KREATEL
COMMUNICATIONS

Mjukvarumodem på begränsad hårdvara

Examensarbete utfört i datatransmission
vid Tekniska Högskolan i Linköping

av

Marcus Gustafsson

LiTH-ISY-EX-1775

97-05-04

Kreatel Communications AB
Teknikringen 1E
583 31 Linköping

www.kreatel.se
info@kreatel.se

Examinator

Ulf Henriksson
Linköping i maj 1997

Handledare på Kreatel Communications AB

David Löwenbrand
Linköping i maj 1997

Innehåll

1 Introduktion 7

1.1 Bakgrund 7
1.1.1 Samtalsrouter 7
1.1.2 Företaget Kreatel Communications AB 8
1.1.3 Vidare läsning 8
1.2 Läsanvisningar 8

2 Problembeskrivning 9

2.1 Översikt 9
2.2 Syfte 9
2.3 Delproblem 9
2.3.1 Begränsad hårdvara 10
2.3.2 Modulation och demodulation 10
2.3.3 Kommunikationsprotokoll 10

3 Systembeskrivning 11

3.1 Översikt 11
3.2 Samtalsroutern 12
3.2.1 Hårdvara 12
3.2.2 Mjukvara 13
3.3 Administrationssystemet 14

4 Bakomliggande teorier och modeller 17

4.1 Digital kommunikation 17
4.1.1 Kanaler 17
4.1.2 Modulation 18
4.1.3 Demodulation 18
4.2 Kommunikationsprotokoll 19
4.2.1 Några av OSI-modellens lager 20

5 Förtydligande av problem 23

5.1 Begränsad hårdvara 23
5.1.1 Beräkningskapaciteten 23
5.2 Modulation och demodulation 23
5.2.1 Demodulation 24
5.3 Kommunikationsprotokoll 24

6 Implementation av kommunikationsprotokoll 25

6.1 Fysiska lagret ... 25
6.1.1 Kodning av data ... 25
6.1.2 Modulation ... 26
6.1.3 Demodulation ... 27
6.1.4 Synkronisering ... 27
6.2 Datalänklager ... 29
6.2.1 Ramformat ... 29
6.2.2 Datalänkprotokoll ... 30
6.3 Nätverkslager ... 34
6.3.1 Uppkoppling och autenticiering ... 34
6.3.2 Nedkoppling ... 34
6.3.3 Exempel på upp- och nedkoppling ... 35
6.4 Applikationslager ... 35
6.4.1 Kommandouppbyggnad ... 36
6.4.2 Kommandon ... 36
6.4.3 Koppling applikationslager — datalänklager ... 38

7 Resultat 41

7.1 Implementerat system ... 41
7.1.1 Prestanda ... 41
7.2 Förbättringar ... 42
7.2.1 Hårdvara ... 42
7.2.2 Mjukvara ... 43
7.3 Erfarenheter ... 44

8 Referenser 45

8.1 Tryckt material ... 45
8.2 Material i elektronisk form ... 45
8.3 Personlig kommunikation ... 46

1 Introduktion

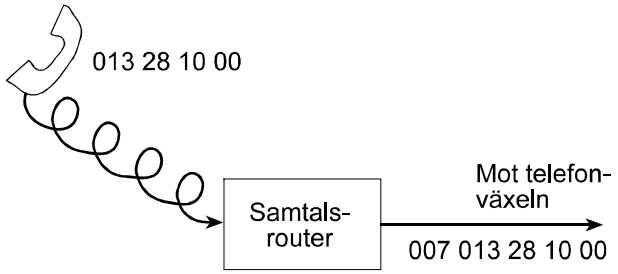
Detta kapitels syfte är att ge läsaren en bakgrund till de problem som detta examensarbete har behandlat.

1.1 Bakgrund

Telemarknaden inom EU ska vara helt avreglerad vid början av 1998, vilket innebär att de traditionella statliga telemonopolen då får utsättas för konkurrens av andra teleoperatörer. Ett av de större problemen för de nya konkurrenterna är att telemonopolen helt dominerar accessnätet, det vill säga de delar av telefonnätet som ligger närmast abonnenten. Att dra egna nya ledningar till abonnenterna är oftast en för dyr lösning för teleoperatören men det förekommer. Lösningen de nya operatörerna är att logiskt knyta abonnenten till sig, även om abonnenten rent fysiskt är kopplad till den traditionella statliga teleoperatören. Det är för denna uppgift som *samtalsroutern* är konstruerad.

1.1.1 Samtalsrouter

Samtalsroutern är ansluten mellan abonnentens telefon och accessnätet. När abonnenten slår ett nummer analyseras det av samtalsroutern. Beroende på hur samtalsroutern är programmerad förändrar den nummersekvensen innan den sänds vidare till den telefonväxel som abonnenten är ansluten till. Detta görs utan att abonnenten märker någon skillnad jämfört med att ringa som tidigare. En konkurrerande teleoperatör kan styra över samtalen till sitt eget telenät genom att placera ut samtalsroutrar hos sina abonnenter.



Figur 1.1 Enklast möjliga exempel på användning av en samtalsrouter. Vid fjärrsamtal görs överkoppling till Tele2:s nät (007) automatiskt utan att abonnenten behöver veta om det.

Samtalsroutern är dock betydligt mer flexibel än så. Den kan t.ex. programmeras till att beroende på pris välja mellan flera olika alternativa operatörer som ska förmedla ett samtal, s.k. *least cost routing*. Den kan också samla statistik från de samtal som ägt rum eller spärra vissa typer av samtal vid angivna tidpunkter. Vilka tjänster som samtalsroutern utför är helt beroende av den mjukvara som ligger i den.

Om samtalsroutern ska klara least cost routing måste det vara möjligt att byta ut prisinformationen i samtalsroutern, eftersom operatörerna då och då ändrar sina taxor. Den enda praktiska metoden för att göra denna uppdatering av pristabeller är att samtalsroutern själv kommunicerar över telenätet med en central uppdateringsserver. För att uppnå ytterligare flexibilitet är det även möjligt att över telenätet byta ut den mjukvara som körs av samtalsroutern. Detta gör att tjänster automatiskt kan läggas till eller uppdateras. Vill man till exempel inte ha least cost routing så är det ändå möjligt att vid ett senare tillfälle köpa till den tjänsten och få samtalsroutern uppdaterad direkt över telenätet.

1.1.2 Företaget Kreatel Communications AB

Kreatel Communications AB är ett litet och ungt företag med säte i Mjärdevi Science Park, Linköping. Företaget har under det senaste året utvecklat ett eget system av samtalsroutrar. Systemet omfattar även en central komponent som tillhandahåller funktioner för systemövervakning och underhåll. Hela systemet går under produktnamnet *Phonix*.

Målmarknaden för systemet är teleoperatörer som efterfrågar ett relativt billigt men ändå mycket avancerat system av samtalsroutrar med tillhörande automatiserad administration som klarar stora volymer av samtalsroutrar.

1.1.3 Vidare läsning

För den som vill veta mer om olika tjänster som en router kan erbjuda eller vill ha en utförlig branschbeskrivning rekommenderas att läsa [Helmers].

1.2 Läsanvisningar

Denna rapport är uppdelad i 8 kapitel. Det första kapitlet ger en bakgrund till existensen av samtalsroutrar. I kapitel 2 finns problembeskrivningen och i kapitel 3 ges en beskrivning av samtalsroutern och det system den verkar i.

I kapitel 4 beskrivs kort digital kommunikation och hur kommunikationsprotokoll är uppbyggda. I kapitel 5 förtydligas och specificeras de uppställda problemen.

Kapitel 6 är rapportens kärna och beskriver och motiverar de designbeslut som har tagits under arbetets gång.

Kapitel 7 beskriver de resultat arbetet har lett fram till samt vilka vidare utvecklingsmöjligheter som finns. Kapitlet summerar även de erfarenheter arbetet har givit.

I kapitel 8 beskrivs det i rapporten refererade materialet.

2 Problembeskrivning

Detta kapitel ägnas helt åt de problem som är examensarbetets syfte att försöka lösa. Först beskrivs övergripande vilket problem som man vill lösa. Därefter beskrivs rapportens syfte i stora drag. Tills sist bryts problemet ned i delproblem.

2.1 Översikt

Problemet som man vill lösa är följande: Att på ett effektivt och säkert sätt från en central server kunna kommunicera med enskilda samtalsroutrar. Detta krävs för att bland annat kunna byta ut prisinformation vid least cost routing, bearbeta data som samtalsroutern har samlat in och för att byta programvaran i samtalsroutern.

Förra generationens samtalsroutrar har ofta ett fast program som man kan påverka genom att förändra programmets indata. Detta innebär att mängden data som behöver överföras vid uppdatering är relativt liten. Men detta begränsar också kraftigt samtalsrouterns flexibilitet. Dessa samtalsroutrar uppdateras på några minuter.

Phonixsystemets samtalsroutrar är däremot väldigt generella på grund av att hela styrprogrammet i samtalsroutern är utbytbart. Detta ökar samtidigt datamängden som behöver överföras. Att överföra de datamängder som Phonixsystemets samtalsroutrar kräver med den teknik förra generationens samtalsroutrar använder sig av skulle kunna ta upp till 20 minuter. Så långa uppdateringstider vilket är helt oacceptabelt om man administrerar ett större antal samtalsroutrar. Det behövs helt enkelt en snabbare och pålitligare metod.

2.2 Syfte

Syftet med detta examensarbete är att utveckla en kommunikationsmetod med hjälp av den hårdvara som redan finns i samtalsroutern. Hårdvaran är förberedd för detta genom att processorn både kan sända och lyssna på telelinjen. Examensarbetet består således av att utnyttja dessa möjligheter och implementera dem i mjukvara för att lösa det angivna problemet.

2.3 Delproblem

Det övergripande problemet kan delas upp i tre delproblem. Dessa beskrivs kortfattat nedan. Problemen förtydligas ytterligare när systemet är beskrivet och nödvändig teori är genomgången.

2.3.1 Begränsad hårdvara

Samtalsroutern Phonix har designats för att vara billig att tillverka men ändå vara flexibel i sin funktionalitet. Hårdvaran är byggd kring en inbäddad processor som klarar av mycket skiftande uppgifter. Hårdvarans begränsning består av processorns låga beräkningskapacitet och dess begränsade minnesutrymme för både variabler och program. Antalet kringkomponenter för filter och liknande är också en begränsning då så mycket som möjligt har optimerats bort för att få ner tillverkningskostnaden.

2.3.2 Modulation och demodulation

Vilken modulationsmetod som ska användas beror på vad för signal man kan generera samt vilka möjligheter man har att demodulera den efter transport över telenätet. Eftersom man har kontroll över hela kommunikationskedjan mellan central server och samtalsrouter är man inte bunden till någon standard men äldre modemstandarder kan fungera som inspirationskälla.

2.3.3 Kommunikationsprotokoll

Kommunikationen måste ske med något protokoll vilket innebär att en analys av dataformat, felkontroll och omsändningsstrategier bör göras. Här begränsar det tillgängliga minnesutrymmet urvalet av dataformat och omsändningsstrategi kraftigt. Detsamma gäller även för felkontrollen som också begränsas av beräkningskapaciteten.

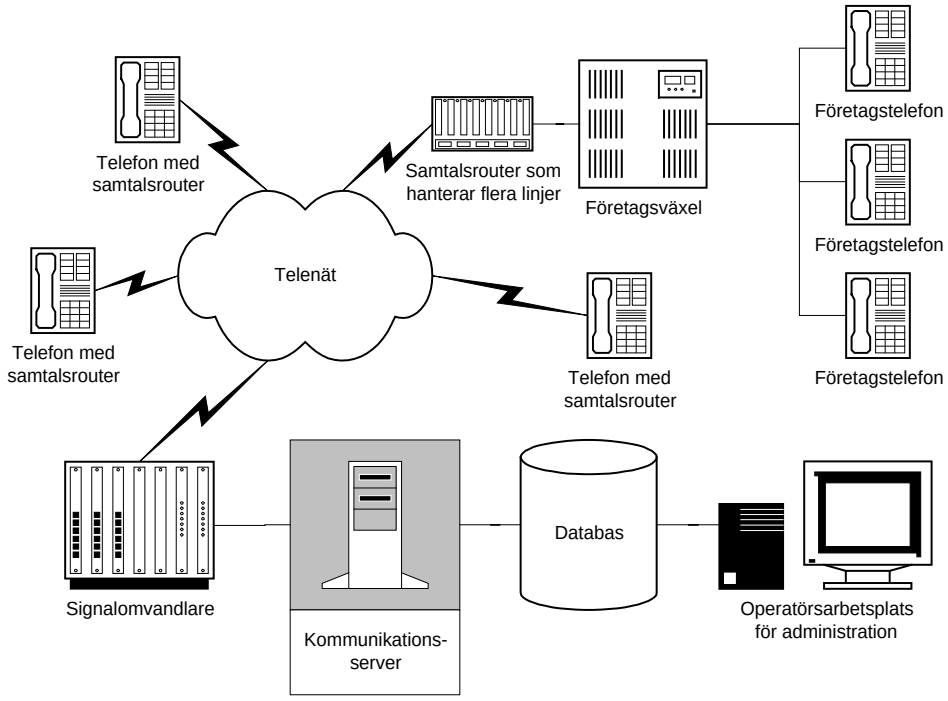
3 Systembeskrivning

Phonixsystemet består av ett centralt administrationssystem och en stor mängd samtalsroutrar sammankopplade via telenätet.

Detta kapitel beskriver först systemet i sin helhet för att därefter beskriva de ingående delarna mer i detalj. Störst vikt läggs på att beskriva samtalsroutern och dess processor eftersom det är den som bestämmer förutsättningarna för problemlösningen.

3.1 Översikt

Systemet är utformat för att det ska vara enkelt att kundanpassa tjänster och kostnadseffektivt att administrera. För att kunna kundanpassa tjänster krävs det att själva samtalsroutern är generell och flexibel. Att använda olika sorters samtalsroutrar för olika typer av tjänster är inte bra eftersom det medför administrativa problem som man vill undvika. För att det ska vara kostnadseffektivt att administrera stora mängder av samtalsroutrar ställs det höga krav på det centrala systemets autonomitet och förmåga att effektivt kommunicera med samtalsroutrar. Man vill helt enkelt att systemet ska sköta sig självt. Det enda manuella inblandningen består av att gentemot en databas ändra den vägvalsinformation som samtalsroutrarna ska använda sig av samt att behandla de felrapporter systemet genererar. Systemet illustreras översiktligt i figur 3.1.



Figur 3.1 *Översikt över hårdvara som ingår i systemet. Samtalsroutrarna som finns distribuerade i telenätet kopplar med vissa mellanrum upp sig mot det centralt placerade administrationssystemet.*

3.2 Samtalsroutern

Samtalsroutern fungerar som ett aktivt filter för den information som passerar mellan abonnentens telefon och telefonbolagets växel. Ett litet exempel visades i figur 1.1.

Samtalsroutern styrs av ett program, skrivet i ett högnivåspråk, som finns lagrat i ett EEPROM. Dessa instruktioner tolkas och utförs av en virtuell maskin i processorn. För att få indata till programmet finns ett antal sensorer som processorns operativsystem behandlar för att sedan generera avbrott och händelser till programmet i den virtuella maskinen.

3.2.1 Hårdvara

Hårdvaran i samtalsroutern består dels av enheter vars uppgift är att förse processorn med information om vad som händer på teleledningen. Andra enheter används av processorn för att styra teleledningen. Exempel på enheter är DTMF-transceiver, ringsignalsdetektor och polaritetssensor.

Processorn

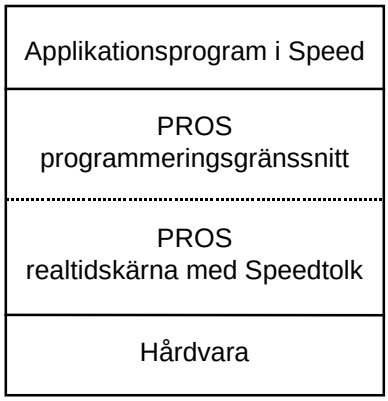
Processorn i samtalsroutern är en PIC16C63 vilket är en 8-bitars inbäddad processor från Microchip Technology Inc. PIC står för *Peripheral Interface Controller* vilket tydligt avspeglar sig på processorns förmåga att styra andra enheter. Den har också inbyggda enheter för att kommunicera via flera olika seriella protokoll (RS-232C, I^2C och SPI). Det finns även en enhet, CCP (*Capture/Compare/Pulse Width Modulation*) som kan detektera och generera pulsbreddsmodulerade signaler.

Processorn har en enkel RISC-kärna med 35 instruktioner och två adresseringsmoder (direkt och indexerad). Alla instruktioner utförs på en instruktionscykel utom hopp

och andra instruktioner som påverkar programpekaren, vilka exekveras på två instruktionscykler. Mer information om processorn och dess inbyggda enheter hittas i [Microchip 95]. Med den klockfrekvens som idag används i samtalsroutern har processorn en kapacitet på 895 000 instruktioner per sekund. Processorns instruktioner är lagrade i ett ROM-minne med plats för 4 096 instruktioner. RAM-minnet består av 192 8-bitars processorregister. Det finns 64 andra register som används för att kontrollera processorns interna timers, in/ut-enheter, styrpinnar samt interna register såsom statusregister och programpekare.

3.2.2 Mjukvara

Mjukvaran i samtalsroutern är uppdelad i fyra lager som visas i figur 3.2. Uppdelningen är nödvändig för att på ett strukturerat sätt kunna hantera det som sker i processorns omgivning.



Figur 3.2 Systemarkitekturen i samtalsroutern Phonix.

Phonix Real-time Operating System

Närmast hårdvaran finns en realtidskärna som tar hand om asynkrona signaler från processorns omgivning. Med hjälp av avancerad signalbehandling förädlas signalerna från hårdvaran till mjukvaruhändelser som applikationsprogrammet kan utnyttja. PROS innehåller också en virtuell maskin. Den virtuella maskinen tolkar och utför byte-kod som lagras i ett externt EEPROM. Samtalsrouterns styrprogram skrivs i ett högnivåspråk som kompileras till den virtuella maskinens byte-kod. Med hjälp av den virtuella maskinen uppnår man stor flexibilitet eftersom applikationsprogrammet blir helt utbytbart samt att man till en viss del blir oberoende av processormodell.

PROS styrs från applikationsprogrammet via ett programmeringsgränssnitt som erbjuder ett brett spektrum av funktioner. Det innehåller bland annat funktioner för att tala om för PROS vilka händelser man är intresserad av. Antag att man exempelvis, inom en viss tid, vill detektera en kopplingston. Då anropar man en funktion som aktiverar kopplingstonsdetektorn. Därefter väntar man på resultatet i form av en händelse, antingen att man har detekterat en kopplingston eller att tiden tog slut (*time-out*). Programmeringsgränssnittet innehåller även funktioner för att enkelt kunna manipulera telelinjen, exempelvis slå siffror med både tonval och pulser.

Programspråket Speed

Speed är ett högnivåspråk i stil med Ada och C som är utvecklat av Kreatel speciellt för att passa avancerade inbäddade system med hårda krav på att hantera många asynkrona händelser med begränsade resurser. Speed skrivs i en C-liknande syntax och kompileras sedan till den byte-kod som körs i den virtuella maskinen.

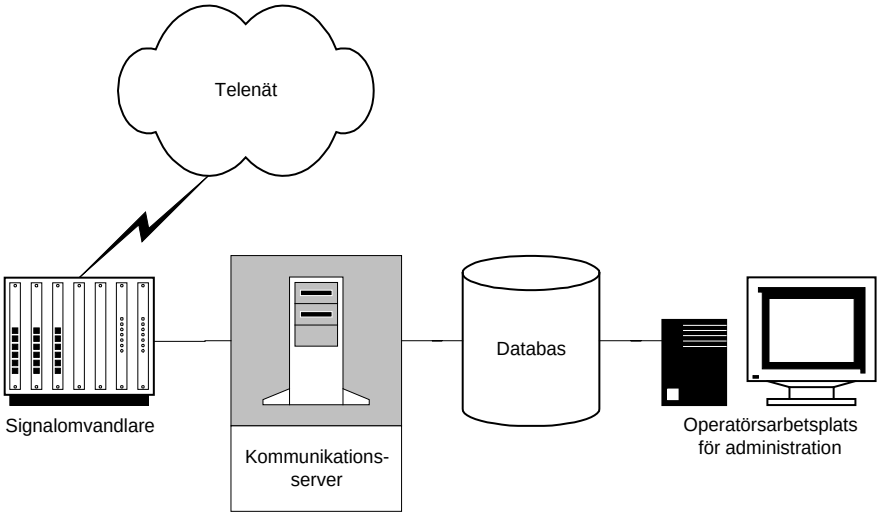
Speed innehåller konstruktioner som vanliga högnivåspråk har, det vill säga. satser för att styra programflödet (*if-else*), iteration (*for*- och *while*-loopar) och funktionsanrop. Speed innehåller även konstruktioner för att enkelt hantera undantag som kan uppstå (*try-finally-catch*). Den mest användbara konstruktionen är *accept-when* som används för att vänta på och hantera flera händelser. I figur 3.3 visas ett exempel på hur Speed ser ut.

```
void DialNumber( constant number_t number )
{
  constant digit_t pos = number.begin( );
  constant digit_t past = number.end( );
  InitTransmitter( );
  accept
  {
    when OnHook
    {
      throw QuitRouting;
    }
    when ReadyToDial
    {
      if pos < past
      {
        DialDigit( pos.value );
        PrintDigit( pos.value );  // debug information to PC
        pos ++;
      }
      else
      {
        break;
      }
    }
  }
}
```

Figur 3.3 *Exempel på programkod skriven i Speed. Funktionen är en del av ett större program och används för att slå ett telefonnummer. Funktionen avbryts när numret är färdigslaget eller när luren läggs på.*

3.3 Administrationssystemet

Administrationssystemet är den del av systemet som hanterar den information som finns i hela systemet. Informationen består av telemäklarens kundregister, prisinformationen över olika taxor och operatörer samt förteckningen av samtalsroutrar och de program samtalsroutrarna innehåller. Informationen lagras i en generell databas som sedan de olika komponenterna i systemet agerar efter eller manipulerar. Hur de olika komponenterna hänger ihop visas i figur 3.4.



Figur 3.4 Ingående komponenter i administrationssystemet.

Administrationssystemet består av fyra olika komponenter. En kommunikationsserver hanterar all datakommunikation med samtalsroutrarna över telefonnätet. Signalomvandlaren består av samma hårdvara som finns i en samtalsrouter men har en helt annan mjukvara. Kommunikationsservern reagerar på de förändringar i databasen som operatören gör från sin arbetsplats.

4 Bakomliggande teorier och modeller

Kommunikation innefattar alltid minst en part som sänder information och minst en part som tar emot information.

När kommunikationen är enkelriktad, det vill säga. att samma part alltid sänder information och en annan part alltid tar emot information, sägs kommunikationen vara *simplex*. Television är ett exempel på simplex kommunikation.

När minst två parter kan sända och ta emot information samtidigt kallas detta *duplex* eller *full duplex*. Man kan helt enkelt prata i mun på varandra och informationen kommer fram ändå. Begreppet *halv duplex* innebär att båda parter kan sända och ta emot information men inte samtidigt. Kommunikationsradio eller ett normalt samtal är ett bra exempel på kommunikation med halv duplex.

Detta kapitels syfte är ge en kort genomgång av hur man överför digital information och beskriva hur man undviker att prata i mun på varandra det vill säga. hur kommunikationsprotokoll fungerar.

4.1 Digital kommunikation

Digital kommunikation innefattar modulation och demodulation av den informationsbärande signalen för att kunna transportera den via en kommunikationskanal.

4.1.1 Kanaler

All kommunikation sker via ett medium – en kanal. Olika kanaler har olika egenskaper med avseende på vilken information den kan överföra och vilka störningar den inför.

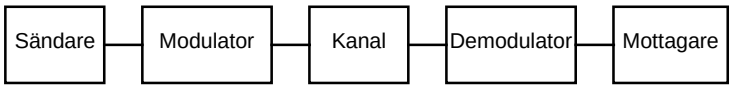
Jämför t.ex. en CD-skiva med din favoritartist, den låter klart utan störningar med att närvara vid konsert med samma artist. Ljudet på konserten är kanske inte lika bra som på CD-skivan men känslan av att vara där i publiken ger en helt annan upplevelse. Detta illustreras av figur 4.1 genom att låta artisten vara sändare och publiken/du är mottagare och kanalen är CD-skiva respektive musikinstrument.

Figur 4.1 Sändare kommunicerar med mottagare via en kanal.

4.1.1.1 Analog telefonikanal

Telefonnätet är utformat för att överföra mänskligt tal med acceptabel kvalité och överför därför endast signaler i frekvensområdet 300 – 3 400 Hz. Detta märks tydligt på att musik över telefon saknar bas och låter lite livlöst.

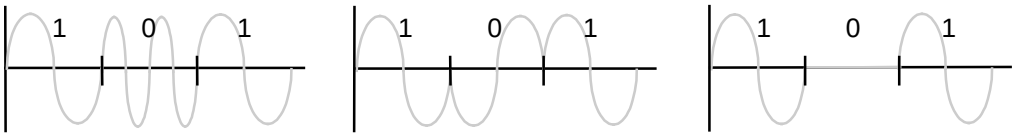
För att överföra digital information över den analoga telefonikanalen så måste den digitala informationen först omvandlas till analog form (moduleras) som kan överföras över telefonikanalen. Mottagaren måste kunna omvandla tillbaka från den analoga formen till den digitala (demodulera). En enhet som både kan modulera och demodulera kallas för modem. Figur 4.2 visar simplex kommunikation över en analog kanal.



Figur 4.2 Sändare kommunicerar med mottagare via en kanal där informationen måste moduleras och demoduleras.

4.1.2 Modulation

Att omvandla en informationsbärande signal till en form som kan överföras på en given kanal kallas modulation. Detta görs genom att dela upp signalen i symboler. Varje symbol översätts till ett förutbestämt mönster enligt den valda modulationsformen. 1:or kan exempelvis översättas till en sinusvåg med frekvensen 1 200 Hz och 0:or till en sinusvåg med frekvensen 2 400 Hz. Detta kallas för frekvensskiftad modulation, på engelska *Frequency Shift Keying* (FSK). Man kan istället förändra amplituden på signalen för att skilja 1:or från 0:or (amplitudmodulation). I moderna modem används amplitudmodulation i kombination med fasskiftad modulation och kallas då för *Quadrature Amplitude Modulation (QAM)*. Fasskiftsmodulation, *Phase Shift Keying* (PSK), skiljer 1:or och 0:or åt genom att sända 1:or med en vågform och 0:or med samma vågform fasskiftad 180 grader. FSK och PSK visas i figur 4.3 tillsammans med enkel form av amplitudmodulation.



Figur 4.3 FSK och PSK samt amplitudmodulation med signal/icke-signal.

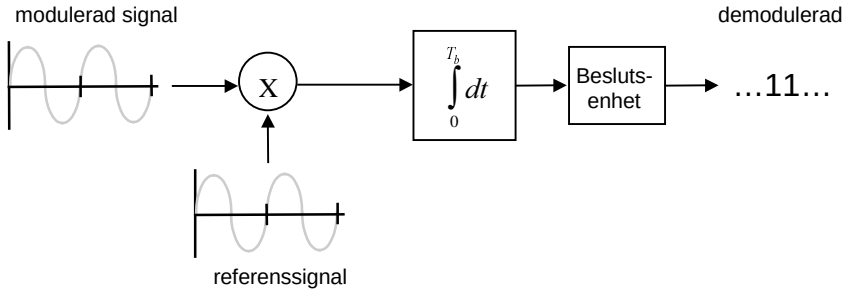
DTMF

Digital Tone Multi Frequency, DTMF eller tonval i vardaglig tal, är en modulationsform för att överföra sifferinformation över telenätet. Tekniken används bland annat i knapptelefoner. DTMF består av 16 olika toner som var för sig moduleras med två olika överlagrade toner. 16 toner innebär att man kan överföra 4 bitar per symbol. För att överföra en ton måste den sändas i cirka 80 ms för att den ska vara möjlig att detektera. Mellan varje ton krävs också en paus på cirka 80 ms, vilket innebär att det tar 160 ms att överföra 4 bitar. Den maximala överföringshastigheten blir då cirka 25 bps (bitar per sekund).

4.1.3 Demodulation

Att demodulera en analog signal oavsett vilken av ovanstående modulationsformer signalen moduleras med kräver ofta multiplikation med en känd signal och

integration följt av ett beslut beroende på resultatet från integrationen [Haykin]. Figur 4.4 illustrerar det hela.



Figur 4.4 En schematisk demodulator.

Riktigt så här enkelt är det inte men det ger en bra bild av de olika steg som demodulation innebär.

4.1.3.1 Synkronisering

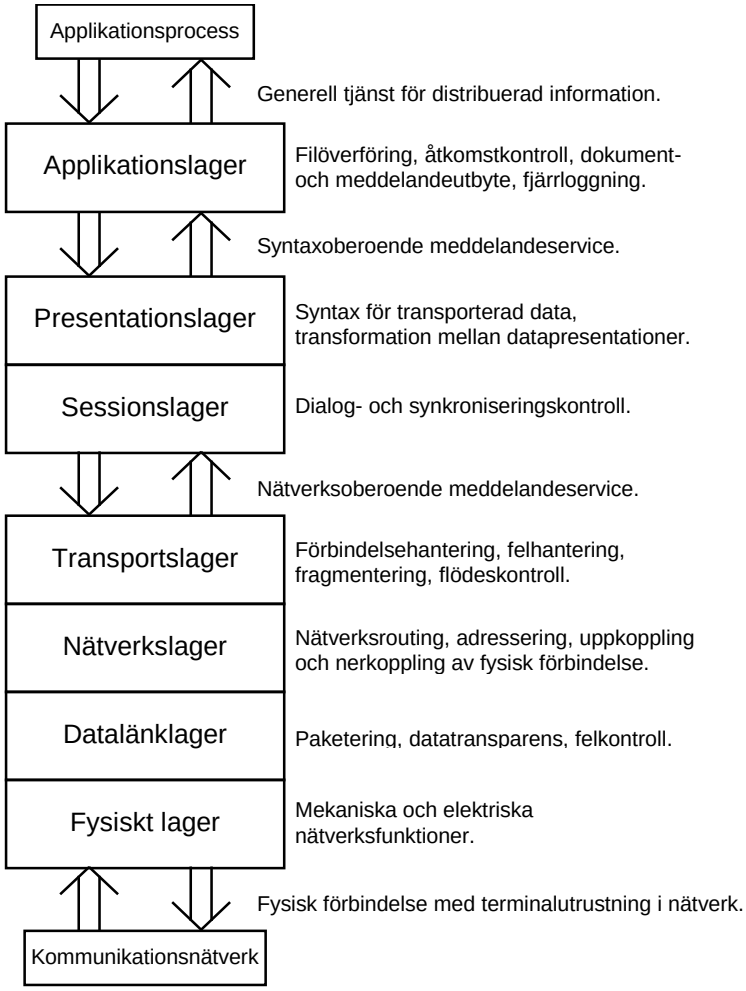
En svårighet för mottagaren är att veta när en modulationssymbol tar slut och en ny börjar. Detta kräver att sändaren även sänder denna information, antingen skilt från signalen eller inbakat i själva modulationsformen.

Denna synkronisering kan ske på flera olika sätt. Enklast att demodulera är att mottagaren helt enkelt vet tidpunkterna för symbolövergångarna genom sändaren att överför denna information på en separat kanal. Att mottagaren har tillgång till denna information kallas för koherent modulation. Att demodulera signalen utan tidsinformationen kallas för icke-koherent demodulation. Detta kräver större funktionalitet av demodulatorn.

4.2 Kommunikationsprotokoll

Det räcker inte med att kunna skicka och ta emot data, det vill säga ettor och nollor, över en kanal. Man behöver ett protokoll som talar om hur man paketerar data och förädlar den till information. Ett protokoll skall dessutom innehålla regler för hur förbindelser upprättas och vilken ände som ska sända och vilken ände som ska ta emot data. Felhantering är också en viktig uppgift för ett protokoll. Att data försvinner eller kommer fram felaktiga eller i fel ordning ska också kunna hanteras.

För att få struktur på ett kommunikationsprotokoll delas de vanligen in i ett antal på varandra liggande lager. Varje lager tillhandahåller en eller flera tjänster åt högre lager och för att kunna utföra sina uppgifter utnyttjar de tjänster i lägre lager. Denna modell är mycket vanlig inom datakommunikationsområdet bland annat för att dess uppbyggnad är lämpad för implementation i programvara. En vanlig modell är *The Reference Model for Open Systems Interconnection* (OSI) framtagen av *International Standards Organization* (ISO) [Hallsall].



Figur 4.5 Sammanfattning av de sju nivåerna i OSI-modellen.

OSI-modellen som sammanfattas i figur 4.5 har ofta kritiserats för att vara för generell, stor och klumpig [Tanenbaum]. Trots kritiken är grundtanken med den strikta lagerindelningen och väl definierade uppgifter mycket användbar.

4.2.1 Några av OSI-modellens lager

Nedan beskrivs de lager i OSI-modellen vars uppgifter är nödvändiga för att lyckas med implementationen av ett mjukvarumodem.

Fysiska lagret

Det fysiska lagret bestämmer vilken modulationsform som ska användas, vilka signalnivåer som ska användas, hastigheter med mera. Det fysiska lagret erbjuder till överliggande lager är tjänsten att sända och ta emot 1:or och 0:or.

Datalänklagret

Datalänklagrets uppgift är att förmedla en ändlig sekvens godtyckliga bitar till en annan nod i nätverket. Länklagret ska också garantera att mottagen bitsekvens är fri från fel.

Nätverkslagret

Nätverkslagret ansvarar för att data sänds till rätt mottagare. Nätverkslagret hanterar även upp och nedkoppling med andra noder i nätverket. För telefonikanalen är det frågan om att slå ett telefonnummer och lyfta på luren när det ringer.

Applikationslager

Applikationslagret använder sig av underliggande lagers tjänster för felfri dataöverföring för att utföra de uppgifter protokollet är avsett att utföra.

5 Förtydligande av problem

Nu när systemet är beskrivet och teorin genomgången är det dags att förtydliga problemen och ange vilka konsekvenser de har för den valda lösningsvägen.

5.1 Begränsad hårdvara

Processorns låga beräkningskapacitet och bristen på RAM-minne är det problem som hela tiden ligger till grund för valda lösningsvägar. Storleken på programminnet som är avsett för modemet uppgår till ungefär 1 000 instruktioner, vilket medför att implementationen bör ske i assembler för att ha fullständig kontroll över mängden instruktionsutrymme som åtgår.

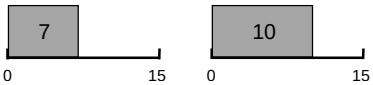
5.1.1 Beräkningskapaciteten

Antag att man vill ta emot data med en bithastighet av 1 200 bitar per sekund (bps). Då har man 895 000/1 200 ≈ 746 instruktioner per bit tillgodo. Detta kan tyckas vara fullt tillräckligt men tänk då på man bör ta fler än ett sampel av signalen innan man avkodar den. Antag att man gör 16 sampel per bit, vilket ger 746/16 ≈ 47 instruktioner per sampel för att sampla, lagra undan och avkoda data. Till detta kommer instruktioner för att lagra undan varje avkodad bit och byte. Att sända, det vill säga modulera, en bit med 746 instruktioner borde vara möjligt förutsatt man väljer en enkel modulationsform. Med dessa siffror i bakhuvudet så inser man att sända och ta emot data samtidigt, det vill säga att full duplex är uteslutet.

5.2 Modulation och demodulation

Äldre modemstandarder använder sig ofta av modulationsmetoderna frekvensskift (FSK, *Frequency Shift Keying*) eller fasskift (PSK, *Phase Shift Keying*) [ITU]. Problemet övergår till hur man genererar en frekvens eller fasskiftad signal med hjälp av processorns inbyggda pulsbreddsmodulator (PWM).

Processorns pulsbreddsmodulator fungerar genom att man anger en periodtid och en pulsbredd. I figur 5.1 visas två exempel på olika pulser man kan genereras.



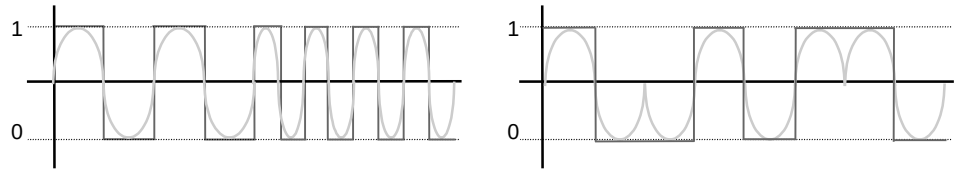
Figur 5.1 Två pulser, med en pulsbredd på 7 respektive 10, båda med periodtiden 15.

Periodtiden och pulsbredden kan man variera relativt fritt. Problemet med dessa pulser är de innehåller mycket övertoner. Enligt telestandarder så får man endast sända ut signaler inom ett visst frekvensband för att inte kollidera med televäxlarnas

styrsignaler. Ett lågpassfilter efter pulsbreddsmodulatorn dämpar övertonerna till en tillåten nivå.

5.2.1 Demodulation

Att demodulera en signal oavsett vilken av ovanstående modulationsformer signalen moduleras med kräver oftast multiplikation med en känd signal och integration av resultatet. Detta gäller för analoga detektorer. Integration är enkelt att implementera, man summerar sampeln helt enkelt. Multiplikation har processorn ingen instruktion för i hårdvara, men detta blir inte heller ett problem utan det överskuggas av ett annat större problem. Insignalen till processorn från telelinjen är nämligen helt digital med endast en bits upplösning. Hur detta påverkar en frekvensskiftad signal och en fasskiftad signal visas i figur 5.2.



Figur 5.2 Signal på telelinjen och insignal till processorn. Frekvensskiftad signal till vänster och fasskiftad signal till höger.

Att själva demodulationsalgoritmen bör innefatta sampling av signalen vid flera tillfällen per bit är inte svårt att inse. Problemet är hur man ska behandla samplen. För en frekvensskiftad signal kan man tänka sig att man räknar antalet övergångar från 0 till 1 och 1 till 0 för att avgöra vilken frekvens man tar emot. För en fasskiftad signal är jämförelse med en referenssignal antagligen en god idé.

5.3 Kommunikationsprotokoll

Att implementera ett generellt och strikt lagerindelat kommunikationsprotkoll tar för mycket programutrymme vilket gör att man noga bör tänka över vilka lager som ska ingå och hur den önskade funktionaliteten ska implementeras. För att inte använda för mycket programutrymme får man antagligen släppa lite på lagerindelningen till bekostnad på abstraktionen. Processorns har begränsningen att den endast kan lagra 8 återhoppsadresser vid nästlade anrop till underprogram. Detta kan vara ett problem då man gärna vill kapsla in funktionalitet i underprogram för att på så sätt uppnå en hög abstraktionsnivå i programkoden.

6 Implementation av kommunikationsprotokoll

Detta kapitel beskriver hur problemlösningen har gått till, vilka problem som har uppstått på vägen och hur de påverkat implementationen.

Att implementera det nedersta lagret först och sedan gå uppåt lager för lager är en ordning som faller sig naturligt eftersom de största begränsningarna är i hårdvaran.

Först börjar man med funktioner för att sända och ta emot enskilda bitar, sedan höjer man ambitionsnivån genom att kunna sända och ta emot en sekvens av bitar. Nästa steg är att kunna få fram en felfri bitsekvens, det vill säga ett fungerande datalänklager. När man väl har kommit så långt har man stora friheter att utforma nätverkslager och applikationslager.

6.1 Fysiska lagret

För att lära sig hur processorn fungerar var det lämpligt att börja med att lära sig att programmera dess pulsbreddsmodulator och på så sätt utreda vilka modulationsformer som är möjliga att generera. Det visade sig att en sinusformad signal med en fix frekvens var möjlig att generera med en acceptabel nivå på övertonerna. Detta styr in valet av modulationsform till fasskiftsmodulation.

6.1.1 Kodning av data

Koherent binär PSK kodar data genom att låta en 1:a representeras av en vågform och en 0:a representeras av samma vågform men 180 grader fasskiftad. Differentiell PSK fungerar inte riktigt på det sättet. DPSK kodar informationen i skillnaden mellan två på varandra efterföljande bitar. En 1:a representeras genom att sända en signal med samma fas som föregående bit, en 0:a representeras genom att fasskifta signalen 180 grader gentemot signalen som den föregående biten sändes med.

Antag att man vill skicka en byte med värdet 201, binärt med den minst signifikanta biten först blir det 10010011. Med koherent PSK sänds vågformen med följande fasvärden $0\ \pi\ \pi\ 0\ \pi\ \pi\ 0\ 0$.

För DSPK måste man anta att man redan har sänt en bit, till exempel en 1:a. Det är viktigt att sändare och mottagare antar samma symbol. Den sända vågformens fasvärden blir $(0)\ 0\ \pi\ 0\ 0\ \pi\ 0\ 0\ 0$.

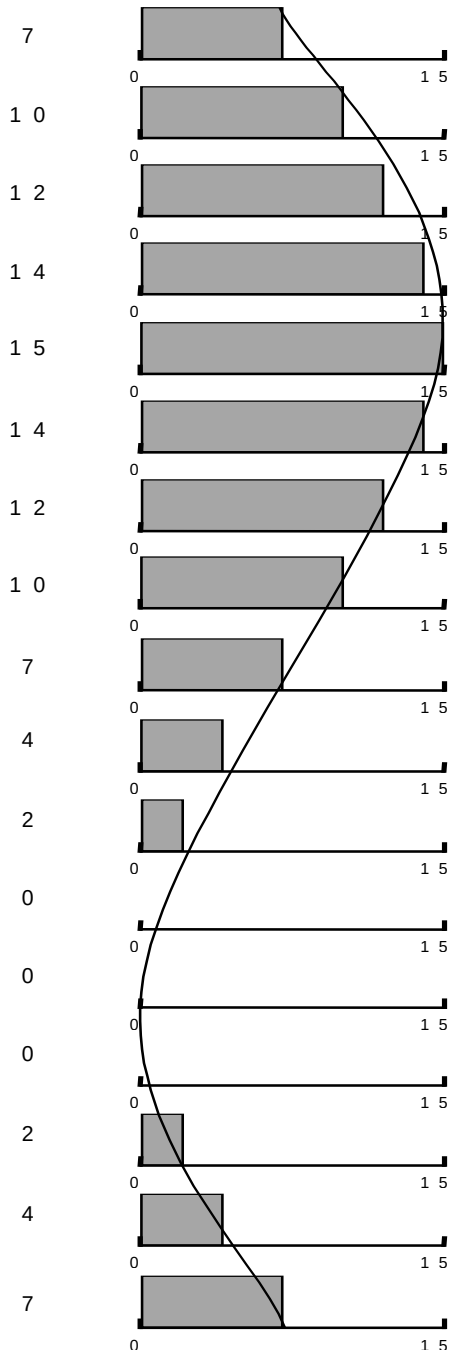
Telefonikanalen har en egenhet som består i att man inte vet vilken polaritet en uppkopplad förbindelse har. Polvändningen i telefonikanalen beror på flera olika saker, bland annat på hur telejacken är installerade gentemot den lokala televäxeln. Detta innebär att en signal kan fasvridas 180 grader utan att man vet om det, vilket är fatalt för vanlig PSK. Differentiell PSK har inga absoluta referenser utan använder sig

av skillnaden mellan signalerna, vilket gör DPSK helt okänslig för polvändning av kanalen.

6.1.2 Modulation

Den digitala informationen moduleras med en bit per period i en sinusformad signal med frekvensen 1 750 Hz. Detta ger en teoretisk överföringshastighet på 1 750 bps.

Varje sinusperiod består av 32 stycken lågpassfiltrerade pulser genererade med PWM-enheten, alla pulser har periodtiden 15, vilket motsvarar 17,85 µs. Varannan puls ändras pulsbredden enligt de värden som återfinns i figur 6.1.

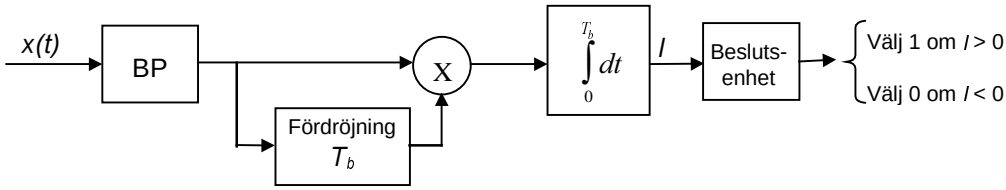


Figur 6.1 PWM-pulser som lågpassfiltreras och blir en sinusvåg med frekvensen 1 750 Hz. Till vänster står de pulsbredder som genererar sinusvågen.

En signal med högre frekvens och samma modulationsmetod som tidigare ger högre överföringshastighet. För att generera en signal med högre frekvens måste periodtiderna på pulserna göras kortare vilket minskar upplösningen på pulsbreddena. Detta leder till att signalens övertoner blir kraftigare vilket kräver ett dyrare lågpassfilter. Den valda frekvensen är således en kompromiss mellan hastighet och övertoner. Frekvensen 1 750 Hz är också anpassad för att hamna någorlunda mitt i telefonikanalens frekvensområde (300–3 400 Hz).

6.1.3 Demodulation

Hur demodulation av en DPSK-signal går till med en analog mottagare visas i figur 6.2.



Figur 6.2 Blockschema för en DSPK-mottagare.

Vid mottagaringången finns ett bandpassfilter centrerat kring bärvågsfrekvensen för att få bort så mycket störningar som möjligt. Efter filterutgången multipliceras signalen med en fördröjd version av sig själv där fördröjningen är en bittid T_b. Resultatet integreras och skickas in i en beslutsenhet som ger en 1:a vid positivt invärde och en 0:a vid negativt invärde.

6.1.3.1 Phonix digitala fasskiftdemodulator

I Phonix finns det inte något bandpassfilter på insignalen. Signalen passerar först en kvantiserare vilket ger en digital insignal. Den digitala signalen är kopplat direkt till processorn vilket innebär att fördröjning, multiplikation och integrering samt beslutsfattande måste göras i mjukvara.

Kort sagt så samplas signalen ett flertal gånger och för varje sampel uppdateras en räknare som används för att avgöra om signalen var fasskiftad eller inte. Räknaren är noll då en period är fasskiftad gentemot den förgående perioden och har sitt maxvärde då perioderna har samma utseende. Exakt hur detta går till beskriv i [Gustafsson].

6.1.4 Synkronisering

Synkronisering mellan sändare och mottagare behövs på grund av att ingen part kan både sända data och lyssna efter data samtidigt. Lösningen är att dela upp tiden mellan att sända data och lyssna efter data. Hur sändare och mottagare kommer överens om när de ska sända respektive lyssna efter data bestäms av datalänklagret.

I det fysiska lagret övergår synkroniseringsproblemet till två olika delproblem som uppstår då man lyssnar efter data. Det första problemet är att kunna skilja signalen från brus, det vill säga att kunna avgöra om det sänds data eller inte. Det andra problemet är symbolsynkroniseringen (ställa symbolklockan), det vill säga var i signalen de enskilda symbolerna finns.

6.1.4.1 Första försöket

Den första metoden som provades byggde på att sändaren först sände ut en signal, med en form som inte kan uppstå i den vanliga signalen, innan den sände ut data.

Synkroniseringssignalen bestod av en sinusvåg med frekvensen 1 167 Hz med varannan period fasskiftad 180 grader. I mottagaren ger en sådan signal upphov till långa sekvenser med 1:or och långa sekvenser med 0:or, bortsett från störningar.

Att detektera en känd signal gör man enligt teorin med ett filter h(t) som har utseendet h(t)=x(T-t), där x(t) är synkroniseringssignalen och T är dess längd. Ytterligare en komplikation är att detta filter också ska klara att signalen är polvänd.

Tyvärr så fungerade inte implementationen av filtret – det blev lurat av brus alldeles för ofta för vara användbart.

6.1.4.2 Andra försöket

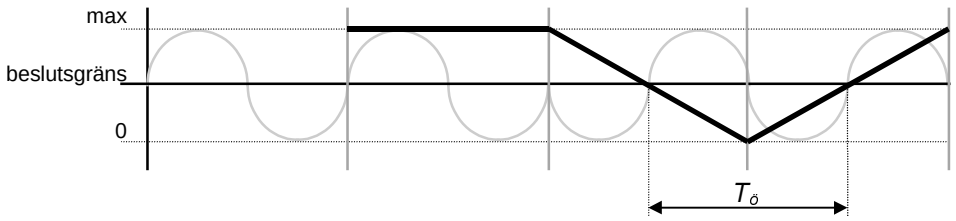
Erfarenheterna från första försöket sa att innan man börjar ta emot data måste man vara säker på att det verkligen är en signal och inte brus man tar emot.

Detta löses genom att låta sändaren först skicka en bärvåg som mottagaren detekterar. När bärvågen är detekterad så väntar mottagaren på ett känt mönster i signalen för att ställa in symbolklockan och därefter kan data tas emot.

Varje gång innan sändaren börjar sända data skickar den ut ett flertal sinusperioder med frekvensen 1 750 Hz det vill säga samma frekvens som datan moduleras med, dock helt utan fasskift. Bärvågen detekteras genom att signalen samplas flera gånger per period. Efter varje sampel görs beräkningar för att få veta om de 12 senaste perioderna var en bärvåg. Om så är fallet så startar inställningen av symbolklockan.

Problemet med symbolsynkroniseringen kan formuleras på följande sätt:
När man samplar signalen, hur vet man vid vilken sampling en ny symbol har börjat?

Mellan bärvågen och själva datan sänds fyra synkroniseringsbitar. Dessa fyra bitar är valda så att man får det vågformsutseende och räknarvärde i detektorn som visas i figur 6.3.



Figur 6.3 Symbolsynkronisering. Den tjocka linjen är räknarvärdet i detektorn. Vågformen kan vara polvänd, men räknarvärdet är ändå det samma.

$T_ö$ är tiden mellan övergångarna av beslutsgränsen. Beroende på hur många sampel man gör under tiden $T_ö$ så antar man att hälften av dessa är på den bit man för närvarande håller på att samplar. Således samplar man *antal sampel per period - antal samplingar under* $T_ö$ */2* gånger till på den sista synkroniseringsbiten och därmed är symbolsynkroniseringen klar och man är redo för att ta emot data.

En nackdel med den här metoden är att tidsmätningen startar och slutar då signalen är mest känslig för störningar men det fungerar bra i praktiken. Alternativt kan söka efter minimum på räknarvärdet men det är mer beräkningskrävande.

6.2 Datalänklager

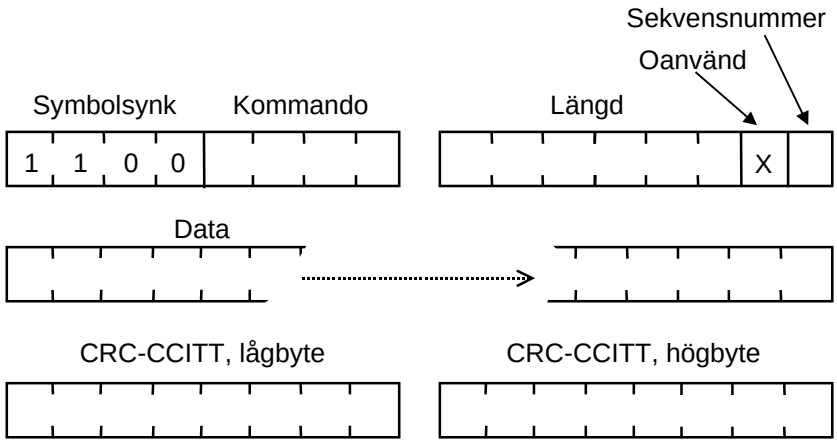
Datalänklagrets uppgift är att förpacka de möjligheter som det fysiska lagret erbjuder till en tjänst för att överföra data felfritt. För att kunna utföra tjänsten så delar datalänklagret upp data i ramar som den skickar iväg till mottagaren. Mottagaren skickar ramar tillbaka till sändaren för att bekräfta att ramarna har kommit fram felfritt. En ram innehåller förutom data, information som består av ramens längd, kontrollsumma på datan och information för flödeskontroll.

Funktionaliteten i datalänklagret har förändrats flera gånger under arbetets gång. Från början byggde det på designen av ett DTMF-baserat protokoll som senare skulle visa sig att vara problematiskt att implementera. Den nuvarande varianten har ett generellare ramformat och ett enklare datalänkprotokoll.

Först beskrivs det nuvarande ramformatet och hur det har förändrats, därefter beskrivs utvecklingen av datalänkprotokollet.

6.2.1 Ramformat

Ramformatet har under arbetets gång ändrats flera gånger. Förändringarna har oftast bestått av att storleken för de olika fälten har ändrats. Ibland har även helt nya fält tillkommit och andra har helt ändrat innehåll eller försvunnit. I figur 6.4 visas de fält som ingår.



Figur 6.4 Ramformatet i Phonix består av tre delar: ramhuvud, data och kontrollsumma.

6.2.1.1 Symbolsynk

De första bitarna i en ram utgörs av fyra symbolsynkroniseringsbitar. Dessa skapar en signal som har utseendet som beskrivs i figur 6.3 och på så sätt signalerar starten på en ram.

6.2.1.2 Längdfält

Längdfältet består av 6 bitar och anger antalet databytes ramen innehåller. Således är det möjligt att skicka upp till 63 stycken databytes i en ram. I praktiken sätter processorns begränsade RAM-minne stopp tidigare. En annan gräns är också att man bara kan skriva ned 32 bytes i taget i EEPROMet. I framtiden är det också tänkbart att utöka längdfältet med den outnyttjade biten vilket skulle ge möjlighet att skicka upp till 127 databytes förutsatt att man har tillräckligt med RAM-minne.

Genom att ha ett längdfält i ramhuvudet så kan mottagaren enkelt veta hur många bytes den ska ta emot. Man undviker samtidigt problemet med datatransparens som uppkommer då man inte vet ramens längd.

6.2.1.3 Kommando

Detta fält tillhör applikationslagret och borde därför ligga i datafältet men har placerats i ramhuvudet för att spara plats. Beskrivs närmare i avsnitt 6.4.

6.2.1.4 Sekvensnummer

Ett sekvensnummer bestående av en bit räcker för kunna skilja på nya ramar och omsändningar av gamla. Hanteringen av sekvensnummer och dess syfte beskrivs nedan.

6.2.1.5 Data

Längden på datafältet är känd så behöver datan inte kodas på något sätt.

6.2.1.6 Kontrollsumma

För att upptäcka fel i överföringen så beräknas en kontrollsumma på datafältet. Kontrollsumman beräknas enligt metoden CRC (*Cyclic Redundancy Check*) [Williams] med 16 bitar med generatorpolynom enligt CCITT [ITU]. Denna metod kan upptäcka alla felskurar med längd 16 eller mindre.

Först användes en metod där alla bytes i ramen summerades med modulo-2 addition. Den metoden klarar av att upptäcka ett bitfel per bitposition i hela ramen. Detta betyder att en ram med två bitfel kan slinka igenom oupptäckt samtidigt som man kan upptäcka fel i en ram med 8 bitfel, beroende på bitfelens placering. Metodens stora fördel var att den tog väldigt enkel att implementera, man hann faktiskt med att beräkna kontrollsumman under pågående mottagning.

Det som gjorde att CRC-CCITT valdes var upptäckten av hur enkel den var att implementera [Warren] samt insikten av hur enkelt en felaktig ram förblir oupptäckt med den första metoden.

6.2.1.7 Oanvänd

Denna bit är en kvarleva från tidigare datalänkprotokoll som beskrivs nedan. Nu finns den kvar för att enkelt kunna utöka längdfältet eller fältet för sekvensnummer.

6.2.2 Datalänkprotokoll

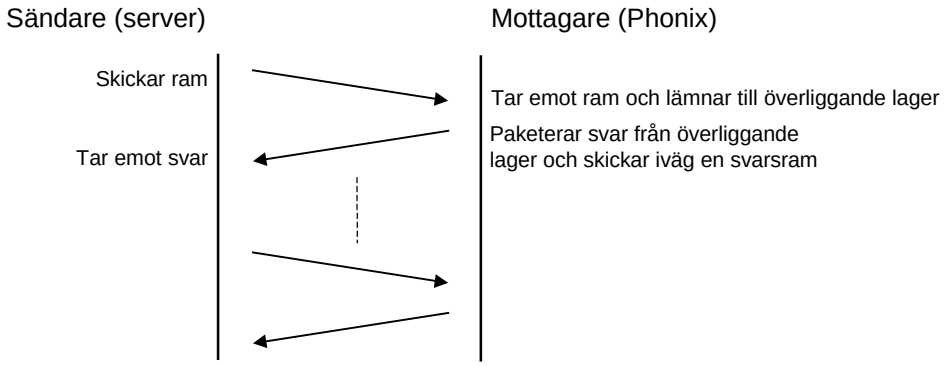
Kort sagt med några fackuttryck är datalänkprotokollet av typen *Positive Acknowledgement with Retransmission (PAR) and piggybacking*. Detta sammanfattar protokollet ganska bra om man vet vad det betyder.

Positive Acknowledgement with Retransmission (PAR) betyder att bekräftelser tolkas som att ramen har kommit fram på rätt sätt. Fås ingen bekräftelse så sker en omsändning.

Piggybacking betyder att man i bekräftelser lägger in data i ramen som ändå ska åt samma håll. Utan *piggybacking* måste man kunna byta riktning på kommunikationen för att transportera data i båda riktningarna. Att byta kommunikationsriktningen visar sig i stycke 6.2.2.2 att vara förenat med vissa problem.

Grundidén i det nuvarande datalänklagret är att det alltid är samma part, kommunikationsservern, som hela tiden skickar en ram i taget och väntar på svar från

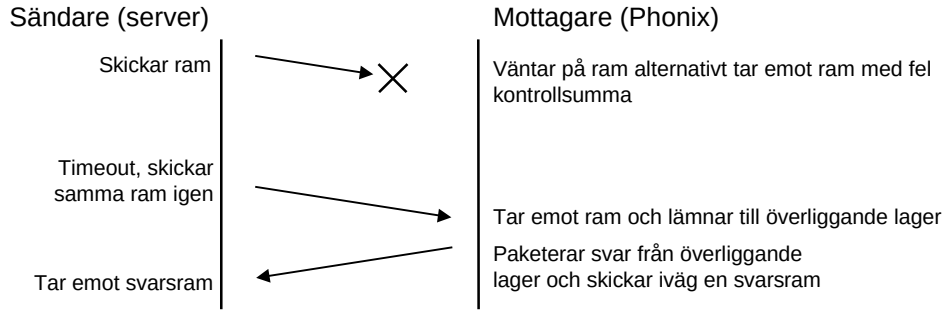
mottagaren, Phonix. När servern har fått svar skickar den nästa ram och väntar på svar och så vidare. I figur 6.5 visas ramflödet i normala fall.



Figur 6.5 Ramflödet mellan server och Phonix. Texten vid de två vertikala linjerna talar om vad sändaren och mottagaren gör vid olika tidpunkter. Pilarna visar ramar som skickas.

Eftersom det finns brus på kanalen kan det bli fel på ramar eller ramar kan försvinna helt. Fel på ramar upptäcks med hjälp av att kontrollsumman. Det finns två olika händelser som orsakar fel kontrollsumma. Den ena är att det uppstår enstaka bitfel i ramen och den andra är att brus och störningar lurar bärvågsdetektorn så att man tar emot skräp. Det enda datalänklagret kan göra när kontrollsumman inte stämmer är att ignorera den felaktiga ramen och vänta på nästa ram.

Antag att en ram från servern försvinner eller tas emot med fel kontrollsumma, händelserna är ekvivalenta för datalänklagret. Phonix får då ingen korrekt ram och kan alltså inte skicka något svar. För att undvika att servern väntar i evighet på ett svar som aldrig kommer, väntar den en viss tid. Har servern inte fått svar inom denna tid så skickar den samma ram igen. Figur 6.6 visar hur det går till.



Figur 6.6 En ram från servern försvinner och sänds igen. En pil med ett kryss framför betyder att ramen inte togs emot korrekt.

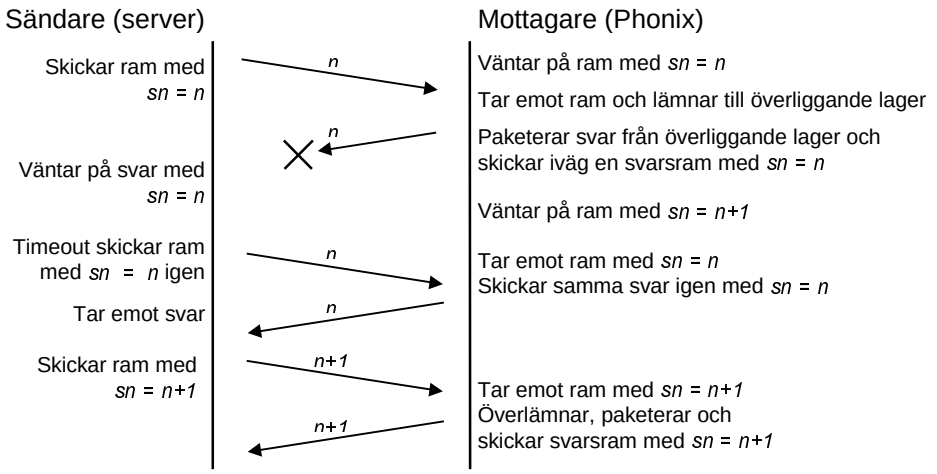
Nu är det tyvärr så att även svarsramar från Phonix kan försvinna, men eftersom servern inte kan skilja på dessa fall så gör den på samma sätt som i figur 6.6. Detta är inte ett helt felfritt beteende men kan avhjälpas med införande av sekvensnummer på ramarna.

6.2.2.1 Hantering av sekvensnummer

Antag att Phonix tar emot en ram från servern, lämnar den till överliggande lager och paketerar och skickar en svarsram till servern. Svarsramen försvinner, servern får en timeout och skickar samma ram igen. Phonix tar emot ramen och överlämnar den till

överliggande lager och därmed har datalänklagret överlämnat samma data två gånger till överliggande lager.

Man vill att datalänklagret ska kunna skilja på en ny ram och en omsändning av en gammal ram. När datalänklagret upptäcker en ram som den redan har överlämnat till överliggande lager så skickar det den svarsram som motsvarar den mottagna ramen utan att överliggande lager blandas in. För att kunna skilja på ramarna så behövs det ett sekvensnummer i ramhuvudet. Figur 6.7 visar hur det går till.

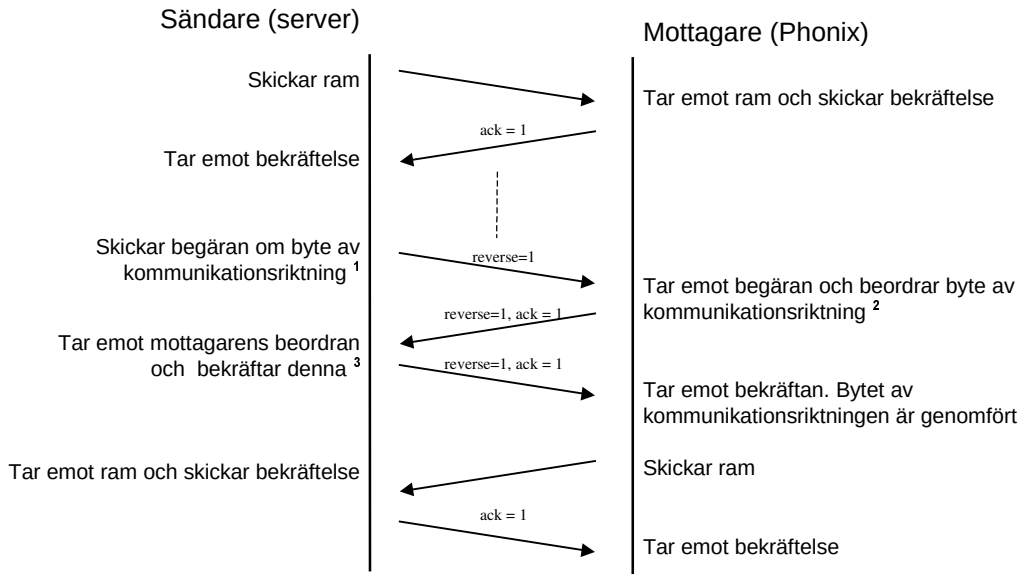


Figur 6.7 Phonix svarsram försvinner och servern gör en omsändning.

Med metoden som beskrivs ovan så får alltid servern svar med rätt sekvensnummer oavsett om den skickar nya ramar eller omsändningar och inga ramar lämnas två gånger till överliggande lager.

6.2.2.2 Ursprungligt datalänkprotokoll

I det ursprungliga datalänkprotokollet skickades ingen data från mottagaren till sändaren utan det fanns funktionalitet för att byta kommunikationsriktning. För att göra detta behövs det en bit (*reverse*) i ramhuvudet för att signalera detta samt en bit (*ack*) för att skilja vanliga ramar från bekräftelseramar. Hur byte av kommunikationsriktningen var tänkt att fungera visas i figur 6.8.



Figur 6.8 Byte av kommunikationsriktning. Då inget anges är bitarna reverse och ack satta till 0.

Problemet som uppstod med den här metoden var att om ramen mottagaren skickar vid [2] försvinner. Det som händer är att båda parter väntar på en ram som aldrig kommer. Sändaren väntar på den försvunna ramen[2] och mottagaren väntar på svaret[3] till den. Båda parter börjar därefter att sända om sina ramar[1,2] då de inte har fått svar inom en viss tid. Detta medför att kollisioner uppstår och att inga ramar kommer fram över huvud taget trots att parterna väntar olika länge innan de sänder om sina ramar. Fullständig analys av vad som egentligen var fel gjordes aldrig innan metoden övergavs till förmån för den först beskrivna varianten av datalänkprotokoll.

6.2.2.3 Övrig felhantering

Övrig felhantering innefattar händelser som varken server eller Phonix har någon större kontroll över, det vill säga möjligheten att upptäcka att inga ramar kommer fram som avsett. Servern löser detta genom att ha en gräns på hur många gånger den försöker att sända samma ram. Överskrids denna gräns lägger servern helt enkelt på luren. Phonix har ett liknande beteende. Om den inte får en ram inom en viss tid så lägger den också på luren.

Okontrollerad nedkoppling

Phonix kopplar också ned om någon lyfter luren till en telefon som är kopplad till den så att en uppdatering inte hindrar utgående samtal. Detta är en okontrollerad nedkoppling och fungerar på följande sätt. Är klykan av, det vill säga någon har lyft luren, så sätts en flagga. Innan datalänklagret ska sända en svarsram till servern så kontrolleras flaggan. Om flaggan är satt så ändrar datalänklagret kommandofältet i ramen till ett värde som motsvarar kommandot *shutdown* och skickar iväg ramen för att sedan lägga på luren. Detta förutsätter att det finns en ram att svara på. Om Phonix istället råkar vänta på en ram från servern och flaggan sätts så läggs luren på direkt.

När datalänklagret i servern upptäcker att kommandofältet är satt till *shutdown* så lägger servern också på luren. Skulle denna ram aldrig komma fram till servern så lägger servern inte på luren förrän den har sänt om samma ram för många gånger. Detta hindrar ändå inte utgående samtal från Phonix eftersom förbindelsen kopplas ned ändå när luren har varit pålagd i ett par sekunder.

6.3 Nätverkslager

Nätverkslagrets uppgift är att sköta uppkoppling, vägval och nedkoppling av förbindelser. Vägval är trivialt ty telefonnätet är uppkopplingsorienterat av sin natur vilket gör att vägvalet görs endast en gång i början av varje uppkoppling.

Nätverkslagret i Phonix är implementationsmässigt sett inte ett separat protokollager. Nätverkslagret är implementerat som kommandon i applikationslagret men funktionaliteten är den samma som för ett separat nätverkslager. Detta uppnås genom att det första kommandot som applikationslagret tillåter är uppkoppling (*hello*) och det andra kommandot är autenticiering (*challenge*). Inga andra kommandon kan utföras innan dessa två är utförda. Nedkopplingskommando kan utföras när som helst efter dessa två kommandon.

6.3.1 Uppkoppling och autenticiering

Uppkopplingen startar i normala fall med att Phonix ringer upp servern genom att helt enkelt slå serverns telefonnummer. När servern har svarat så sänder den ut en DTMF-sekvens samt ett uppkopplingspaket med modemet. DTMF-sekvensen används för att Phonix ska veta att det är en server som ringer så den avbryter styrprogrammet och övergår till uppdateringsläge. DTMF-sekvensen används också för att koppla ifrån automatsvarande enheter (som till exempel faxar och telefonsvarare). Denna finess är endast användbar då servern ringer upp Phonix.

Phonix svarar med sitt serienummer och ett slumptal som används för autenticiering. Phonix skickar också med ytterligare ett värde som talar om varför den ringer servern.

6.3.1.1 Autenticiering

Autenticieringen används för att förhindra att en server kan ringa upp en konkurrents samtalsroutrar. Autenticieringen bygger på att varje Phonix har en hemlig nyckel. Denna nyckel känner endast den tillåtna servern till. När servern vet vilken Phonix som den har förbindelse med kan den slå upp vilken hemlig nyckel som den uppringande Phonix-routern har. Den hemliga nyckel och slumptalet används som indata till en envägsfunktion som genererar ett tal som skickas tillbaka till Phonix. Phonix applicerar samma funktion på den hemliga nyckeln och slumptalet och kontrollerar att resultatet från servern är det samma. Är resultatet från servern fel så kopplas förbindelsen ned.

Detta förfarande innebär att den hemliga nyckel aldrig sänds över den osäkra telefonkanalen vilket är bra då avlyssning inte ger någon information om den hemliga nyckel.

6.3.2 Nedkoppling

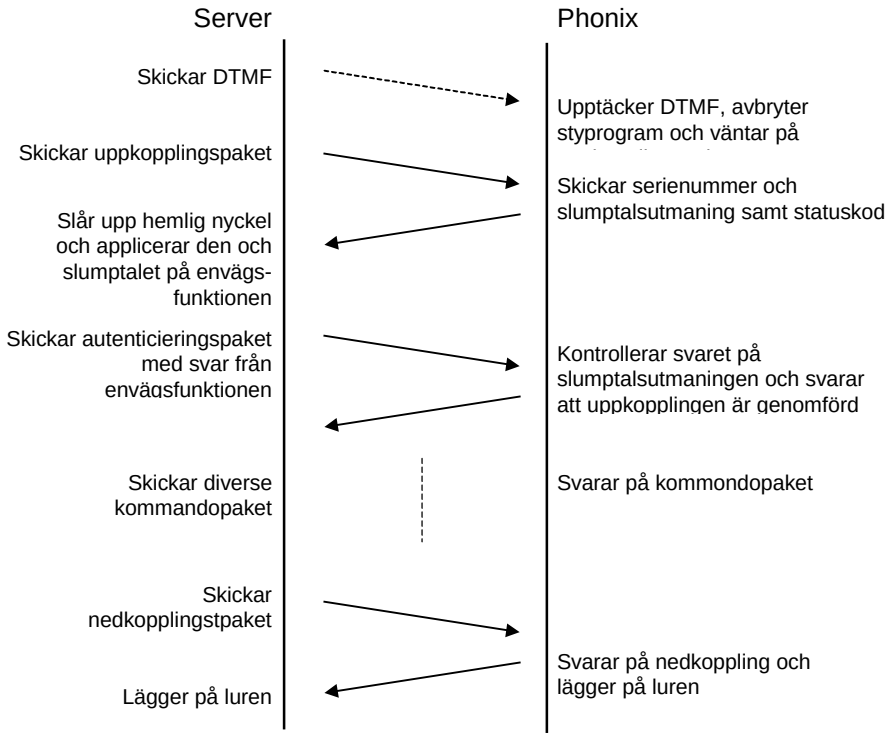
Nedkoppling kan ske på två sätt; kontrollerat eller okontrollerat. Kontrollerad nedkoppling sker genom att servern skickar ett nedkopplingspaket som Phonix skickar ett svar på. Därefter kopplas förbindelsen ned genom att server och Phonix lägger på luren.

Okontrollerad nedkoppling hanteras av datalänklagret och beskrivs ovan i avsnitt 6.2.2.3. Implementationen av den okontrollerade nedkopplingen tänjer en hel del på indelningen i protokollager. Formellt är den ett kommando i applikationslagret trots att funktionaliteten hos kommandot hör hemma i nätverkslagret men är implementerat i datalänklagret. Anledningen till detta är den hårda koppling applikationslager och datalänklager har för att kunna implementeras utan att

förbruka mycket programutrymme. Denna koppling beskrivs närmare i applikationslagret.

6.3.3 Exempel på upp- och nedkoppling

I figur 6.9 visas, med notation liknande den i figurerna i datalänklagret, hur upp- och nedkoppling går till.



Figur 6.9 Exempel på hur uppkoppling, autenticiering och nedkoppling.

6.4 Applikationslager

Applikationslagret är det lager som utnyttjar den pålitliga datatransport som de undre lagren erbjuder och förpackar detta till tjänster som möjliggör snabb och säker uppdatering av styrprogrammet i Phonix.

De tjänster man önskar sig av applikationslagret är följande:

- Kontrollera vilket styrprogram som finns i samtalsroutern
- Hämta data som samtalsroutern har samlat in
- Byta styrprogram i samtalsroutern
- Kontrollera att byte av styrprogram utfördes korrekt

Dessa tjänster kan översättas till kommandon som kan läsa, skriva och beräkna kontrollsumma på områden i samtalsrouterns minne (EEPROM).

Följande extra funktioner är användbara under utveckling och testning:

- Tända och släcka samtalsrouterns lysdiod

- Samla in statistik om händelser i datalänklagret

Statistiken om händelser i datalänklagret kan även användas till att mäta kvaliteten på förbindelsen mellan server och samtalsrouter.

6.4.1 Kommandouppbyggnad

Varje kommando består av en kommandoidentifierare och en uppsättning parametrar samt tillhörande data. Kommandoidentifieraren består av 4 bitar vilket ger 16 olika kommandon. För att komma förbi denna begränsning finns ett utökat kommando. En översikt över vilka kommandon och vilka kommandoidentifierare de har visas i tabell 6.1. Identifierare 9 t.o.m. 14 används inte för närvarande.

6.4.1.1 Gemensamma parametrar

Parametrarna enhet, adress och längd är gemensamma för flera kommandon.

Enhet

Enhet anger vilket minne som avses i en flerlinjers-Phonix. Phonix samtalsroutrar finns i versioner med flera linjer där varje linje kan har ett separat styrprogram. Dock så sker kommunikationen med servern alltid på en och samma linje vilket innebär att den linjen distribuerar kommandon till de övriga linjerna. Fördelen med detta är att servern med en uppkoppling kan uppdatera flera linjer samtidigt. Denna parameter skickas alltid med men ignoreras av enlinjers samtalsroutrar för att inte införa kompabilitetsproblem.

Adress

Adress anger på vilken adress i minnet kommandot ska utföras. Adress anges med 16 bitar med den minst signifikanta byten först. 16-bitars adresser gör det möjligt att adressera 64 kilobyte minne vilket kan anses som fullt tillräckligt då samtalsroutrarnas minnen rymmer 8 kilobyte. Det största seriella EEPROMen, av den typ som används i samtalsroutrarna, som finns att tillgå idag rymmer 16 kilobyte [Löwenbrand].

Längd

Längd anger hur stort område, i antal bytes, kommandot ska utföras på. Längd anges med 16 bitar med den minst signifikanta byten först.

6.4.2 Kommandon

I tabell 6.1 listas vilka kommandon som finns och i följande stycken beskrivs varje kommando för sig.

Tabell 6.1 Översikt av kommandon i Phonix applikationslager

Kommando	Kommandoidentifierare
Extended	0
Hello	1
Challenge	2
Writemem	3
Readmem	4
Statistics	5
ControlLED	6
Getchecksum	7
Disconnect	8
Shutdown	15

6.4.2.1 Writemem

Writemem-kommandot används för att skriva i samtalsrouterns minne. Servern skickar ett writemem-kommando som består av enhet, adress, längd och data. Format visas i figur 6.10. Samtalsroutern tolkar parametrarna och skriver ned data på angiven adress. Samtalsroutern skickar sedan en bekräftelse till servern för att meddela att kommandot är utfört. En bekräftelse består endast av kommandoidentifierare.

Kommando Id(½)	Enhet (1)	Adress (2)	Längd (2)	Data (1-65535)
3	0	0x00,0x04	0x12,0x0a	0x4d,0x47,...

Figur 6.10 Ett writemem-paket. Siffrorna inom parentes anger antal bytes varje fält upptar.

6.4.2.2 Readmem

Readmem-kommandot används för att läsa i samtalsrouterns minne. Servern skickar ett readmem-kommando som består av enhet, adress och längd. Alltså samma format som i figur 6.10 fast utan datafältet. Samtalsroutern läser angivet minnesblock och paketerar det i ett svarspaket som skickas till servern, se figur 6.11.

Kommando Id(½)	Data (1-65535)
4	0x4d,0x61,0x72,0x63,0x75,0x73,0x20...

Figur 6.11 Ett svarspaket på ett readmem-kommando.

Implemtationen av readmem-kommandot innehåller en spärr mot att läsa vissa minnesadresser för att exempelvis skydda den hemliga nyckel som används för autenticiering. Skyddet fungerar så att resultatet alltid blir nollor när man försöker läsa dessa områden.

6.4.2.3 Statistics

Statistics är ett kommando som används för att få statistik över olika händelser i datalänklagret. Följande händelser räknas:

- Sändning av ram

- Omsändning av ram
- Mottagning av ram
- Mottagning av ram med fel sekvensnummer
- Mottagning av ram med fel kontrollsumma

Genom att sammanställa dessa händelser från datalänklagrerna hos båda parter får man en bra bild av händelseförloppet.

Ett statistics-kommando består enbart av kommandoidentifieraren. Samtalsroutern paketerar statistikdata i en bekräftelse som sänds till servern. Att man tar emot en ram med fel sekvensnummer räknas endast i servern för att för samtalsroutern betyder det att servern sänder om den föregående ramen, se figur 6.7. Detta ger i sin tur att servern aldrig kan ta emot en ram med felaktigt sekvensnummer såvida inte bitfel uppstår och ändrar sekvensnummersbiten och andra bitar så att kontrollsumman fortfarande stämmer. Detta har aldrig hänt och kommer antagligen aldrig att göra det heller.

6.4.2.4 ControlLED

ControlLED är ett kommando som används för att kontrollera lysdioden i felsökningssyfte på samtalsroutern. Ett ControlLED-kommando har endast en byte som parameter. Värdet noll på parametervärdet tolkas som att lysdioden ska vara släckt, övriga värden tänder lysdioden. När kommandot är utfört sänds en bekräftelse på utfört kommando bestående endast av kommandoidentifierare.

6.4.2.5 Getchecksum

Getchecksum är ett kommando som beräknar CRC-32 på angivet minnesområde. Servern skickar ett paket med samma parameterformat som readmem-kommandot. Samtalsroutern beräknar minnesområdets kontrollsumma och sänder resultatet (4 bytes) till servern i en bekräftelse.

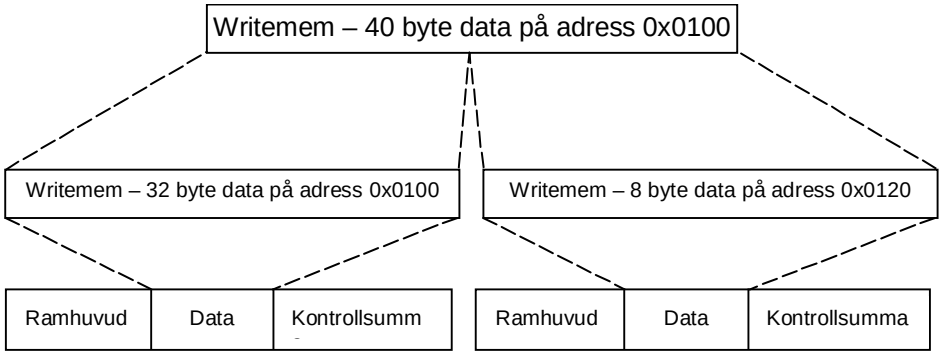
Att beräkna kontrollsumma på minnesområden är ett effektivt sätt för servern att kontrollera att det den har skrivit i samtalrouterns minne är korrekt. Utan detta kommando skulle servern behöva läsa tillbaka hela styrprogrammet och jämföra det med det styrprogram servern anser sig ha skrivit ned. Detta beteende skulle dubblera uppdateringtiden.

6.4.2.6 Extended

Extended är inget enskilt kommando utan det är tänkt att användas för framtida utökningar av applikationslagret. Extended har en byte som parameter som anger vilket utökat kommando som ska utföras. För närvarande är extended inte implementerat på grund av att det fortfarande finns gott om kommandoidentifierare lediga.

6.4.3 Koppling applikationslager — datalänklager

Den hårda kopplingen mellan applikationslager och datalänklager består av applikationslagrets kommandon delas upp i flera mindre kommandon vars argument och resultat ryms i en enda ram i datalänklagret. Detta görs för att spara in komplexitet och kod i applikationslagret. För närvarande är det endast writemem- och readmem-kommandona som drabbas av denna uppdelning. Hur ett writemem-kommando delas visas i figur 6.12. Readmem-kommandon delas upp så att dess svar alltid ryms i en enda ram.



Figur 6.12 Uppdelning av writemem-kommando till två nya kommandon vardera ryms i en ram.

Alternativet till uppdelningen som visas i figur 6.12 är att man skickar ett paket med argument och 40 bytes data. Detta paket delas upp i två ramar, den första innehåller argument samt 32 bytes data och den andra innehåller endast de resterande 8 databyten. För att kunna göra på detta sätt måste applikationslagret kunna hantera att dess paket kan delas upp på flera ramar. Detta är den önskvärda funktionaliteten men det tar tyvärr för mycket programminne att hålla reda på om varje kommando har fått alla ramar det behöver. Den nuvarande lösningen fungerar men den genererar något längre ramar som behöver sändas. Detta beror på att man skickar upp till 5 byte kommandoargument extra i varje ram som man inte behöver göra med den nyss beskrivna metoden.

7 Resultat

Detta kapitel sammanfattar resultaten av arbetet med utvecklingen av mjukvarumodemet i Phonix. Först beskrivs det implementerade systemet följt av hur det kan förbättras och till sist nämns de erfarenheter arbetet har givit.

7.1 Implementerat system

De delproblem som togs upp i kapitel 5 är lösta på följande sätt:

Den begränsade beräkningskapaciteten har hela tiden utmanat till att implementera de valda algoritmerna så effektivt som möjligt för att uppnå högsta möjliga datahastighet. 1 750 bps innebär att det går åt 512 instruktioner för att modulera och demodulera en bit.

Att generera en signal var inte särskilt svårt då man väl hade fått fram ett kodskelett där man enkelt kunde variera pulsbredder och periodtider. Det svåra var att implementera kodning av data utan att behöva sänka modulationshastigheten.

Valet av algoritm för att demodulera en DPSK-signal visade sig vara ett lyckokast. Det svåra var att göra en effektiv implementation så att man kan sampla så ofta som möjligt för att få ett bättre beslutsunderlag vid avkodning samtidigt som man behöver instruktioner till att lagra undan avkodad data.

Kommunikationsprotokollet blev bra trots att de olika protokollagren har pressats ihop på grund av utrymmesskäl. Hela protokollet tar ungefär 800 instruktioner i samtalsrouterns programminne och en del mer i serverns på grund av det är den som styr hela förloppet samt att där har inte utrymmeskraven varit lika hårda. Det lager som var svårast att få ordning på var datalänklagret som reviderades ett flertal gånger men med Tanenbaums motto *"Keep It Simple Stupid"* i bakhuvudet blev det till slut bra.

Ett annat problem som vållade stor huvudbry var synkroniseringen som dessutom till en början hade förbisetts. Med en del grävande i böcker och bollande med olika idéer och ett antal olika implementationer så hittades en fungerande metod.

7.1.1 Prestanda

Prestandan hos modemet har visat sig vara hög förutsatt att linjekvaliten är god. Den teoretiska gränsen för modemets datahastighet är 1 260 bps, om man räknar med fördröjningar vid bekräftelser och vid beräkning av kontrollsumma, och bara räknar med data, det vill säga bärvåg, ramhuvud och kontrollsumma räknas inte. Fälttester har visat datahastigheter på 1 190 bps vid goda förhållanden vilket får anses ganska vara nära den teoretiska gränsen. Differensen beror antagligen på fördröjningar som

uppstår i kommunikationsservern. En tidig implementation av kommunikationsservern gav endast drygt 1 000 bps vid goda förhållanden.

På riktigt dåliga förbindelser sjunker datahastigheten drastiskt. Tester gjordes på en koppling från Sverige till Tyskland och tillbaka till Sverige via England. Vid vanliga samtal på denna förbindelse så är störningarna påtagliga — skorrande, knäppande, mycket bakgrundsbrus, fördröjningar och tar bort början på meningar. För modemet är det svårt att få fram ramar felfritt, men efter justeringar som gör att endast korta ramar skickas uppnår man hastigheter på 30–50 bps. Inte särskilt användbart för uppdateringar av samtalsroutrar men det visar att modemet fungerar även på dåliga förbindelser.

7.2 Förbättringar

Det finns en hel del förbättringar av modemet som kan göras om man ändrar förutsättningarna något. I detta stycke beskrivs vilka förbättringar och förändringar som kan göras och varför de inte är gjorda. Med detta menas hur man kan göra modemkommunikationen snabbare, säkrare och effektivare.

Först beskrivs vilka förbättringar och förändringar i hårdvaran som kan göras. Därefter beskrivs hur dessa förändringar påverkar möjligheterna att förbättra mjukvaran.

7.2.1 Hårdvara

Nedan nämns några sätt att förbättra modemets prestanda och kvalitet genom förändringar i hårdvaran.

Förändringar av hårdvaran vid modemets in- och utgångar

Ett bättre filter på modemutgången som dämpar övertonerna från pulsbreddmodulatorn skulle göra att signalnivån skulle kunna höjas något och på så sätt öka störtåligheten ytterligare. Idag är inte signalnivån den högsta tillåtna just på grund av signalens innehåll av övertoner. Ett bättre filter betyder också fler komponenter vilket ger en högre tillverkningskostnad. Med en högre klockfrekvens i processorn kan generera en bättre signal vilket innebär att nuvarande filter är tillräckligt bra.

På modemingången är spektrumet av förbättringar stort. Allt från att införa en A/D-omvandlare till DPSK-avkodare i hårdvara. Att införa en DSPK-avkodare, om det nu finns några, är antagligen inte ekonomiskt försvarbart. Att använda en Schmitt-trigger har visat sig inte ge märkbart bättre prestanda. Med en A/D-omvandlare skulle man kunna göra avkodningen av data effektivare. En A/D-omvandlare skulle också kunna användas till andra uppgifter, till exempel att detektera kopplingston och DTMF-siffror och på så sätt spara in andra komponenter.

Högre klockfrekvens

Dagens processor finns i en variant med 20 MHz klockfrekvens. Att använda den varianten istället för den med 3,58 MHz ger stora möjligheter till förbättringar i utformningen av algoritmerna i mjukvaran. Dagens val av klockfrekvens beror på att processorns klockkristall även är klocka åt den DTMF-transceiver som ingår i hårdvaran. DTMF-transceivern kräver klockfrekvensen 3,58 MHz och ytterligare en kristall vore för dyrt.

Annan processor

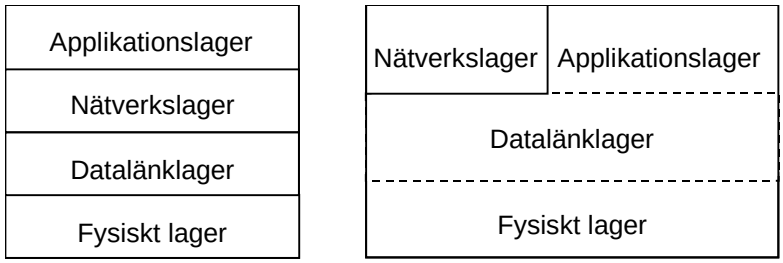
Att byta till en snabbare processor med större programminne och registerutrymme samt med andra inbyggda finesser som A/D-omvandlare och multiplikation i hårdvara skulle ändra förutsättningarna radikalt för modemet. Både med avseende på prestanda och implementation. Det högre priset kräver dock att en sådan processor tar över fler uppgifter som idag hanteras av annan hårdvara.

7.2.2 Mjukvara

I mjukvaran finns det många vitt skilda förbättringar som kan göras men som för närvarande hindras av processorprestanda eller minnesutrymme. Nedan beskrivs flera möjligheter.

Striktare gränser mellan de olika protokollagren

Hur de olika lagren hänger ihop visas i figur 7.1, till vänster visas ambitionen och till höger hur det blev på grund av hårdvarans begränsningar.



Figur 7.1 Protokollager i Phonix. Till vänster den eftersträvade formen och till höger den verkliga.

Att nätverkslagret är implementerat som en del i applikationslagret är antagligen inget hinder för att införa flera olika uppkopplingsmetoder men det skulle underlätta om det var ett separat lager.

Den hårda kopplingen mellan applikationslager och datalänklager vill man abstrahera bort. Vitsen med att ha en väl definierad gräns mellan dessa två lager är att man då enklare kan utnyttja datalänklagrets tjänster till att enkelt implementera nya tjänster/kommandon i applikationslagret.

Idag är datalänklagret och det fysiska lagret också hårt kopplade. Det fysiska lagret och den nedre delen av datalänklagret består endast av två funktioner, *sänd ram* och *ta emot ram*. Med bättre abstraktion skulle det vara möjligt att kunna införa alternativa fysiska lager. Detta gör att man kan implementera flera modulationsmetoder för att använda vid olika linjekvaliteter.

Justering av bitklockan under mottagning

Att justera bitklockan under mottagning skulle göra signalen mer okänslig mot frekvensförskjutningar som telefonkanalen kan införa. För närvarande räcker inte beräkningskraften i processorn för att göra detta. I vilken omfattning frekvensförskjutningar genererar bitfel är inte utrett.

Datakomprimering

Att komprimera data är ett enkelt sätt att öka överföringshastigheten. Detta kräver en snabbare processor och med större RAM-minne för att bli användbart.

Annan modulationsform

Med högre klockfrekvens kan man enkelt öka modulationshastigheten till 2 400 symboler per sekund och demodulera en 4PSK-modulerad signal med dagens algoritmer. Detta innebär en datahastighet på 4 800 bps. Med hjälp av en A/D-omvandlare kan man kanske demodulera mer avancerande modulationsformer, 16-QAM med en faltningskodare till exempel.

Felrättande kod

Att införa en felrättande kod i kombination med en felupptäckande innebär att man kan klara telelinjer med sämre kvalitet men kräver mer beräkningskraft än vad som finns tillgängligt idag.

7.3 Erfarenheter

Att utveckla mjukvarumodemet i Phonix har varit arbetsamt, utmanande, stimulerande och frustrerande. Frustrationen har bestått i svårigheter att få utdata från processorn om vilket tillstånd programmet befinner sig i. Oscilloskop och lysdioder är två användbara verktyg men räcker inte alltid till. Tyvärr finns det heller inga väl fungerande hårdvaruemulatorer för processorn.

Hårdvarans begränsning känns också ibland frustrerade men samtidigt är det utmanande att försöka pressa upp prestandan genom att utnyttja all tillgänglig kapacitet i processorn.

Det har varit mycket stimulerande att arbeta på ett litet företag i fronten inom sitt område. Man känner sig snabbt hemma när man kan vara med och påverka designen och förutsättningarna för det man utvecklar.

Kreatel Communications har fått värdefulla erfarenheter om vilka möjligheter det finns för ett mjukvarumodem i en samtalsrouter. Dessa erfarenheter har delvis lett till att nästa version av Phonix samtalsrouter kommer att byggas kring en egenutvecklad processor som är 6 gånger snabbare än den nuvarande samt att den har inbyggda A/D-omvandlare och mycket mer minne.

Mina egna erfarenheter består bland annat av att man verkligen har fått tillämpa sina kunskaper i digital kommunikation och kommunikationsprotokoll samt att jag har fått ökade kunskaper om hårdvara i telenätet. Jag har också fått vetskap om hur produktutveckling går till och vilka olika steg som ingår.

8 Referenser

De källor och det material som har använts under arbetets gång delas in i tre grupper för att tydligt skilja olika media från varandra. Referenser märkta med • refereras inte explicit från texten men har varit till hjälp och stöd i arbetet.

8.1 Tryckt material

Ahlin, L., Zander J., *Digital radiokommunikation*, Studentlitteratur. •

Gustafsson, M., *Phonixmodemet i detalj*, Kreatel Communications AB.

Halsall, F., *Data communications, computer networks and open systems*, Addison Wesley.

Haykin, Simon, *Digital Communications*, John Wiley & Sons.

Helmers, D., *Kommunikationsserver i system av samtalsroutrar*, Linköpings Universitet (LiTH-IDA-Ex-9653).

ITU, *CCITT Red book volume VIII - Fascicle VIII Data communication over the telephone network - recommendations of the V series*, ITU.

Lin, S., Costello D., *Error Control Coding: Fundamentals and Applications*, Prentice Hall. •

Microchip 95, *PIC16/17 Microcontroller Data Book*, Microchip.

Microchip 94, *Embedded control handbook*, Microchip. •

Tanenbaum, A., *Computer networks, 3rd edition*, Prentice Hall.

8.2 Material i elektronisk form

Williams, R. N., *A Painless Guide to CRC Error Detection Algorithms*, ftp://ftp.rocksoft.com/clients/rocksoft/papers/crc_v3.txt, Rocksoft Pty Ltd.

Warren, A., The Embedded Systems Programming Answer Line, http://www.geocities.com/SiliconValley/2499/, Fast Forward Engineering.

8.3 Personlig kommunikation

Följande personer på Kreatel Communications AB har varit till stöd för arbetet i form av bollplank och inspirationsgivare.

Helmers, Daniel, Utvecklare av kommunikationsserver. •

Nilsson, Jonas, Hårdvarukonstruktör. •

Melin, Påhl, Teknisk chef. •

Malm, Oskar, Kompilatorkonstruktör. •

Löwenbrand, David, VD.