofta med respektive akronym. Därför är det på sin plats att förklara några av de förekommande.

Profibus	-	Process Field Bus
CAN	-	Controller Area Network
FIP	-	Factory Information Protocol
AS-i	-	Actuator Sensor Interface

Det finns också bussar som är inriktade på att styra enheter som vanligtvis används i elsystem. Av den anledningen kallas sådana för installationsbussar. Enheter som styrs kan t.ex. vara belysningsarmaturer, persiener, värme-element, larmsystem, luftkonditionering, varmvattenberedare och motorer. Sådana enheter styrs inte endast av strömbrytare utan också av timers, ljussensorer, IR-mottagare, brandvarnare, fönster- och dörrkontakter, rörelse-detektorer, vindmätare, termostater etc. Bussarna kan förses med **gateway** mot omvärlden, t.ex. till ISDN eller bredband.

European Installation Bus (EIB) är exempel på en installationsbuss som använder en tvåledarbuss. Bussen beskrivs i DIN V VDE 0829 och kan installeras med en typ av kabel som består av fyra tvinnade ledare (YCYM) varav endast två används. Busskablarna kan dras parallellt med enheternas starkströmsledningar (AC 230/400 V). Enheterna ansluts till bussen via dataskenor som gör installationen modulär. Kommunikationen över bussen sker med telegram och accessmetoden CSMA/CD. (Accessmetoder beskrivs i avsnittet **Principen master/slave och accessmetoder**.) Ett bussegment får vara maximalt 1000 m långt och ansluta 64 enheter. Det största avståndet mellan två enheter får vara 700 m och avståndet mellan en enhet och spänningssförsörjningen får vara 350 m.

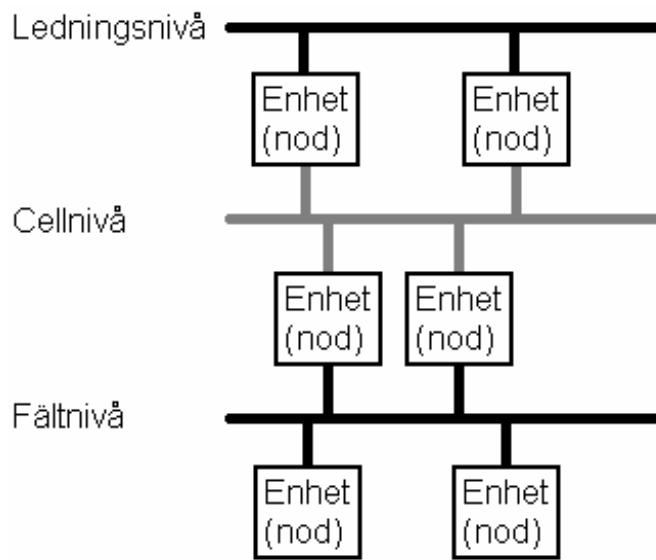
Principen master/slave och accessmetoder

Många av de fältbussar som finns på marknaden bygger på principen **master/slave** som innebär att överordnade enheter, **masters**, kan styra underordnade enheter, **slaves**. Därför sägs masters vara aktiva noder och slaves vara passiva. Enheterna är anslutna till ett gemensamt kabelsystem, en s.k. buss. För att enheterna som vill skicka meddelanden över bussen inte ska krocka med varandra, måste samtliga enheter följa en vedertagen accessmetod. På fältbussar är det masters som har rättigheten att skicka meddelanden till såväl andra masters som slaves. Däremot kan en slave endast svara på en begäran. Därför är det masters skyldighet att sända endast när så är tillåtet på bussen.

Det förekommer olika accessmetoder på fältbussar. En sådan accessmetod (och lokalt nät) kallas **Token bus** som innebär att sändningsrätten i tur och ordning skickas runt till masters. Detta görs med ett meddelande som kallas **token**, dvs. en form av ”stafettpinne”. Den master som för tillfället har fått token har alltså möjlighet att skicka ett meddelande över bussen. Varje enhet på nätet har en unik adress som gör att ett meddelande kan skickas till avsedd mottagare.

En annan förekommande accessmetod är **Carrier Sens Multiple Access with Collision Detection** (CSMA/CD) som är av typen **contention**, dvs. den enhet som är först ut på det gemensamma transmissionsmediet får sända. Alla enheter lyssnar ständigt på trafiken. Den enheten som sänder lyssnar också för att kunna uppfatta kollisioner. Om en kollision uppstår så kommer en av enheterna sända ut en störsignal, s.k. **jam**, som avbryter all kommunikation. Enheterna börjar efter slumpmässigt valda tider att sända på nytt. CSMA/CD är accessmetoden i det lokala nätet **Ethernet**. Detta nät finns i flera versioner men många av dessa bygger på CSMA/CD.

På CAN används en förändrad version av accessmetoden CSMA/CD som har försetts med utökad förmåga att undvika kollisioner och optimalt användande av det gemensamma transmissionsmediet. Denna ombyggnad av CSMA/CD kallas **Non-destructive bitwise arbitration**. Alternativt kallas detta CSMA/CD AMP, där AMP står för **Arbitration on Message Priority**.



Figur 1. Strukturen i ett komplett system.

Det är inte svårt att förstå att PC och PLC är exempel på masters. I allmänhet kallas sådana programmerbara enheter för mastermoduler. Ett PLC-system kan innehålla flera mastermoduler i valfri kombination. Om flera masters är aktiva under drift, kallas detta förhållande **multimaster**. I annat fall är det ett system med **monomaster**.

Typiska slavenheter är I/O-moduler, operatörsterminaler, ventiler, drivutrustningar och transmitters. Observera att ett mindre PLC-system kan utgöra en slavenhet i ett större system. Det totala systemet kan nämligen vara strukturerat med ledningsnivå, cellnivå och fältnivå. Denna struktur visas i figur 1 och används av t.ex. Profibus. Ett PLC-system på cellnivå kan alltså samtidigt vara slav åt enheter på ledningsnivån och master över enheter på fältet.

På ledningsnivån finns olika typer av styr- och övervakningssystem som t.ex. datorer, PLC-system och robotar. På denna nivå är funktionerna inte nödvändigtvis tidskritiska som gör att t.ex. TCP/IP är lämpliga transport- och nätprotokoll. På så sätt kan övervakningsprogram med fördel byggas upp med de användargränsnitt som normalt finns på **Internet** och **intranets**, t.ex. webbläsare. På ledningsnivå kan man med fördel använda **Manufacturing Message Specification** (MMS) som är ett klient/server-system för industriell styrning enligt ISO 9506.

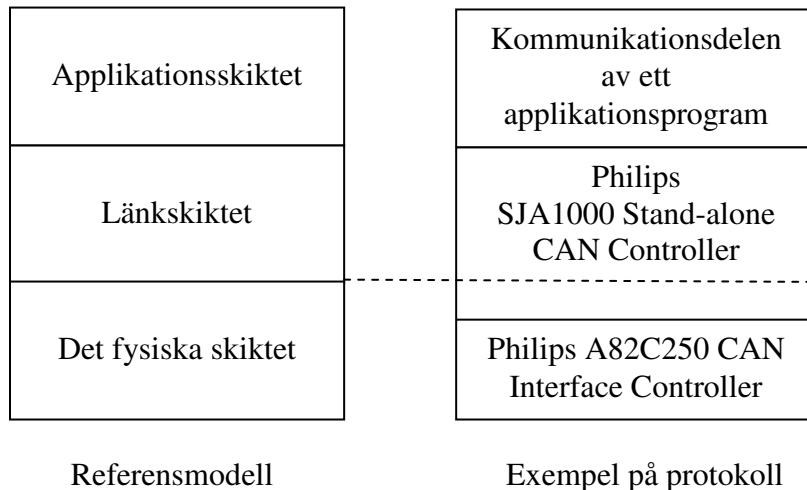
Om applikationerna är tidskritiska är det lämpligt att införa en cellnivå. Det kan t.ex. vara kommunikation mellan styrsystem och distribuerade in- och utgångsmoduler. Eftersom applikationerna kan vara olika tidskritiska så finns det behov av en längsta nivå. Denna nivå kallas fältet. Förutom att applikationerna är mer tidskritiska än på cellnivå så kan det också finnas spänningsförsörjning på bussen. Somliga bussar har utökade säkerhetsfunktioner på fältnivån för att ge egensäkert område, dvs. bussen kan användas i kemisk och petrokemisk industri. Spänningsförsörjning på bussen hör ofta ihop med utökade säkerhetsfunktioner.

Fältbussmodell

International Organization for Standardization (ISO) har tagit fram en referensmodell kallad **Open System Interconnection** (OSI) för datakommunikation i öppna system. OSI är ett ramverk som underlättar framtagande av kommunikationsprotokoll. Denna referensmodell har sju skikt och kan användas för att beskriva noderna på ledningsnivån i figur 1. Antalet skikt som definieras för en nod beror på enhetens funktion. Servrar och arbetsstationer behöver som regel alla sju skikten medan sammankopplande enheter, med undantag av **protocol converters** (som också kallas **gateways**), endast innehåller några av de lägre skikten.⁴

En motsvarande referensmodell för noder på fältbussar kan göras med endast tre skikt: fysiskt, länk och applikation. Dessa tre skikt motsvarar skikt nr. 1, 2 respektive 7 i OSI-modellen, enligt ISO 7498. Referensmodellen för fältbussar innehåller alltså protokoll på cell- och fältnivån. Liksom för system som kan beskrivas med OSI-modellen behöver inte nödvändigtvis alla noder innehålla samtliga skikt. Dessutom kan inte alla fältbussar beskrivas med denna referensmodell.

Protokoll i applikationsskiktet brukar kallas **Higher Layer Protocol** (HLP) och realiseras med program. I figurs 2 ges också ett exempel på protokoll i skikten. Observera att Philips SJA1000 Stand-alone CAN Controller omfattar en del av det fysiska skiktet.



Figur 2. Referensmodell för fältbussar.

Data i den avsändande noden kallas **meddelande** i applikationsskiktet och **ram** i länkskiktet. De enskilda **bitarna** i en sådan ram skickas av det fysiska skiktet ut på det gemensamma transmissionsmediet och tas emot av den mottagande nodens fysiska skikt. Ramen skickas vidare till länkskiktet där den kontrolleras. Meddelandet i ramen överförs sedan till applikationsskiktet.

⁴ OSI-modellen förklaras i avsnittet **Referensmodeller**.

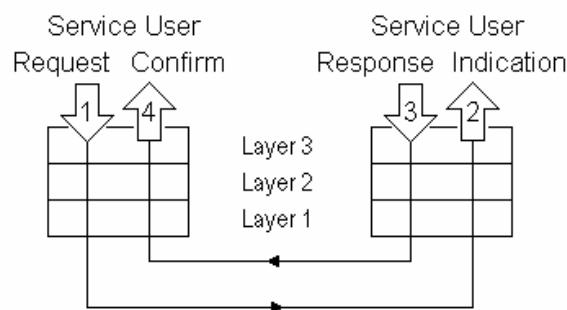
För att begära tjänster används fyra grundläggande primitiver: **request**, **indication**, **response** och **confirm**. Dessa är endast grundläggande. För att göra en komplett begäran om tjänst så anropar ett överliggande skikt dess närmast underliggande med t.ex. **SEND.Request**. Detta kallas för en primitiv. Om vi också har **SEND.Indication**, **SEND.Response** och **SEND.Confirm** så finns det fyra primitiver för tjänsten SEND. En tjänst definieras alltså som en uppsättning av primitiver. På fältbussar har vi behov av tjänster som t.ex. skickar data med kvittens, skickar data utan kvittens och som begär data av annan nod.

När en primitiv anropas anges en s.k. **Service Access Point** (SAP). SAP:er är portar som finns i gränssnitten mellan skikten. Antag att applikationen (**service user**) i en nod vill begära en tjänst. Vilken primitiv som det är fråga om är just nu inte av intresse eftersom vi vill studera hur de grundläggande primitiverna används. Detta inleds med att service user ger request till applikationsskiktet. Se (1) i figur 3. Då sätter applikationsskiktet samman ett meddelande som när det skickas ned till länkskiktet kallas **Application Protocol Data Unit** (APDU). Denna enhet består av dels ett datafält, dvs. ett meddelande, och dels **Protocol Control Information** (PCI) som är specifikt för applikationsskiktet.

I länkskiktet tillkommer ytterligare PCI för kontroll även på denna nivå. Resultatet kallas för en ram. Det fysiska skiktet skickar successivt ut ramens bitar som överförs till den mottagande noden.

På mottagarens länknivå utförs kontroll och bearbetning med hjälp av PCI. Det som återstår efter detta är ju APDU som skickas till applikationsskiktet. Där beaktas PCI för denna nivå. Applikationsskiktet ger indication till nodens service user (2). Den mottagande noden sätter samman ett svar som med response skickas tillbaka till den avsändande noden (3). Förfarandet med PCI upprepas för response på samma sätt som för request. Till sist får vi confirm i den avsändande noden (4).

Detta visar att vi har kommunikation mellan skikten i en nod, mellan noderna och från varje skikt i den ena noden till motsvarande skikt i den andra noden. Det sist nämnda framgår av PCI-hanteringen. För att detaljerat beskriva en fältbuss, använder vi i följande avsnitt CAN som studieobjekt. Därefter studerar vi uppbyggnaden av Profibus fastän ej lika detaljerat som i undersökningen av CAN.



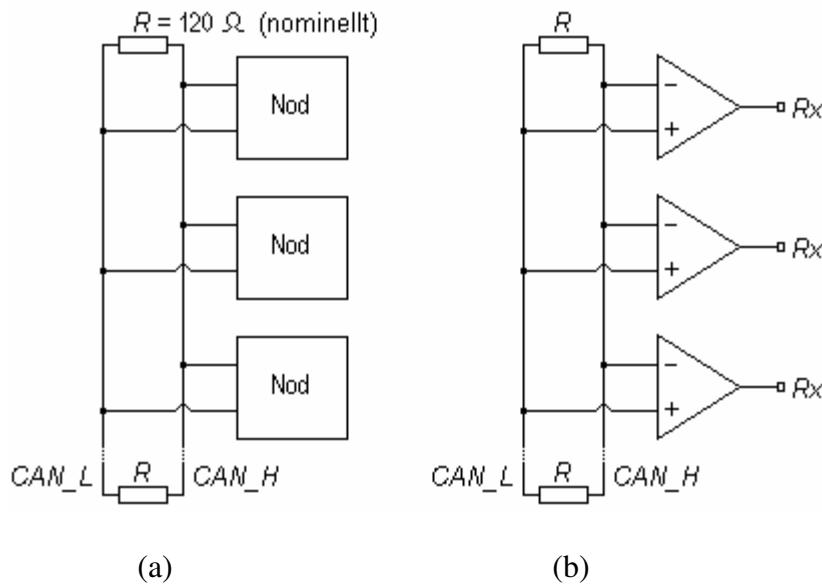
Figur 3. Kommunikation mellan två noder.

CAN

Bosch utvecklade 1986 ett fältbussystem avsett för att minska antalet elkablar i fordon. Tanken var att tunga kabelstammar skulle ersättas med en enkel seriell buss som inte bara skulle hantera långsamma funktioner såsom belysning, fönsterhissar och stolinställningar, utan också realtidsfunktioner som temperatur, oljetryck och motorvarvtal. Fältbussen kom att kallas **Controller Area Network** (CAN) och används numera även för industriellt bruk.

CAN är basen i exempelvis Volcano som är Volvos fältbuss för personbilar och i DeviceNet som ABB använder i robotar. Det ställs nämligen liknande krav i industriell miljö som i fordon. Bussen måste fungera i svår elektrisk miljö, överföra data med hög säkerhet, vara anpassningsbar för olika tillämpningar (s.k. **öppen arkitektur**) och tillåta olika transmissionsmedia.

Dessa goda egenskaper finns på CAN tillsammans med förmågan att upptäcka och hantera fel under kommunikationen, dels på meddelandenivån, dels på bitnivån. Under konstruktionen är CAN-systemet kostnadseffektivt att designa och implementera. Dessutom är ett CAN-system enkelt att konfigurera och modifiera. Central felsökning kan göras både under konstruktion och service.



Figur 4. (a) Principiell CAN-buss.
(b) Förenklade kretsar för mottagning i noderna.

Transmissionmedia kan vara t.ex. **Twisted Pair Cable** (TP-kabel, skärmad som oskärmad), flatkabel och optiska fibrer. Vanliga telefonkablar kan också fungera i miljöer som inte är allt för elektriskt störande. Det pågår arbete med att ta fram standarden SAE J2411 för kablar med enkla ledare, dvs. otvinnad kabel. Kontakter kan t.ex. vara vanlig 9-polig DSUB, 5-polig mini-C (rund) och 6-polig tysk DT04-6P (som används i hydrauliska applikationer).

Det är fråga om en differentiell tvåledarbuss som enligt ISO 11898 ska kunna fungera även om någon av ledarna är av, någon av ledarna är kortsluten till strömförsörjningen eller någon av ledarna är kortsluten till jord. I sådana situationer tillåts sämre **signal-brus-förhållande** (SNR) än i normala fall. Om så skulle vara fallet att bågge ledarna är av, kan konstruktionen av systemet tillåta begränsad funktion i de delsystem som har uppstått p.g.a. avbrottet.

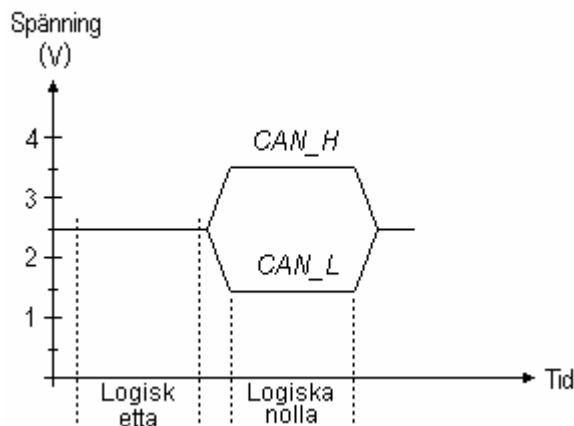
I figur 4 visas att signalen **receive data** (Rx) i en nod sätts samman av signalerna i de bågge ledarna så att

$$Rx = CAN_L - CAN_H.$$

Eftersom signalerna representeras med spänningar (enligt figur 5), låter vi en störning med spänningen u_s adderas till bågge signalerna. Vi får

$$Rx = (CAN_L + u_s) - (CAN_H + u_s) = CAN_L - CAN_H + u_s - u_s = CAN_L - CAN_H,$$

dvs. de differentiella signalerna minskar inverkan av störningar.



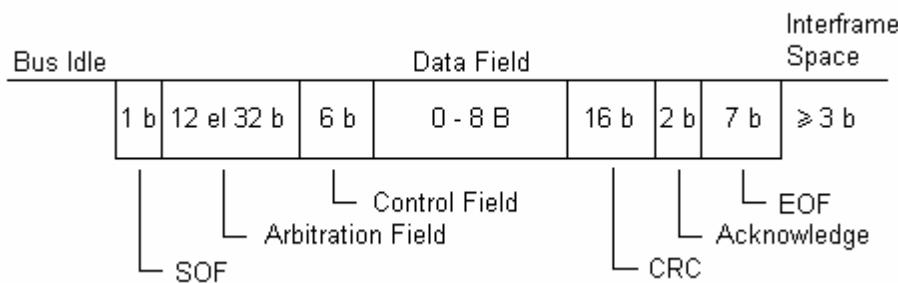
Figur 5. Representation av logisk etta och nolla.
(Typiska värden med terminerad buss.)

ISO 11519 (egentligen 11519-2) ger rekommendationer för CAN-system med låga bithastigheter (5 kbps–125 kbps) och ISO 11898 för höga bithastigheter (upp till 1 Mbps). Den maximala bithastigheten är alltså 1 Mbps. Denna maximala bithastighet gäller om busslängden är upp till 40 m. På längre bussar kan denna bithastighet inte garanteras, t.ex. är den maximala bithastigheten endast 20 kbps när bussen är 1 km.

Enligt ISO 11898 ska bussen termineras med motstånd som har resistansen 120Ω . Detta är ett nominellt värde. Valet av terminering baseras på typ av ledare (resistans per längdenhet och tvärsnittsarea), busslängd och bithastighet. Lämpligt intervall är $[108, 132] \Omega$. Detta är ingen regel. Det förekommer situationer där man rekommenderar resistanser ända upp till 300Ω .

Det finns flera tillverkare av CAN-kretsar, t.ex. Intel, Philips, Motorola, Texas, Siliconix och NEC. Dessa har tagit fram en rad olika halvledarprodukter som t.ex. CAN-kontroller med inbyggd analog/digital-omvandlare (ADC), pulsbreddsmodulator (PWM) och timers. Den öppna arkitekturen gör att det i sista hand är konstruktören som bestämmer hur dessa CAN-kretsar ska kopplas samman till ett fungerande system. Detta har medfört ett brett användningsområde som förutom fordon och robotar är textilmaskiner, navigationsutrustningar, hissar, trafikstyrningssystem, TV-sändare, lyftkranar, jordbruksmaskiner, kopieringsmaskiner, utrustningar för medicinskt bruk m.m.

CAN följer referensmodellen för fältbussar enligt figur 2 och en CAN-ram för meddelande, s.k. **data frame** eller **message frame**, visas i figur 6. Om datafälten är tomta, kan det vara fråga om en **remote frame** som används för att begära data från noder.



Figur 6. Ramformat för överföring av data på CAN.

Dessutom behövs det ramar för att indikera fel, **error frames** och **overload frames**. En error frame är en avbrytande ram (jam) som består av sex nollor (**active error flag**) följt av åtta ettor för avgränsning (**error delimiter**). De sex nollorna hindrar all annan trafik. Logiska nollor är nämligen **dominanta** på bussen som gör att ettor inte kan ändra på spänningen. Logiska ettor sägs vara **recessiva**. Efter error delimiter är det obligatoriskt att vänta under ytterligare tre bittider innan sändningen får återupptas.

Observera att alla noder på en buss inte reagerar på samtliga fel. Eftersom noderna också kan reagera olika snabbt på ett fel, påstås man att det kan bli sex till tolv nollor innan den error delimiter som verkligen kommer att märkas på bussen. Tidigare sådana blir ju ”överskrivna” av nollorna.

En overload frame är en ram för felmeddelande som sänds endast om den mottagande noden inte hinner ta emot en ram. Detta inträffar om den mottagande noden är upptagen av internt arbete. Anledningen till att man gör skillnad på error frame och overload frame är att de sänds vid olika tidpunkter. En overload frame sänds vid behov när den inkommende ramen har kommit fram till åtminstone den sista biten i fältet EOF och senast under **Interframe Space** (IFS). Däremot sänds en error frame i ett så tidigt skede som möjligt.

Ramarna sänds med accessmetoden **non-destructive bitwise arbitration** som är en förbättring av CSMA/CD. Kollisioner på bussen undviks genom att den nod som sänder eller försöker att sända undersöker om ramens bitar ger rätt spänning på bussen, s.k. **bit monitoring**. Detta resulterar i att bitar med motsatt tecken orsakar spänningsfel. Bitarna i en ram skickas ut på bussen med den mest signifikanta biten först och identifieraren kommer

nästan först i en ram. Endast en startbit som är logisk nolla kommer före. Denna identifierare används av mottagarna för att avgöra om de ska använda meddelandet i en ram eller ej.

Om flera ramar sänds samtidigt på bussen, gäller regeln att den ram som har högst prioritet får överföras på bussen. De övriga ramarna måste vänta på sin tur. Ju lägre värde en identifierare har desto högre prioritet får ramen. Dessa identifierare bestäms under konstruktionen av systemet. En identifierare som har lägre värde än en annan innehåller antingen fler nollor i de mest signifikanta bitarna eller så är den första biten som skiljer dem åt en nolla i den lägre identifieraren. Vi har t.ex. $0000_2 < 0001_2 < 0010_2 < 0011_2 < 0100_2$. Om en nod upptäcker att bussen är upptagen av en ram med högre prioritet, ska den genast sluta att sända. Därför överlever endast den ram som har högst prioritet av två eller flera samtidiga.

Eftersom alla noder på bussen tar emot samma ram, är det möjligt att andra noder än den avsedda mottagaren också tar till sig meddelandet. Detta kan utnyttjas för t.ex. övervakning och statistik. Man kan också skicka medvetet till flera mottagare.

Bitarna i en ram skickas ut med koden **Non-return to Zero** (NRZ).⁵ Koden använder två signalnivåer, i detta fallet spänningar, för att representera en logisk nolla respektive en etta. Låg nivå representerar en logisk nolla och hög nivå en logisk etta.

Eftersom en sekvens av samma binära tecken inte ger något signalomslag, kan mottagaren förlora bitsynkroniseringen. Problemet lösas med s.k. **bit stuffing** som innebär att fem lika binära tecken efter varandra alltid följs av en sjätte bit med motsatt binärt tecken. Detta är orsaken till att active error flag kan realiseras med sex nollor utan att sekvenser av bitar i meddelanden kan förväxlas med denna störsignal.

Om ingen kommunikation sker, är bussen i sitt viloläge som är hög nivå. Det första som inträffar när en ram skickas ut är att nivån blir låg. Denna förändring kallas **Start of Frame** (SOF). Aktiv nivå är alltså låg för SOF. Därefter kommer mottagaradressen i urskiljningsfältet (**arbitration field**), kontrollfältet, data, kontrollsumman för data (**Cyclic Redundancy Check - CRC**), kvittensen (**acknowledge**) och slutet på ramen som indikeras med **End of Frame** (EOF).

Ramformatet skiljer sig mellan olika versioner av CAN beträffande antalet bitar i urskiljningsfältet. Detta fält har i versionerna 1.x och 2.0A en 11-bitars identifierare och en bit för **Remote Transmission Request** (RTR). Dessa ramar kallas standardramar. Om RTR är låg är det en data frame. Annars är det en remote frame som har tomt datafält, dvs. en begäran från en master till en slav att den senare ska sända data. Denna princip kallas för polling.

I versionen 2.0B finns ytterligare en identifierare som omfattar 18 bitar så att den totala identifieringen använder 29 bitar. En sådan ram kallas utökad (eng. *extended*).

Vidare finns det i 2.0B en **Identifier Extension** (IDE) som låg betyder att identifieraren använder endast 11 av de 29 bitarna. Detta ger kompatibilitet med 2.0A.

Dessutom har 2.0B en **Substitute Remote Request** (SRR) som har samma position som RTR i 2.0A. Denna bit är alltid hög, vilket gör att en standardram med låg RTR går före en utökad

⁵ NRZ förklaras i avsnittet **Signalbehandling**.

ram som har samma 11-bitars identifierare. Detta utnyttjas när 2.0A-noder används tillsammans med 2.0B-noder på samma buss.

En 2.0B-nod kan vara aktiv eller passiv. En **CAN 2.0B Active Controller** sänder och tar emot utökade ramar. Däremot hanterar en **CAN 2.0B Passive Controller** endast standardramar. Den ignorerar utökade ramar utan att sända ut error frame. En 2.0B-nod undersöker alltså om IDE är hög (utökad ram) eller låg (standardram).

Vi har $2^{11} - 16 = 2032$ st. identifierare för 1.x och 2.0A och $2^{29} - 4.194.304 = 536.676.608$ st. identifierare för 2.0B.⁶ Det maximala antalet identifierare i 2.0B torde räcka tämligen långt.

Kontrollfältet i 1.x börjar med två låga bitar (r_1 och r_0) som från början var reserverade för framtiden. I 2.0A sammafaller r_1 med positionen för IDE i 2.0B. M.a.o. används r_1 för IDE i 2.0A och sätts i detta fall alltid låg. Resten av kontrollfältet (fyra bitar) utgörs av **Data Length Code** (DLC) som anger antalet oktetter i datafältet. (En oktett består av åtta bitar, dvs. en byte – 1 B.)

Version 1.x	Identifier 11 b	RTR 1 b	r_1 1 b	r_0 1 b	DLC 4 b			
Version 2.0A	Identifier 11 b	RTR 1 b	IDE 1 b	r_0 1 b	DLC 4 b			
Version 2.0B	Identifier 11 b	SRR 1 b	IDE 1 b	Identifier 18 b	RTR 1 b	r_1 1 b	r_0 1 b	DLC 4 b

Figur 7. Urskiljnings- och kontrollfält i olika nodversioner.

Datafältet kan innehålla ett meddelande som kan vara upp till 8 B stort. Om datafältet är tomt, kan det vara en begäran om data eller ett svar. Om det är en begäran, anger master i DLC det exakta antalet oktetter som slaven ska sända.

Acknowledge-fältet består av två bitar varav den första kallas **slot** som är hög (recessiv) när ramen skickas ut och som kvitteras efter överföringen, dvs. sätts låg (dominant) av **samtliga** noder som har tagit emot ramen oavsett om de är intresserade av innehållet eller ej. Detta sker omedelbart så att avsändaren genom bit monitoring kan avgöra om ramen kom fram till minst en annan nod eller ej. Huruvida en speciell mottagare har kunnat använda innehållet i ramen indikeras inte av acknowledge-fältet. En mottagare kan till och med vara bortkopplad från bussen utan att slot-biten ger information om detta. Den andra biten i acknowledge-fältet är en avskiljare (**acknowledge delimiter**) som sänds ut hög (recessiv).

⁶ Subtraktionen med 16 för versionerna 1.x och 2.0A kommer av att alla kombinationer av 111111XXXX, där X antingen är 0 eller 1, undviks. Detta ger $2^{11-7} = 16$ som garanterar kompatibilitet mellan olika tillverkares produkter. Enligt Kvaser AB bör dessa kombinationer undvikas även för noder av version 2.0B. I detta fall subtraherar vi med $2^{29-7} = 4.194.304$.

EOF avslutar ramen med sju höga bitar (recessiva). Nästa ram kan sändas tidigast efter ett uppehåll om tre bitar så att databehandlingar, t.ex. beräkningar, ska hinna bli färdiga. Framför allt ska noderna förberedas för kommande flank i SOH. Det nödvändiga uppehållet IFS kallas alternativt **Intermission field** (INT). Om ingen ram sänds direkt efter IFS, återgår systemet till **bus idle**, dvs. bussen går över till viloläget.

Genom att studera ramens utseende har vi sett att det finns felhantering på både meddelandenivån och bitnivån. Eventuella fel i datafältet kan upptäckas med CRC som omfattar 15 bitar, dvs. CRC upptäcker fel på meddelandenivån. Fel som upptäcks på detta sätt kan vara upp till sex slummässiga bitfel eller fel i sammanhangande sekvenser med upp till 15 bitar, s.k. **burst errors**.⁷ Däremot görs ingen felrättning. Den 16:e biten i CRC-fältet är en avskiljare (**CRC delimiter**) som sänds ut hög (recessiv). Den är en av de tre fördefinierade bitfälten som kontrolleras av **frame check**. De övriga två fördefinierade bitfälten är acknowledge delimitrar och EOF.

Frame check räknas också som kontroll på meddelandenivån. Den tredje felhanteringen på meddelandenivån är **acknowledgement error check** som innebär att avsändaren kontrollerar att slot verkligen sätts låg (dominant) av åtminstone en av noderna på bussen.

Felhanteringen på bitnivån utgörs av bit monitoring och bit stuffing. Vid fel utsänds error frames.

På CAN används även mer avancerad felhantering som åtskillnaden mellan temporära och permanenta fel, summering av antalet fel, olika nodtillstånd beroende på antal fel, flödeskontroll m.m. Noderna summerar fel för att kunna sättas i olika tillstånd. I varje nod finns ett register (**Receive Error Counter** – REC) vars värde räknas upp med ett eller åtta vid fel under mottagning och ned med ett när en ram mottas utan fel. Ett annat register (**Transmit Error Counter** – TEC) räknas upp med åtta vid fel under sändning och ned med ett när en ram sänds utan fel. Uppräkning med åtta görs vid sådana fel som anses allvarliga och alltså anses sändningsfel vara ett allvarligt fel.

Om något av de två registren överskrider 127 övergår noden från **error active mode** till **error passive mode**. I detta tillstånd fungerar noden normalt men måste vid fel skicka ut en annorlunda error frame som istället för error active flag har en **error passive flag** som består av sex ettor. På så sätt kommer noden inte att störa den övriga trafiken på bussen med ett ständigt upprepande av error active flags.

Om registervärdena sjunker under 128 återgår noden till error active mode. Skulle det vara så att felet blir bestående och att värdet i TEC till slut överskrider 255, kopplas noden bort från bussen, dvs. noden sätts i **bus off mode**. För att efter detta ta noden idrift krävs att en operatör gör reset av noden, antingen i hårdvaran eller via program.

Vi sammanfattar felhanteringen med följande:

Felhantering på meddelandenivån

- CRC: Kontroll av data med hjälp av checksumman.

⁷ CRC förklaras i avsnittet **Linjeprocedurer**.

- Frame check: Kontroll av bitarna CRC delimiter, acknowledge delimiter och EOF.
- Acknowledgement error check: Kontroll av att slot sätts låg av åtminstone en nod på bussen, dvs. att en ram som sänds har kommit fram till någon av noderna.

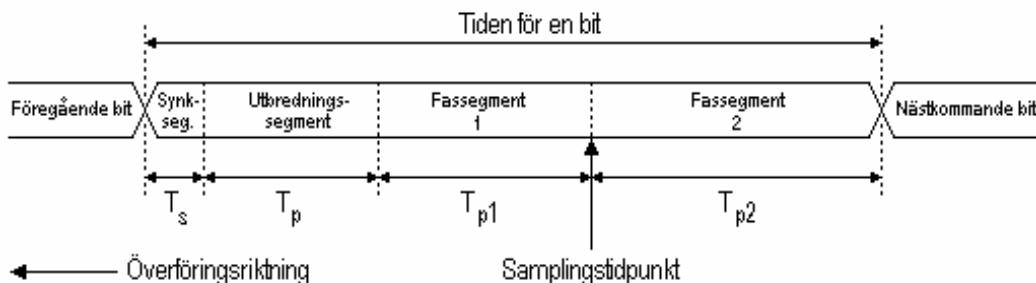
Felhantering på bitnivån

- Bit monitoring: Den nod som sänder kontrollerar att bussen får rätt spänning för varje bit som sänds.
- Bit stuffing: Fem lika binära tecken efter varandra ska alltid följas av en bit av motsatt tecken. Annars tolkas detta som active error flag (sex nollor) eller passive errors flag (sex ettor). Bit stuffing ger också en god grund för bitsynkroniseringen.

Vi har tidigare sagt att den minsta tiden mellan två ramar (IFS) är satt till tre gånger bittiden. Anledningen är att noderna ska hinna behandla data, t.ex. utföra beräkningar och sätta ihop ramar. Om systemets klockperiod vore lika med bittiden, skulle IFS inte räcka till för detta. Dessutom måste samplingar göras i väldefinierade tidpunkter för att systemet ska bli tillförlitligt. Därför består en bittid av flera segment som har till uppgift att definiera en säker samplingstidpunkt.

Det nödvändiga intervallet för en bit visas i figur 8. Varje segment är i sin tur är sammansatt av flera tidsenheter som kallas **time quanta** (T_q). Det är konstruktören som sätter denna tidsenhet till ett bestämt antal heltalsmultiplar, s.k. **clock prescaler value** (p), av klockperioden för nodens interna buss. Klockfrekvensen för nodens interna buss har i sin tur ett bestämt förhållande till oscillatorfrekvensen (f). Bussens klockfrekvens är $r^{-1}f$. Om vi t.ex. har $f = 16 \text{ MHz}$, $r = 2$ och $p = 1$, ger detta att

$$T_q = p/(r^{-1}f) = rp/f = 2/16 \text{ ns} = 125 \text{ ns}$$



Figur 8. Tiden för en bit på bussen.

Eftersom bitarna i en ram kan växla mellan de två logiska tillstånden, noll och ett, så måste vi räkna med att omslagen tar tid. Omslagstiden beror på impedanser som finns i all elektronik och i alla elektriska ledare. För att invänta omslagen används ett synkroniseringssegment som upptar tiden

$$T_s = T_q$$

Det tar också tid att överföra elektriska och optiska signaler. Därför sätts utbredningssegmentet (eng. *propagation segment*) sätts till

$$T_p = \alpha T_q$$

där heltalskonstanten $\alpha \in [1, 8]$. Det är möjligt att med tillval ändra den övre gränsen till ett ännu högre värde. Eftersom varje nod har en egen klocka, kan det mycket väl förekomma både varierande klockfrekvenser (p.g.a. onoggrannhet och drift i kristallerna) och olika faslägen. Detta ger behov av två fassegment. Det första segmentet skyddar mot varierande klockfrekvenser och **positiva** fasfel genom att förlänga bittiden med

$$T_{p1} = \beta T_q$$

där heltalskonstanten $\beta \in [1, 8]$. Konstanten β kan liksom α genom tillval sättas till ett ännu högre värde. Det andra segmentet ger säkerhetsmarginal mot varierande klockfrekvenser och **negativa** fasfel genom att förlänga med en tid som ofta väljs till

$$T_{p2} = \max(2, \beta) T_q$$

För att erhålla korrekt värde på en bit ska samplingen göras i slutet av det första fassegmentet. De fyra segmenten som utgör en bittid ger hög säkerhet förutsatt att noderna är tidssynkroniserade med varandra. Sådan synkronisering, s.k. **hard synchronization**, görs varje gång mottagaren tar emot SOF (**negativ** flank).

Dessutom finjusteras synkroniseringen automatiskt om det visar sig att en flank kommer i fel tidpunkt, dvs. före eller efter synkroniseringssegmentet. Detta kallas **resynchronization** eller alternativt **soft synchronization**. Justeringen γT_q görs endast en gång per bit, där den programmerbara heltalskonstanten γ ofta väljs i intervallet $[1, \min(4, \beta)]$ och kallas **Synchronization Jump Width** (SJW). En förutsättning för resynchronization är att värdet ändras mellan bitar så att en flank markerar början av den andra biten. I annat fall, dvs. om en bit har samma värde som den föregående, kan justering inte utföras.

Om en flank kommer före synkroniseringssegmentet, dvs. i den föregående bitens andra fassegment, förkortas den föregående bitens andra fassegment med **SJW** (hos mottagaren). Om en flank kommer tidigare än tiden **SJW** i den nya bitens första fassegmentet, startas det första fassegmentet om (hos mottagaren). Å andra sidan, om en flank kommer lika eller senare i det första fassegmentet än tiden **SJW**, förlängs den nya bitens första fassegmentet med **SJW** (hos mottagaren).

Resynchronization ser alltså till att samplingen görs vid rätt tillfälle. Skulle det vara så att en flank kommer ännu senare, dvs. efter tidpunkten för sampling, sampelas fel värde. I detta fall är inte finjusteringen tillräcklig och felhanteringen kommer att upptäcka en felaktig ram.

För att göra samplingen ännu säkrare kan man i de flesta noder välja att utföra tre samplingar per bittid, samtliga i slutet av det första fassegmentet.

Tiderna T_s , T_p , T_{p1} och T_{p2} är enligt ISO 11898 programmerbara, dvs. inställningarna kan ändras och sparar i speciella register. Sådana register finns i varje kontroller som hanterar nodernas kommunikation. Vi ställer in $BRP = p$, $SJW = \gamma$, $TSEG1 = \alpha + \beta$, $TSEG2 = \max(2, \beta)$ och för tre samplingar/bit ska $SAM = 1$, annars 0. Observera att beteckningar och förhållande mellan tider kan variera mellan olika produkter.

Vi sammanfattar synkroniseringen med följande:

Synkronisering

- Hard synchronization (ramsynkronisering): Eftersom varje ram inleds med en negativ flank (SOF), vet noderna när varje ram börjar.
- Resynchronization (soft synchronization, bitsynkronisering): Tidpunkten för samplingen finjusteras beroende på när en bit kommer i förhållande till den föregående. För att denna justering ska vara möjlig krävs att värdet växlar mellan två bitar. Bit stuffing ger en god grund för bitsynkroniseringen eftersom denna teknik garanterar högst fem lika bitar efter varandra i data frames.

I en **basic CAN**-nod påminner samspelet mellan kontrollern och processorn om kommunikationen mellan **Universal Asynchronous Receiver Transmitter** (UART) och processorn i en persondator. UART är en del av persondatorns seriella port. Processorn i en **basic CAN**-nod arbetar mycket med att undersöka identifierna i mottagna ramar. Det alltså processorn som undersöker om en ram är adresserad till noden eller ej.

Om identifieringen istället görs av kontrollern, avlastas processorn som kan ägna sig åt beräkningar. Denna typ av acceptansfilter brukar kompletteras med till antalet många mottagarbuffertar. Sådana noder är av typen **full CAN**.

Begreppen **basic CAN** och **full CAN** hänger med sedan de första CAN-systemen. Numera saknar begreppen någon större betydelse eftersom noder av dessa typer är kompatibla (förutsatt kompatibla ramformat enligt 1.x, 2.0A och 2.0B).

Andra gränssnitt är exempelvis **Serial Link Input/Output** (SLIO) som fungerar som en gateway mot omvälden och kan hantera dubbeldirektad kommunikation, PC-card (f.d. PCMCIA) för bärbara datorer och VME. Det finns också analysatorer som kan anslutas direkt till CAN-bussar.

På marknaden finner vi **Higher Layer Protocols** (HLP:er) som t.ex. **CAN Application Layer** (CAL) från den internationella organisationen **CAN in Automation** (CiA), **CANopen** som bygger på CAL och är ett resultat av projektet **ESPIRIT III ASPIC**, **DeviceNet** från Allen-Bradley, **Smart Distributed System** (SDS) från Honeywell och **CAN Kingdom** från det svenska företaget Kvaser AB. CAL har blivit en de facto-standard för HLP.

Profibus

Profibus är en av de ledande öppna fältbussarna i Europa och används över hela världen inom t.ex. tillverkningsindustrin, processindustrin och för värme och ventilation i fastigheter. Många tillverkare av automationsutrustningar har Profibus-gränssnitt på sina produkter. Teknologin utvecklas och administreras av Profibus-användargrupperna. Alla regionala användargrupper är förenade i den sammanhållande organisationen **Profibus International**. Profibus finns i tre versioner som är avsedda för olika ändamål. Vi finner följande versioner:

- Profibus-FMS

Fieldbus Message Specification (FMS) är en universell buss på cellnivå.

- Profibus-DP

Detta är en snabb buss på fältnivå. DP är förkortningen för **Decentral Peripheral**.

- Profibus-PA

Det finns behov av fältbussar som ger egensäkra områden i miljöer där det är hög risk för explosioner som t.ex. inom kemisk och petrokemisk industri. **Process Automation (PA)** är en sådan på fältnivå.

De tre versionerna av Profibus passar väl in i den allmänna strukturen för ett komplett system enligt figur 1. På ledningsnivå finns inte nödvändigtvis tidskritiska funktioner (busscykeltid < 1000 ms) och som transport- och nätprotokoll passar t.ex. TCP/IP. Denna nivå kan omfatta olika styr- och övervakningsfunktioner. Man kan också använda klient/server-systemet MMS.

På cellnivå kan vi använda Profibus-FMS för att hantera tidskritiska funktioner (busscykeltid < 100 ms). Cellnivån kan användas för kommunikation mellan styrsystem och distribuerade in- och utgångsmoduler.

Fältnivån har samma funktioner som cellnivån men med ännu mer tidskritiska funktioner (busscykeltid < 10 ms). Det kan också finnas behov av spänningsförsörjning över bussen på fältnivån. Därför passar Profibus-DP utmärkt för denna. Om man på fältnivå arbetar i miljöer med hög risk för explosioner, bör man istället använda Profibus-PA som har utökade säkerhetsfunktioner.

De olika versionerna av Profibus beskrivs av standarder. Vi har EN 50 170 (del 2) som är en generell beskrivning alla tre versionerna och DIN E 19245 (del 4) som beskriver det speciella IEC-gränssnittet i Profibus-PA. I figur 9 presenteras protokoll och standarder enligt referensmodellen för fältbussar. Standarden EN 50 170 (del 2) beskriver FMS, FDL och det fysiska skiktet.

Kabelsystem med RS-485 kallas **H2** inom Profibus. Det är fråga om skärmad (ibland också oskärmad) TP-kabel med 9-poliga D-sub-kontakter. Varje nod behöver två D-sub-kontakter,

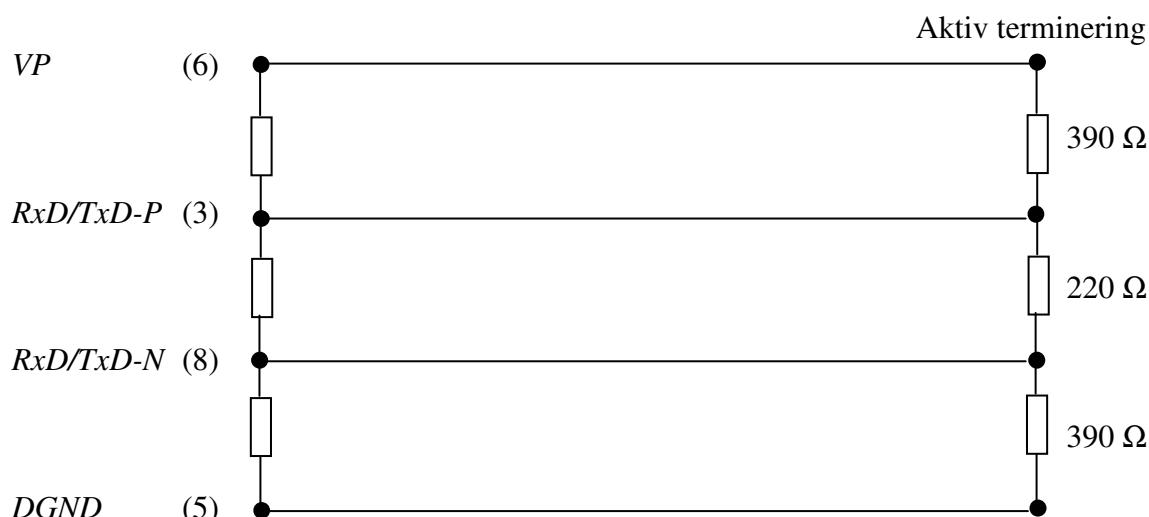
en för signal in och en annan för signal ut. Det går också att sätta in droppkablar mellan bussen och noderna. Det är alltså fråga om en fysisk busstopologi.

	Profibus-FMS	Profibus-DP	Profibus-PA
Applikation (3)	FMS		
Länk (2)	FDL	FDL	FDL och IEC-gränssnitt
Fysiskt (1)	RS-485 och fiberoptik	RS-485 och fiberoptik	IEC 1158-2

Figur 9. Protokollstackar för Profibus-versionerna.
(FDL är förkortningen för **Fieldbus Data Link**.)

Kabelsystemet H2 använder två ledare för datatransport. Dessa kallas *RxD/TxD-P* och *Rxd/TxD-N* eftersom samma ledare användas för både mottagning och sändning. Data-signalerna skickas differentiellt över de två ledarna (*P* och *N*). Detta reducerar inverkan av störningar på samma sätt som för CAN. Logisk nolla och etta representeras med var sin differentiell spänningsnivå.

Dessutom finns det två ledare för spänningsförsörjning på bussen. Dessa ledare kallas *VP* (positiv matningsspänning) och *DGND* (jord). Bussen termineras för att undvika reflexioner p.g.a. höga signalfrekvenser. Termineringarna sitter på bussens bågge ändar och utgörs av en seriekoppling av motstånd ($390 \Omega + 220 \Omega + 390 \Omega$). Eftersom *VP* och *DGND* också kopplas in i termineringen, kallas denna för **aktiv**. Kabelsystemet H2 visas i figur 10.



Figur 10. Kabelsystemet H2.

Bussen kan förses med repeterare (förstärkare, eng. *repeaters*) för att klara av belastningen av flera noder (masters/slaves). Man kan ansluta upp till 32 noder på en buss utan repeterare och med repeterare upp till 127.

Den användbara bithastigheten beror på kabelsystemets längd. Det högsta möjliga bithastigheten som är 12 Mbps och fås på bussar som har upp till 100 m kabellängd. Bithastighetens variation med kabelsystemets längd visas i tabell 1. Om droppkablar används, rekommenderas inte bithastigheter över 500 kbps.

Kabelsystemet längd (m)	Max. användbar bithastighet (kbps)
100	12000
1000	187,5
1200	9,6

Tabell 1. Bithastigheter i kabelsystemet H2.

Alternativet till RS-485 för Profibus-FMS och Profibus-DP är att använda fiberoptik (FO). Detta är ett bra alternativ i miljöer med elektromagnetiska störningar eftersom optiska fibrer är helt okänsliga för sådana. Fibrer av plast rekommenderas för avstånd upp till 50 m och för längre avstånd än så rekommenderas fibrer av glas. Naturligtvis är fibrer av plast billigare än de av glas. Många tillverkare har speciella kontakter med inbyggd konvertering mellan RS-485 och FO. Detta gör att det är lätt att koppla ihop RS-485 med FO.

Profibus-PA använder kabelsystemet som beskrivs av standarden IEC 1158-2 och som i Profibus kallas **H1**. Detta kabelsystem är konstruerat för att ge **egensäkra områden**. Liksom för Profibus-FMS och Profibus-DP byggs kabelsystemet upp med en fysisk busstopologi. Däremot kan man förgrena busstopologin i Profibus-PA till en trädstruktur. De olika delbussarna (grenarna) kopplas ihop med s.k. fältfördelare (repeterare).

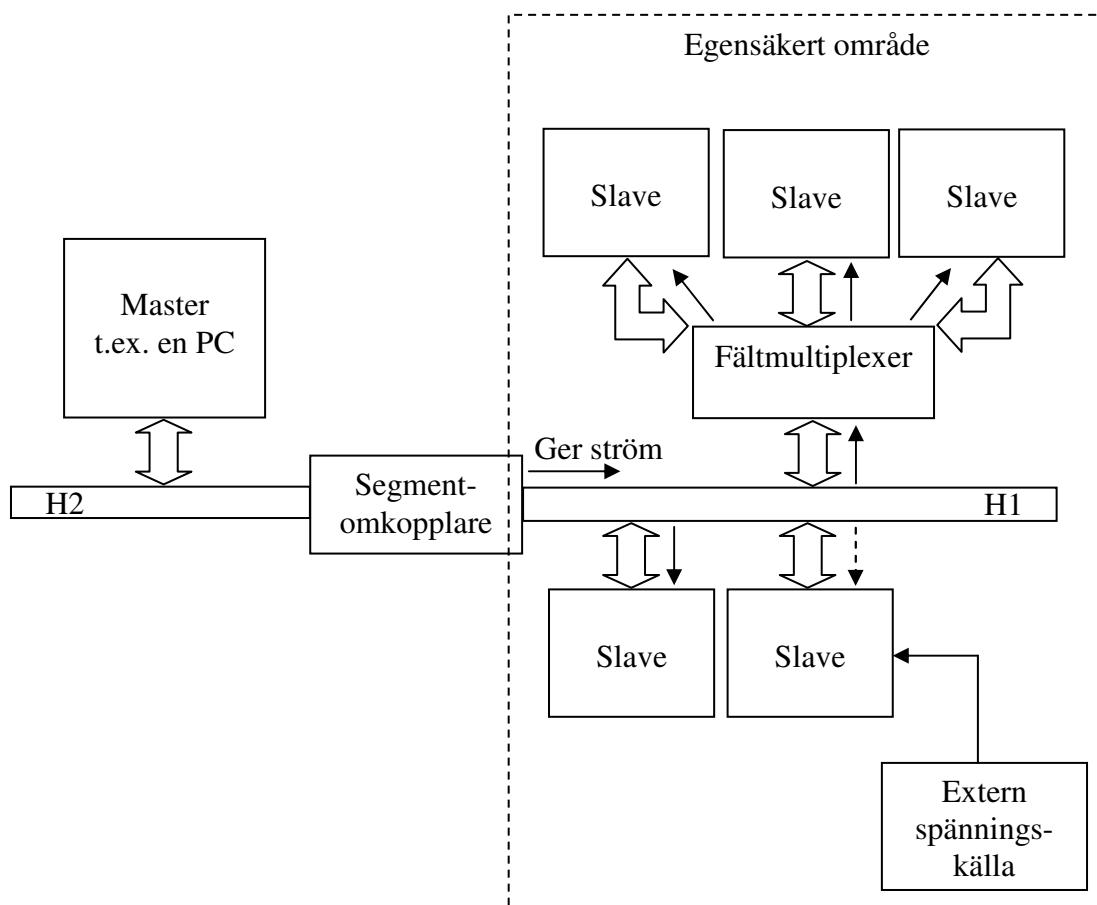
Dessutom kan den fysiska bussen kopplas ihop med stjärnnät. För att erhålla ett stjärnnät kopplas noderna ihop med bussen via en fältmultiplexer. Varje nod har en egen kabel till fältmultiplexern.

Profibus-PA kan anslutas till både Profibus-FMS och Profibus-PA. För en sådan koppling används en segmentomkopplare. I praktiken är det nödvändigt att använda segmentomkopplare eftersom det inte är lämpligt att ansluta annan elektrisk utrustning än egensäker där Profibus-PA används. Detta innebär att persondatorer och PLC:er (som vanligtvis fungerar som masters) ansluts till antingen Profibus-FMS eller Profibus-DP medan slaves sitter på Profibus-PA. (Det finns inget generellt hinder att ansluta en master till Profibus-PA. Naturligtvis måste en sådan enhet uppfylla kravet för egensäkert område på samma sätt som en slave.)

Kabelsystemet (H1) för Profibus-PA beskrivs i figur 11. På H1 används skärmad och oskärmad TP-kabel. Det behövs endast ett par ledare i denna TP-kabel eftersom det inte får finnas separata spänningsförsörjningsledare – endast försörjning via dataledarna är tillåtet.

Noderna på Profibus-PA fungerar som passiva strömförbrukare. Varje nod drar minst 10 mA. Detta är den s.k. basströmmen. Förutom att logisk nolla och detta representeras med strömmar så är dessa modulerade med Manchesterkod. Denna kod är av typen NRZ.⁸ Logisk nolla och detta representeras med variationer från basströmmen som är +/- 9 mA. Detta innebär att en nod alltid drar ström (≥ 1 mA) som i sin tur betyder att ingen energi tillförs bussen från noden varken under sändning eller i viloläget. Detta är grunden för det egensäkra området.

Segmentomkopplaren ska placeras långt från den explosiva miljön, t.ex. i ett operatörsrum, för att inte sänka säkerheten. De noder som så behöver kan få extern försörjning från spänningssällor som uppfyller isolationskravet enligt standarden EN 50020. I annat fall tas ström från busskabeln eftersom det finns en (och endast en) spänningssälla för försörjning per kabelsegment.



Figur 11. Exempel på kabelsystem (H1) för Profibus-PA.

På Profibus-PA kopplas ingen spänningsförsörjning till termineringarna. Därför kallas denna för **passiv**. Den utgörs av en seriekopplad RC-krets med resistansen 100Ω och kapacitansen $1 \mu\text{F}$, en RC-krets vardera på busskabelns bågge ändar.

Det är möjligt att ansluta upp till 32 noder per kabelsegment på H1. Maximalt kan man sätta in fyra fältfördelare, dvs. koppla ihop fem kabelsegment i en trädstruktur. Trots detta är det

⁸ Manchester-kod och NRZ förklaras i avsnittet **Signalbehandling**.

maximala antalet noder begränsat till 126 eftersom sammankopplande enheter (fältfördelare, fältmultiplexerare, segmentomkopplare etc.) och ev. handprogrammerare också drar ström på bussen. Detta medför också att det inte är enkelt att redovisa den maximala kabellängden eftersom den beror på försörjningen (spänningsskällorna), strömbehovet och ledarnas tvär-snittsarea. Ett tummått är maximala kabellängder från 500 m till 1900 m. Då ska man ha klart för sig att bithastigheten alltid är 31,25 kbps.

Linjeproceduren (länkprotokollet) FDL hanterar bussåtkomsten och använder samma princip som i det lokala nätet **Token bus**. FDL kan upptäcka fel på kabelsystemet och mottagande noder. Dessutom hanterar FDL adressering. I denna version av Token bus får endast masters sända respektive efterfråga data. Naturligtvis får också en slave skicka data men endast efter begäran från en master. Likaväl som en master kan kommunicera med en slave, kan masters kommunicera med varandra. Dessutom kan flera masters tala med en och samma slave fastän inte samtidigt.

Om det finns flera masters, skickas en speciell ram som kallas **token** runt mellan dessa. Den som har token äger rätt att sända över bussen. Token skickas runt mellan masters enligt en sändningslista. Detta ger upphov till en logisk ring för token. Observera att ringen endast är logisk eftersom den fysiska topologin är en buss. **Target rotating time** anger den maximala väntetiden innan token ska komma till en master. När token väl har anlånt, får noden behålla den maximalt den tid som **token hold time** anger.

De dataramar som skickas över Profibus kallas telegram. Om en master skickar ett telegram till en slav för att begära data, kallas detta för **request telegram**. Svaret från slaven kallas **response telegram**. Data i telegram förses med paritetsbitar. Dessutom omgärdas data av start- och stopptecken. Säkerhetskodningen av data följer standarden IEC 870-5-1. Därför är Hammingavståndet (eng. *Hamming distance, HD*) lika med fyra i varje telegram. Ju högre HD, desto lättare är det för FDL att upptäcka bitfel. Att $HD = 4$ betyder relativt hög datasäkerhet med tanke på att man använder paritetsbitar.⁹

Förutom att telegrammen kan sändas med metoden **unicasting** (en mottagare), tillämpas också **broadcasting** (alla noder är mottagare till samma telegram) och **multicasting** (en grupp av noder är mottagare till samma telegram). Med telegram kan FDL utföra följande tjänster:

- Sända data och få kvitto
- Sända data utan att få kvitto
- Begära att få data
- Cyklist begära att få data

⁹ Paritetsbit och HD förklaras i avsnittet **Linjeprocedurer**.

Profibus kan konfigureras på två sätt med avseende på antalet masters. Följande konfigurationer är möjliga:

- Monomastersystem

Endast en aktiv master är i drift.

- Multimastersystem

Flera aktiva masters är i drift.

I Profibus-DP klassindelar masters. I den första klassen (DPM1) är masters centrala styr-enheter och i den andra klassen (DPM2) är masters till för programmering, konfigurering och diagnos.

För att konfigurera Profibus används principen **plug-and-play**. För detta ändamål tillhandahålls filer från **Device Database** (DDB), s.k. DDB-filer. Dessa erhålls från respektive tillverkare eller från <http://www.profibus.com>. Eftersom tyskarna tillverkar många tillbehör för Profibus är det av intresse att känna till den tyska benämningen **Gerät Stamm Datei** (GSD).

Kontroll, diagnos, konfigurering och synkronisering är välutvecklade funktioner i Profibus-DP. Masters kan exempelvis aktivera/deaktivera slaves, kontrollera konfigurationen i slaves, tilldela adresser till slaves, synkronisera utgångar i slaves (**sync mode**) och synkronisera ingångar i slaves (**freeze mode**). Masters i klassen DPM1 kan konfigureras över bussen av masters i DPM2.

I noderna kan diagnos av olika typ ställas: stationsrelaterad, modulrelaterad och kanalrelaterad. Stationsrelaterad diagnos är exempelvis mätning av temperatur och spänning. Modulrelaterad diagnos är exempelvis undersökning av en 8-bitars utgång. Kanalrelaterad diagnos är undersökning av enskilda in- och utgångsbitar för att hitta fel som kan uppstå vid exempelvis kortslutning.

Sammanställning av några fältbussar

Filbus är ett mångsidigt system för fjärrstyrning, baserat på intelligent kommunikation mellan databaserna. Ger möjlighet till funktioner såsom pulsräkning, fördröjning innan utförande och sändning/mottagning till/från andra moduler i nätet. Flera värdadaptrar finns tillgängliga. Master/slave-princip.

Bitbus var från början ett tillägg till system avsedda för fjärrstyrning med multibuss-system. Detta bussystem är det mest använda idag. Det tillåter program att bli nedladdade i en avlägsen nod. Master/slave-princip.

FIP ger bestämmande och tillförlitligt sätt för kommunikation innehållande processvariabler och meddelanden. Den sänder ut en identifieringsvariabel till alla noder i nätet. Den nod som har den variabeln får chansen att lägga ut information på nätet. Alla moduler som behöver den utlagda informationen tar upp den samtidigt. Detta ger en decentralisering i databas med variabler i noderna vilket medför realtidskaraktäristika. Det här tar bort företeelsen med nodadresser.

Profibus ger bestämmande kommunikation mellan databas och PLC. Detta sker med 'stafett-pinneprincipen'. Profibus definierar kommunikationsrelationerna multimaster och master/slave med cyklistiskt eller icke-cyklistiskt tillträde. Profibus skiljer på bekräftade och obekräftade tjänster samt tillåter processkommunikation, sändning och multitasking.

CAN-bussar arbetar seriellt och är designade för att ge en effektiv, tillförlitlig och ekonomisk länk mellan givare och utförare. Från början utvecklades CAN för att förenkla kabeldragningar i bilar. Den har spritt sig till andra områden tack vare sina framstående egenskaper.”¹⁰

En i sammanhanget annorlunda teknik är att decentralisera bearbetningen av information såsom i **Lonworks**. I denna fältbuss görs decentralisera bearbetningar i neuronchip till vilka givarna kopplas. Eftersom kontrollalgoritmerna implementeras i neuronchip blir centralenheten avlastad. Kommunikationsprotokollet **Lon talk** ger kontakt mellan olika neuronchip och centralenheten. Detta protokoll kan hantera kollisioner på det gemensamma transmissionsmediet och är specificerat för olika överföringsmedier som t.ex. optiska fibrer och eter (med radio). Det finns också överföringslänkar med infrarött ljus.

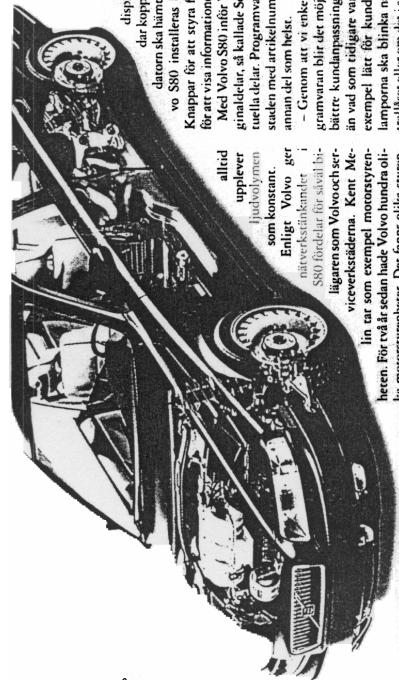
¹⁰ Fredrik Bagge, Niklas Persson och Melker Sundkvist, avsnittet *Jämförelse mellan Fältbussar* i kompendiet *CIM - 96, Rapporter från kursen "Flexibla datorintegreerade system"*, 1996. Se referenserna.

Bilaga till Fältbussar

Nya Volvo S80 har 18 datorer



Här är alla datorer i Volvo S80



display-inställnings-och-släd-

dar kopplas till de sensorer som
dårt ska hämta data ifrån. På en Vol-

vo S80 installeras bara ny programvara.
Knappar för att styra kreditkort och display

för att visa information om hems redan på plats.

Med Volvo S80 införde Volvo en ny typ av
gränssnitt, kallade Software Parts eller virtu-

tuelle delar. Programvaran beställs av verk-

staden med artnumrernummer precis som vilken
annan del som helst.

– Genom att vi enkelt kan uppdatera pro-

gramvaran blir det möjligt att göra berövligat
bättre kundtillsynning av bilens funktioner
i vad som tidigare varit möjligt. Det är till

exempel lart att kunden att bestämma om
ljuserna ska blinka när man använder cen-

tralläsk eller om den ställer sig tur till.
Sensör. Syrsinkelsensor
Sensor. Syrsinkelsensor
som mäter räntens förändring.
Informationen används av
andrasystem, till exempel
för dynamisk stabilitets-
kontroll. För bilen stod,
bronsen bilen sätts lämp-
igt hjul men måste dock
förläts.

Alltig: Volvo ger

universitetskänsla

i S80 fördeleda för skola hi-

ligen som Volvoch-ser-

vicerikstämma. Kent. Me-

lin var exempel motorsyste-

men. För svår sedan hade干
ka motoreverhöreder. Det fans olika styren-

heter beroende på om man nu var fem- eller
sexcylindrig, van der var manuell eller auto-

mattisk växellåda och så vidare. Sed till häl-

van var av dock identiska, det som skilde var
programvaran i dem. Och i synnerheterna var

programvaran hållbarid och kunde alltså in-

te uppdateras.

På fabriksgolvet stod ett stort träd med alla

silika motornsysteriter. För viktiderna var
det annu varje. I princip mäter varje verkstads-

renehmer i Lager. Med Volvo

S80 behöver verkstaden

bära hilla en enda

motorsysten. Den enten

är conn vid leverans men addas med fält

Aven för bilägaren finnsde fördelar med den

hållbarid S80. Skrä en farhällare in-

ställarna i en vanlig bil berydler den i halv dag

och verkstaden för mantering, av vacumpump,

servomotor, vacuumbrake och regula-

re. Skrä en farhällare installeras S80 man-

teras bara en knappslag i rattnen och en ny pro-

gramvara laddas ned till en av bilens datorer.

Ett annat exempel är att installera en farhäll-

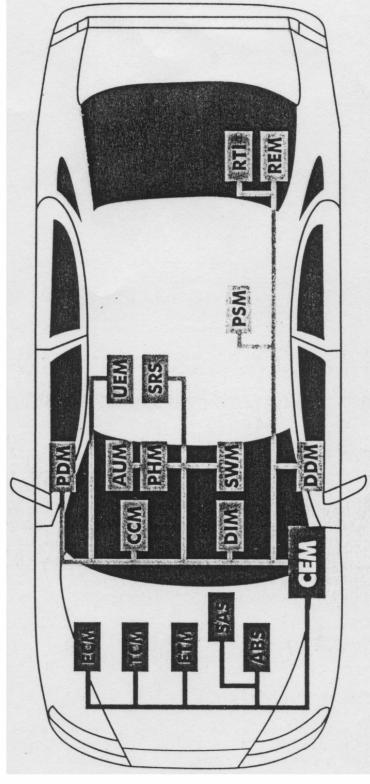
re i bilen måste utspelat styrenhet och

förhäller och då sedan förspelar han

bilens programvara.

"Volvo S80 kan få samma upplägg som dagens sharewareprogram på Internet"

SRS = Airbag	SWW = Steering Wheel
PWM = Power Seat Module.	SMW = Steering Wheel Module. Tid hand om de
PCM = Power Shift Module.	reglage och knoppar som
PSM = Power Shift Module.	styr på rattan.
RDS = Radio Traffic Information Module.	PHM = Phone Module.
RTI = Radio Traffic Information Module.	Den integrerade GSM-telefonen. Det är ett datorsystem som används för information från RDS (Radio Data System) och GPS (Global Positioning System) samt positionering.
TCM = Transmission Control Module.	REM = Rear Electronic Module. Generellt syr enhet för funktioner i den bakre delen av bilen som till exempel belysning och bokulsverme.
ETM = Electronic Throttle Module.	PCM = Central Electronic Module. Generellt syr enhet för funktioner i den framre delen av bilen som till exempel belysning och bokulsverme.
DIM = Driver Information Module.	PSM = Passenger Door Module. Större funktioner i bilen som finns på CD-spelare.
SDM = Side Impact Protection Module.	CCM = Climate Control Module. Större klimatregleringen.
CEM = Central Electronic Module.	DDM = Driver Door Module. Större alla funktioner i dörren, som till exempel lås, larm, bakspegel och fönsterhiss.
ECM = Engine Control Module. Reglerar och styrs motorn. Större delar bland annat insprutning och bensininsprutning.	UHM = Upper Headrest Module. Syr funktioner i taket som akustiko, belysning och larmvarsensor.
SAS = Steering Angle Sensor. Syrsinkelsensor som mäter räntens förändring.	
CCM = Climate Control Module. Större klimatregleringen.	
REM = Rear Electronic Module. Generellt syr enhet för funktioner i den bakre delen av bilen som till exempel belysning och bokulsverme.	



Volvos datorer består av samma delar som din vanliga PC

En dator, Volvo S80 är inte ut som en person-
dator men den består i princip av samma kom-
ponenter, det vill säga en processor och en arbets-
minne. Igelva programvaran lagras i så kallat
flashminne – en minneschip som kommer ihåg
data även när strömmen är avslagen men som
också är omprogrammerbar.

Av de 18 datorerna i bilen har fyra stycken 32-
bitars processorer, ett par har 64-bitars proces-
sorer medan de resterande har 8-bitars proces-
sorer och därmed en långt lägre kapacitet.

Ibland för att skydda komponenterna mot fukt,
smuts och deströj, i Volvo S80 är det inte Win-

dows 98 som gäller, utan ett enkelt, litet opera-
tivsystem som Volvo utvecklat.

Nästvenket ibland är en CAN-nätverk. CAN är en standard för
Controller Area Network och är en standard för
nätverk i bilar som utvecklades av Bosch och Intel i
mitten av 1980-talet. Genom att använda det,
kan Volvo köpa in kompatibla komponenter från
olika leverantörer.