Voice over IP

Gateway og Terminal

Gruppe 550, Datateknik 1999

INSTITUT FOR elektroniske systemer AALBORG universitet

AALBORG UNIVERSITET
INSTITUT FOR ELEKTRONISKE SYSTEMER
AFDELING FOR PROCESKONTROL

Frederik Bajersvej 7 ▪ DK 9220 AALBORG Ø ▪ Telefon 98 15 85 22

TITEL:	VoIP gateway og terminal
TEMA:	Sandtidskommunikation
PROJEKTPERIODE:	5. semester, datateknik
PROJEKTGRUPPE:	550

GRUPPEMEDLEMMER:
Claus Albøge
Mads G. Christensen
Tonny Gregersen
Karsten Jensen
Peter Korsgaard
Lars J. Kristensen
Robert Stepien

VEJLEDERE:
Arne Skou
Jens Jakobsen

OPLAGSTAL:	11
ANTAL SIDER:	134
ANTAL BILAG:	0
FÆRDIGGJORT:	17/12/99

Synopsis

Denne rapport omhandler udviklingen af et realtids Voice over IP-system, nærmere bestemt enhederne i en H.323-zone: gateway og terminal. Gateway'en er en gateway mellem et IP-LAN og et antal ISDN2-forbindelser. Til terminalen på IP-LAN'et er et antal PSTN-telefoner tilkoblet vha. en kombination af lydkort og parallelport. Systemet udvikles med udgangspunkt i Anbefaling H.323 for multimediakommunikation over et pakkebaseret netværk. Til netværksprogrammeringen er sockets (winsock) anvendt og til lydkortskommunikationen DirectSound 7.0. Programmet er multitrådet fremstillet til afvikling under Windows 2000. Analyse, design og implementation er foretaget objektorienteret. Desuden undersøges forsekllige Windows-versioners egnethed mht. realtidsafvikling.

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS
DEPARTMENT OF CONTROL ENGINEERING

Frederik Bajersvej 7 ▪ DK 9220 AALBORG OE ▪ Telephone 98 15 85 22

TITLE:	VoIP Gateway and Terminal
THEME:	Real-time Communication Systems
PROJECT PERIOD:	5th semester, Computer Engineering
PROJECT GROUP:	550

PARTICIPANTS:
Claus Albøge
Mads G. Christensen
Tonny Gregersen
Karsten Jensen
Peter Korsgaard
Lars J. Kristensen
Robert Stepien

SUPERVISORS:
Arne Skou
Jens Jakobsen

COPIES:	11
NUMBER OF PAGES:	134
ENCLOSED PAGES:	0
FINISHED:	12/17/99

Abstract

This report documents the development of a real-time Voice over IP system, i.e. the units of a H.323 zone: a gateway and a terminal. The gateway is a gateway between an IP LAN and a number of ISDN2-connections. A number of PSTN telephones are connected to the terminal residing on the IP-lan using a combination of a sound card and a parallel port. The development of the system takes its starting point in Recommendation H.323 for multimedia communication on a packet switched network. The network programming is based on sockets (winsock) and the sound card on DirectSound 7.0. The programs are multi threaded inteded for use in Windows 2000. Analysis, design and implementation is object oriented. Also a study of the different Windows versions as to their real-time capabilities is conducted.

Forord

Denne rapport udgør dokumentationen af projektgruppe 550's arbejde på datateknik 5. semester 1999 på Institut for Elektroniske Systemer, Aalborg Universitet. Det overordnede tema for semesteret er sandtidskommunikationssystemer.

Projektet er lavet i samarbejde med Tele Danmark repræsenteret ved Jens Jakobsen. Projektgruppen takker Tele Danmark for samarbejdet og lån af udstyr og dokumentation.

Rapporten henvender sig hovedsageligt til vejledere, censor samt senere studerende på datateknik.

Baggrundsviden, såsom DTMF-dekodning, RTP/RTCP og CAPI, er at finde i appendiks. Rapporten (appendiks untaget) anbefales læst i kronologisk rækkefølge. En studierapport indeholdende overvejelser angående semesteret som læreproces indgår i rapporten.

Aalborg, 17. december 1999.

Claus Albøge

Lars Jochumsen Kristensen

Tonny Gregersen

Mads Græsbøll Christensen

Karsten Jensen

Peter Korsgaard

Robert Stepien

Indhold

Kapitel 1

Indledning

I en tid, der af mange betegnes som informationsalderen, kommer der dagligt nye tiltag til at udbrede information og skabe kommunikation mellem mennesker verden over. Og netop inden for kommunikation foregår der en rivende udvikling. Et af de sidste skud på stammen er IP telefoni / Voice over IP.
Som det ligger i ordene, drejer det sig om en ny indgangsvinkel til telefoni, eller rettere, en ny måde at anvende det eksisterende telefonnet på. Hidtil har telefonsamtaler bestået af reservation af en linie hele vejen mellem de implicerede parter; men efter Internettets større og større betydning og udbredelse blandt den almene borger, er man begyndt at se på Internettets opbygning og de teknologier, der ligger bag, med henblik på at anvende disse til telefoni.
Alt dette har medført et stort arbejde med udvikling og design af standarder og protokoller til brug ved Voice over IP. Pt. udvikles og arbejdes der stadig på ovenstående. Der er efterhånden udarbejdet en tilpas mængde information, der gør det muligt at implementere Voice over IP systemer.
Netop i dette tages udgangspunktet til nærværende projekt. I samarbejde med Tele Danmark udvikles et IP-telefoni system, der skal gøre det muligt at implementere IP-telefoner i et "fremtidshus". Systemet skal i hovedtræk gøre det muligt at benytte "fremtidshusets" Internetforbindelse til, udover almindelig Internet-forbrug, at telefonere. Således skal det være muligt at telefonere, både internt via husets lokal net (LAN) og eksternt via opkoblingen til sin Internet udbyder. Fordelen ved eksterne opkald ligger i, at man kun betaler for forbindelsen til sin Internetudbyder.
Det bør nævnes, at projektet skal ses som et pionerprojekt, i den forstand at der arbejdes med helt nye teknologier, der stadig er under udvikling.

Kapitel 2

Kravspecifikation

2.1 Indledning

Et realtidssystem til håndtering af PSTN-telefonsamtaler fra et IP-baseret LAN-netværk til et antal ISDN-linier udvikles.
Projektet har taget sit udgangspunkt i et projektforslag, stillet af Jens Jakobsen fra Tele Danmark. Denne respræsenterer også TeleDanmark.

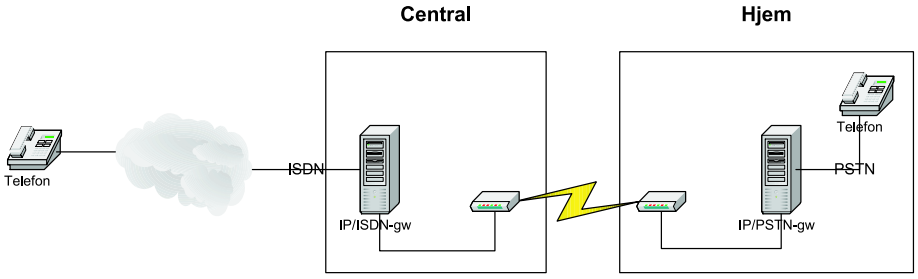
Produktnavn: VoIP gateway og terminal.

Systemet tænkes at indgå i et Tele Danmark-projekt omhandlende opbygningen af et "fremtidshus", hvori et LAN i hjemmet indgår.
De formelle rammer for projektet udgøres af studieordning for datateknik, 5. semester.

2.2 Generel beskrivelse

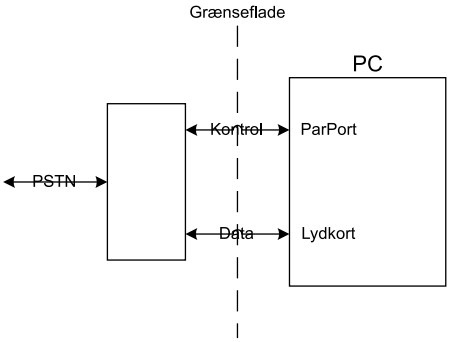
2.2.1 Systembeskrivelse

Generelt kan projektet beskrives som omhandlende en specifik anvendelse af et LAN i hjemmet, nærmere bestemt Voice over IP (VoIP).
Udviklingen sker med baggrund i eksisterende og anvendte teknologier, som f.eks. PSTN. Der er altså ikke tale om en udskiftning af eksisterende systemer, som f.eks. en IP-telefon ville være.
Systemet består af en IP/ISDN-gateway og en IP/PSTN-gateway. Software til begge udvikles.
Et antal IP/PSTN-gateway'er placeres i hjemmet, forbundet til LAN'et. PSTN-telefoner tilsluttes gateway'erne. En ADSL, fiber-linie eller lignende forbinder LAN'et til en central, hvor en IP/ISDN-gateway er placeret. Herfra føres opkaldene videre til et antal ISDN-linier, der forbinder systemet med det eksisterende telefonnet. Dette ses på figur 2.1.
Den eksterne grænseflade, der viser, hvordan PSTN-telefonen grænser op til



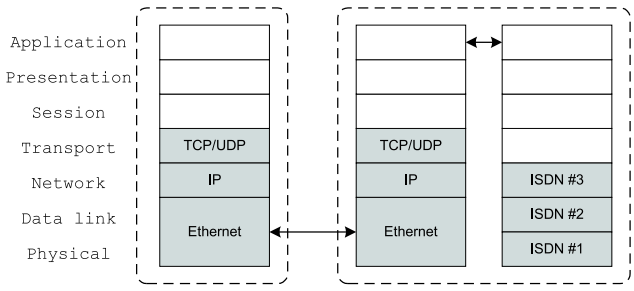
Figur 2.1: Systembeskrivelse. IP/PSTN- og IP/ISDN-gateway'erne udvikles.

PSTN/IP-gateway'en, ses på figur 2.2.



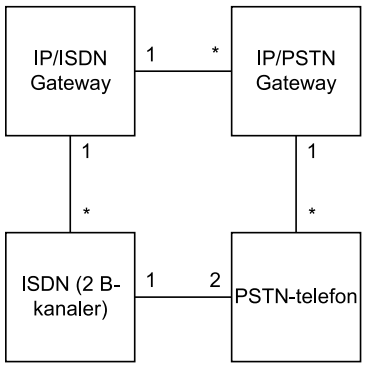
Figur 2.2: Ekstern grænseflade mod PSTN-telefon.

Alt, til venstre for den stiplede linie, udvikles ikke, men grænsefladen specificeres.
Denne afgrænsning skal ses som et udtryk for, at udviklingen af hardware ligger uden for temaet for datateknik, 5. semester.
Figur 2.3 viser miljøet, systemet skal indgå i, repræsenteret ved OSI-stak referencemodellen.



Figur 2.3: OSI-stak repræsentation af omgivelserne, systemet skal indgå i.

Figur 2.4 viser forholdet mellem antallet af de forskellige gateway'er, telefoner og BRI ISDN-linier.



Figur 2.4: Forhold mellem antal gateway'er, telefoner og linier.

Systemet skal altså være skalerbart, med nævnte forhold overholdt. Det vil være begrænset af ressourcerne, eksemplevis antallet af IRQ, DMA-kanaler, PCI-slots og CPU-kraft på de anvendte computere og linien mellem centralen og hjemmet.

2.2.2 Funktionalitet

Data opsamles fra PSTN-telefoner ved hjælp af en konverteringsenhed, der kontroleres via en standard parallelport og overfører data til PSTN/IP gatewayen via lydkort. Disse overføres via et IP-baseret netværk i henhold til H.323 for kommunikation over pakkebaserede netværk, der ikke nødvendigvis garanterer QoS (Quality of Service) til en IP/ISDN-gateway, der foretager nødvendig omformattering (til G.711) af data og sender dem over et antal ISDN linier, der tilgåes igennem CAPI 2.0 (Common ISDN Application Programming Interface). Systemet skal håndtere opkaldsprocedurer, såvel indgående som udgående.

2.2.3 Begrænsninger

Det fremstillede system har en række begrænsninger:

- Systemet vil have karakter af en prototype
- Support for multipoint-konference (H.323) udvikles ikke
- Support for video-telefoni (H.320) udvikles ikke
- Kun et begrænset antal supplerende tjenester udvikles

2.2.4 Brugerprofil

Brugerne af VoIP-systemet er dels brugere af analog telefoni (PSTN), og dels teknikere som skal intallere, konfigurere og vedligeholde systemet.

2.2.5 Krav til udviklingsforløbet

Værktøj

Systemet udvikles i ANSI C++, og MS Visual C++ udviklingsmiljøet anvendes. Rapport og anden dokumentation skrives i LaTeX. Til revisionsstyring bruges CVS. UML-notation anvendes.

Metode

Som udviklingsmetode bruges primært dele af CODARTS (Concurrent Design Approach for Real-Time Systems) og COBRA (Concurrent Object-Based Realtime Analysis). SPU (Struktureret Program Udvikling) bruges til udarbejdelse af kravspecifikationen. I projektets implementationsfase laves der integrationstest. Det samlede system testes ved en accepttest.

Standarder

Til ISDN-kommunikation vil vi benytte følgende standarder:

- CAPI 2.0, eller
- Q.931 (Digital subscriber signalling system no. 1 (DSS 1) - ISDN user network interface layer 3 specification for basic call control), afhængigt af om OS'et understøtter CAPI
- G.711 (PCM Pulse code modulation of voice frequencies)

Til VoIP-kommunikation vil vi benytte følgende standarder/anbefalinger:

- H.323 (Packet-based multimedia communikation)
- H.245 (Control protocol for multimedia communication)
- H.225 (Call signalling protocols and media stream packetization for packet based multimedia communication)
- H.246 (Interworking of H-series multimedia terminals with H-series multimedia terminals and voice/voiceband terminals on GSTN and ISDN)
- G.711 (PCM Pulse code modulation of voice frequencies)
- TCP/IP-protokolstakken
- RTP/RTCP

2.2.6 Omfang af kundeleverence

Der vil blive leveret en rapport samt den udviklede software. Rapporten vil blive leveret ved projektaflevering. Softwaren vil være tilgænglig til download på projektgruppens hjemmeside, http://www.kom.auc.dk/~jacmet/d5, i perioden fra afleveringsdato til eksaminationsdato.

2.2.7 Forudsætninger

Det forudsættets, at den for udviklingen nødvendige hardware stilles til rådighed af kunden:

- En eller flere BRI ISDN-linier
- En eller flere PSTN-telefoner
- En ISDN-telefon (til test)
- Diverse kabler
- To PC'ere

Til PC'erne skal følgende kort være til stede:

- CAPI 2.0 ISDN-kort
- Full Duplex-lydkort
- IEEE 802.3-kompatibelt netværkskort

Med forholdene beskrevet i figur 2.4 overholdt.

2.3 Specifikke krav

2.3.1 Obligatoriske krav

Systemet skal opfylde følgende krav:

- Systemet skal være transparent for PSTN-brugeren (ikke understøttede tjenester dog undtaget.)
- Skalerbart antal forbindelser, som beskrevet i 2.2.1
- De i 2.2.5 noterede standarder anvendes
- Frekvensområde: $300Hz$ - $3,1kHz$
- Realtidsafvikling

Desuden udvikles en minimal klient (H.323-terminal) til test af systemet.

2.3.2 Optionelle krav

Følgende krav skal overvejes i designet og muligvis realiseres:

- Kryptering af data i henhold til H.235 (Security and encryption for H-series multimedia terminals)
- Benyttelse af PRI (Primary Rate Interface) ISDN
- Højere komprimering af lyddata efter en af følgende standarder:

  - G.722 (7 kHz audio coding within 64 kbit/s)
  - G.723.1 (Speech coders: Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s)
  - G.726
  - G.728 (Coding of speech at 16 kbit/s using low-delay code exited linear prediction)
  - G.729 (Coding of speech at 8 kbit/s using conjugate structure algebraic code exited linear prediction (CS-ACELP))

- Supplerende tjenester
  - Vis/skjul nummer
  - Spærring

2.3.3 Fremtidige udvidelsesmuligheder

En række fremtidige udvidelsesmuligheder ses i systemet. Disse vil ikke indgå i designet.

- Andre supplerende tjenester
- GUI-konfigurationsværktøj

2.4 Eksterne grænseflade-krav

2.4.1 Krav til netværk

Nuværende standard IP-version benyttes (IPv4).
Systemet kræver minimum effektiv båndbredde på 104kb/sec pr. PSTN linie. (Se 4.3) For at systemet skal kunne indgå i et eksisterende LAN (Tele Danmarks fremtidshus), sættes der ikke krav om, at netværket er dedikeret til til systemet. Dog skal netværket dimensioneres efter, den øvrige brug, således af der til hver en tid er den nødvendige båndbredde tilrådighed for dette system.

2.4.2 Hardware-grænseflade

Systemets eksterne hardware-grænseflade består af:

- En til flere BRI (Basic Rate Interface) ISDN linier til telefoni (i gatewaysiden).
- En til flere standard parallelporte (SPP) samt stereo full-duplex audio forbindelser til PSTN-styringsenheden (i terminalsiden). Se figur 2.2 samt specifikation forneden.
- Mellem terminal og gateway består hardware-grænsefladen af et IP-baseret WAN/LAN.

Specifikation af hardware-grænsefladen til PSTN-styringsenheden

Grænsefladen består af en analog og en digital del. Den analoge del står for overførsel af lyd (herunder tale og DTMF-toner) til/fra PSTN-enhederne. Den digitale del står for at overføre information om ringetonen til PSTN-enhederne og information om rørets tilstand fra PSTN-enhederne. Se figur 2.2.

Analog grænseflade

Den analoge grænseflade udnytter indgangene og udgangene i et full-duplex stereolydkort. Eet lydkort kan således bruges til to PSTN-enheder. Antallet af lydkort i PC'en er begrænset af antallet af ledige slots, IRQ- og DMA-kanaler. Som udgang fra lydkortet skal højttalerudgangen (speaker out) eller linieudgangen (line out) benyttes. Som indgang til lydkortet skal linieindgangen (line in) benyttes. Impedansforhold og spændingsniveauer skal tilpasses således, at de passer til alle populære lydkort.

Digital grænseflade

En standard-parallelport (SPP) benyttes. Da denne kun har fem dataindgange, er antallet af PSTN-enheder begrænset til fem per parallelport. Der kan maksimalt benyttes tre parallelporte i en PC.
Standard-parallelporten har ikke en defineret elektrisk specifikation I IEEE 1284 standarden for bidirektionelle parallelporte kræves det, at man ikke belaster parallelportens udgange med mere end $20\mu A$. Det anbefales, at andre krav fra IEEE 1284 også følges.

Parallelporten følger TTL-spændingsniveauer:

- Et ben, der er elektrisk højt, har en spænding på over $2,4V$.
- Et ben, der er elektrisk lavt, har en spænding på under $0,8V$.

Output af ringetone foregår ved at parallelporten sætter databenene til det rigtige niveau. Et elektrisk højt databen betyder at den tilhørende PSTN-enhed skal ringe uden afbrydelse. Et elektrisk lavt databen betyder at den tilhørende PSTN-enhed ikke skal ringe. Se figur 2.5.
Input af telefonrørets tilstand foregår ved at sætte inputbenene (ben 10, 11, 12, 13 og 15) til det rigtige niveau. Et elektrisk højt inputben betyder at den tilhørende PSTN-enheds rør er løftet. Et elektrisk lavt inputben betyder at den tilhørende PSTN-enheds rør er lagt på. Se figur 2.5.

2.4.3 Kommunikations- og softwaregrænseflade

Kommunikations-grænsefladenerne mellem systemets bestanddele opbygges med udgangspunkt i OSI-stakken, som vist på figur 2.3. TCP/IP-protokolstakken anvendes.

DB25 ben	Signalnavn	Beskrivelse
5	Data 3	Outputdata til PSTN-enhed 1
6	Data 4	Outputdata til PSTN-enhed 2
7	Data 5	Outputdata til PSTN-enhed 3
8	Data 6	Outputdata til PSTN-enhed 4
9	Data 7	Outputdata til PSTN-enhed 5
15	/Error	Inputdata fra PSTN-enhed 1
13	SelectIn	Inputdata fra PSTN-enhed 2
12	PaperEnd	Inputdata fra PSTN-enhed 3
10	/Ack	Inputdata fra PSTN-enhed 4
11	Busy	Inputdata fra PSTN-enhed 5
18-25	Ground	Signal jord

Figur 2.5: Parallelport-grænseflade til maks. fem PSTN-enheder.

Systemet skal afvikles på Windows NT 4.0. Socket API'en anvendes til netværksprogrammeringen. Der benyttes CAPI 2.0 til interfacing af ISDN.

2.4.4 Brugergrænseflade

Ingen brugergrænseflade til konfigurering af gateway'en udvikles. Konfigureringen foregår i en tekstfil.

2.5 Kvalitetsfaktorer

Følgende afvejning af kvalitetsfaktorer udgør en vejledende prioritering for abejdet under analyse, design og implementation:

Pålidelighed: Pålideligheden af produktet er meget vigtig, idet stabilitet og høj oppe-tid prioriteres højt for gateway systemer.

Vedligeholdelsesvenlighed: Vedligeholdelsesvenlighed vægtes ikke højt, dog skal indkapsling samt god dokumentation sikre mulighed for hurtig tilpasning og eventuel ændring af fejl.

Udvidelsesvenlighed: Systemet er dedikeret til det aktuelle formål, hvorfor udvidelsesvenlighed ikke er vigtig, dog bør systemet være skalerbart, således af flere PSTN- og ISDN-linier kan tilsluttes.

Brugervenlighed: Brugervenligheden, i forhold til administration, er mindre vigtig, da bruger-interaktion er minimal. Derfor foregår al konfiguration af gateway'en gennem en tekst-fil, som beskrevet i afsnit 2.4.4. Til normalt brug, skal systemet være transparent for brugerne.

Genbrugbarhed: Da systemet består af både en IP/ISDN-gateway og IP/PSTN-gateway er det vigtigt at softwaren udvikles og opbygges på en sådan måde at den kan genbruges i begge systemer.

Integritet: Det er vigtigt at systemet har kort opstartstid efter nedbrud, strømsvigt etc.

Effektivitet: Den vigtigste kvalitet, systemet skal besidde er effektivitet. Systemet skal have en ydeevne, der sikrer at tale kan foregå som specificeret i afsnit 2.3.1.

Kapitel 3

ITU-T Anbefaling H.323

3.1 Indledning

H.323 er en anbefaling for terminaler og andre enheder, der tilbyder tjenester for multimedia-kommunikationer over et pakkebaseret netværk, som ikke nødvendigvis tilbyder en garanteret QoS. Ved multimedia forstås audio, video og data mellem to eller flere H.323 enheder. Understøttelse af audio er obligatorisk, mens understøttelse af video og data er optionelt. Det pakkebaserede netværk, over hvilket H.323 enheder kommunikerer, kan være en point-to-point-forbindelse, et enkelt netværkssegment eller et internetværk bestående af flere segmenter med komplekse topologier. Anbefalingen omfatter ikke netværksgrænsefladen, det fysiske netværk eller den transportprotokol, der anvendes på netværket. I det følgende gives en oversigt over de dele af H.323, der er relevante for dette projekt.

3.2 Opbygning

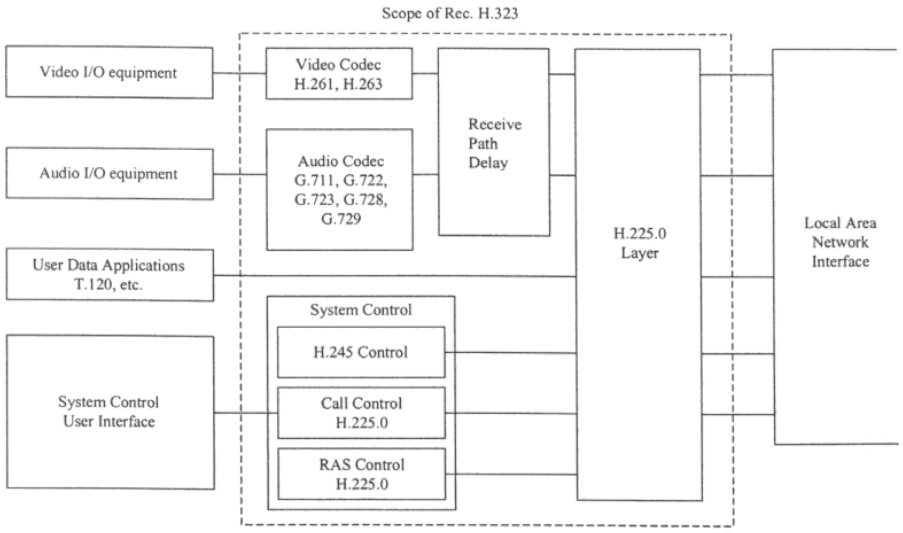
Et H.323 system kan bestå af følgende enheder:

- Terminal: Enhed, der understøtter transmission og modtagelse af audio, video (optionel) og data (optionel)
- Gateway: Enhed, der understøtter mapning mellem et pakkebaseret og et kredsløbskoblet netværk, kaldsopsætning og clearing på begge sider og oversættelse mellem komprimeringsstandarder (optionel)
- Gatekeeper: Optionel enhed der, hvis den er repræsenteret i et system, skal tilbyde følgende tjenester: Adresseoversættelse, adgangskontrol for netværket, båndbreddekontrol og zone-management (gatekeeperen skal tilbyde de tre førstnævnte tjenester for terminaler, MCU'er (Multi Control Units), og gateway'er, der er registreret hos gatekeeperen). Gatekeeperen kan endvidere håndtere en række optionelle funktioner, eksempelvis kaldsautorisation og båndbredde-management

- Multipoint controller (MC) og MCU: Optionelle enheder, der anvendes i forbindelse med multipoint-forbindelser (konferencer)

3.2.1 Terminal

Figur 3.1 viser et eksempel på en H.323-terminal. Diagrammet viser video-koder/dekoder, audio-koder/dekoder, H.225.0-lag, systemkontrol-funktioner og grænseflader til brugerudstyr og til det pakkebaserede netværk. Alle H.323 terminaler skal indeholde et H.225.0-lag, en systemkontrol-enhed og en audio-koder/dekoder, hvorimod video-koder/dekoder og bruger-applikationsdelen er optionelle.



Figur 3.1: H.323 terminal

Grænseflade til pakkebaseret netværk

Grænsefladen til det pakkebaserede netværk er implementations-specifik og er ikke omfattet af denne anbefaling. Netværksgrænsefladen skal dog tilbyde de tjenester, der er beskrevet i Anbefaling H.225.0 (beskrevet nedenfor). Disse inkluderer følgende: Pålidelig (eksempelvis TCP) end-to-end-tjeneste er obligatorisk for H.245-kontrolkanalen (beskrevet nedenfor), datakanalerne og kanalerne til kaldsopsætning. Ikke-pålidelig (eksempelvis UDP) end-to-end-tjeneste er obligatorisk for Audio-kanalerne, Video-kanalerne og RAS (Registration, Admission and Status)-kanalerne.

Audio-koder/dekoder

Alle H.323-terminaler skal have en audio-koder/dekoder, der skal kunne kode og dekode lyd i henhold til Anbefaling G.711 og alle terminaler skal kunne sende

og modtage A-law og μ-law. Understøttelse af kodning og dekodning i henhold til andre anbefalinger er optionel.

H.225

H.323-endepunkter anvender to signalkanaler til kaldskontrol og til funktioner relateret til gatekeeperen. Formatet der bruges på disse kanaler skal følge Anbefaling H.225.0. Anbefaling H.225.0 omfatter kommunikation mellem H.323-enheder på det samme pakkebaserede netværk, der anvender samme transportprotokol. Formålet med denne anbefaling er at tilbyde redskaber til synkronisering af pakker, der anvender de underliggende pakkebaserede transportfaciliteter. Anbefalingen anvender RTP/RTCP til synkronisering af datapakker for alle underliggende pakkebaserede netværk. Anbefalingen indeholder en detaljeret beskrivelse af anvendelsen af RTP/RTCP. Implementationen af H.225.0 skal følge Anbefaling Q.931 (ISDN user-network interface layer 3 specification for basic call control).

H.245

Anbefaling H.245 definerer en række procedurer, der tillader udveksling af, og angiver syntaks og semantik for, kontrolinformation vedrørende følgende punkter:

- Master/slave-bestemmelse
- Forhandling om hvilke resourcer der understøttes
- Logisk kanalsignalering
- Round-trip-delay-bestemmelse

H.245-beskederne falder i fire kategorier: Request, Response, Command og Indication. Request-beskederne anmoder om en specifik handling hos modtageren, deriblandt et øjeblikkeligt svar. Response-beskederne svarer på en tilsvarende Request-besked. Command-beskeder anmoder om en specifik handling, men kræver ikke noget svar. Indication-beskeder er informative og kræver ikke nogen handling eller noget svar.

3.2.2 Gateway

En H.323 gateway håndterer den nødvendige oversættelse mellem transmissionsformater og kommunikationsprocedurer (kontrolsignaler). Denne oversættelse er specificeret i anbefaling H.246 (under stadig udarbejdelse). Desuden skal gateway'en håndtere kaldsopsætning og clearing af forbindelser på både de pakkebaserede og kredsløbskoblede netværk. Oversættelse mellem video-, audio- og dataformater kan også udføres i gateway'en. En gateway kan optræde som en terminal eller MCU på H.323 nettet, alt efter producentens ønske. Man kan ændre status under et kald ved hjælp af H.245. Endelig kan gateway'en optræde som multiple H.323 terminaler (eksempelvis 1 per udgående linje). Generelt

er hensigten med gateway'en, når den ikke opererer som MCU, at repræsentere karakteristikkerne for et endpoint på det pakkebaserede netværk for et endpoint på det kredsløbskoblede netværk og omvendt, på en måde som er transparant. En gateway kan være forbundet med en anden gateway via det kredsløbskoblede netværk for at tilbyde kommunikation mellem H.323-terminaler, der ikke er på det samme netværk.

3.2.3 Gatekeeper

Gatekeeper'en, der er optionel i et H.323-system, tilbyder kaldskontrol-tjenester til H.323-endepunkterne. Når en gatekeeper er repræsenteret i et system skal den tilbyde de følgende tjenester:

- Adresseoversættelse
- Adgangskontrol
- Båndbreddekontrol
- Zone kontrol

Derudover kan den udføre en række optionelle funktioner.

Adresseoversættelse

Gatekeeper'en skal skal have en funktion til at oversætte mellem et alias og den fysiske netværksadresse. Man kan vælge at bruge forskellige adresser i et H.323 system, eksempelvis findes der en standard, E.164, for nummerering (The international public telecommunication numbering plan).

Adgangskontrol

Gatekeeper'en skal foretage adgangskontrol ved hjælp af H.225 ARQ/ACF/ARJ (Admission Request/Admission Confirmation/Admission Reject)-beskeder.

Båndbreddekontrol

Gatekeeper'en skal understøtte båndbreddekontrol via BRQ/BRJ/BCF (Bandwidth Change Request/Bandwidth Change Reject/Bandwidth Change Confirmation)-beskeder. Funktionen kan dog også være en dummy, der accepterer alle anmodninger.

Zone kontrol

Gaekeeper'en skal stille ovennævnte funktionalitet til rådighed for terminaler, gateway'er og MCU'er der er registreret hos gatekeeper'en.

3.3 Procedurer for kaldssignalering

Fremgangsmåden ved kommunikation mellem to endepunkter er som følger:

- Fase A: Kaldsopsætning
- Fase B: Indledende kommunikation og forhandling om understøttelse af resourcer
- Fase C: Etablering af audiovisuel kommunikation
- Fase D: Kaldstjenester
- Fase E: Kaldsterminering

3.3.1 Kaldsopsætning

Kaldsopsætning finder sted ved hjælp af de beskeder til kaldskontrol, der er defineret i Anbefaling H.225.0. Forespørgsler om reservering af båndbredde bør finde sted så hurtigt som muligt.
Der er ingen eksplicit synkronisering eller locking mellem to endepunkter under kaldsopsætningen. Det er derfor op til applikationen at håndtere problematikker vedrørende samtidighed.
Et endepunkt skal kunne sende en Alerting-besked. Alerting-beskeden indikerer, at den kaldte part er blevet varskoet (alerted) om et indkommende kald. I det tilfælde, hvor der kommunikeres igennem en gateway, skal gateway'en sende alerting-beskeden, når den modtager en ringe-indikation fra det kredsløbskoblede netværk. Hvis et endepunkt kan svare på en Setup-besked med en Connect-, Call Proceeding-, eller Release Complete-besked inden 4 sekunder, kræves det ikke, at det sender en Alerting-besked. Et endepunkt, der sender en Setup-besked kan altså forvente at modtage en af ovennævnte beskeder indenfor 4 sekunder.
For at bevare konsistens i betydningen af Connect-beskeden mellem pakkebaserede netværk og kredsløbskoblede nedtværk, bør en Connect-besked kun sendes, når det er sikkert at forhandlingen om understøttelse af resourcer (H.245) føres til ende med succes, og et minimum af kommunikation vil finde sted.
H.323 definerer en særlig Fast Connect Procedure, der muliggør en kaldsopsætning, der ikke gør brug af de procedurer, der er defineret i Anbefaling H.245. Det kaldte endepunkt har dog mulighed for at afvise at bruge denne procedure, enten fordi det ikke har implementeret den (den er optionel) eller fordi det ønsker at anvende nogle af egenskaberne i de procedurer, der er defineret i Anbefaling H.245. En anden mulighed er, at et endepunkt, efter at have indledt eller accepteret Fast Connect Proceduren, nu ønsker at skifte til H.245-procedurerne. H.323 definerer også hvordan et sådant skifte kan udføres.
Når en ekstern terminal kalder et endepunkt på netværket gennem en gateway, foregår udvekslingen af beskeder mellem gateway og endepunkt som mellem to endepunkter. Det samme er tilfældet, når et endepunkt på netværket kalder en ekstern terminal gennem gateway'en. I dette tilfælde bør en gateway sende en

Call Proceeding-besked, hvis den ikke forventer at kunne svare med en Alerting-, Connect-, eller Release Complete-besked inden 4 sekunder.

3.3.2 Indledende kommunikation og forhandling om understøttelse af resourcer

Når begge sider har udvekslet beskeder vedrørende kaldsopsætning, skal de etablere H.245-kontrolkanalen. De i Anbefaling H.245 definerede procedurer anvendes over H.245-kontrolkanalen til forhandling om understøttelse af resourcer og til at åbne medie-kanalerne. Med henblik på at spare resourcer, synkronisere kaldsopsætning og reducere tiden, der medgår til kaldsopsætning, definerer H.323 hvordan man kan indkapsle H.245-beskeder i de Q-931-beskeder, der i henhold til Anbefaling H.225 anvendes til kaldsopsætning. Ligesom ved Fast Connect Proceduren, er det også her defineret, hvordan man kan skifte til en separat H.245-forbindelse på et hvilket som helst tidspunkt.

3.3.3 Etablering af audiovisuel kommunikation

Efter den indledende kommunikation har fastlagt master/slave-forholdet og forhandlingen om understøttelse af resourcer er bragt til ende, skal de logiske kanaler, der skal anvendes til overførsel af informations-streams, åbnes. Audio- og video-streams overføres ved hjælp af en ikke-pålidelig protokol, mens overførsel af data håndteres ved hjælp af en pålidelig protokol.

3.3.4 Kaldstjenester

H.323 definerer en række tjenester det er muligt at implementere ved hjælp af en gatekeeper.

3.3.5 Kaldsterminering

Begge endepunkter kan terminere et audio-kald ved hjælp af følgende procedure:

1. Det bør afbryde transmissionen af audio og dernæst lukke alle logiske kanaler for audio.
2. Det skal sende en H.245-**endSessionCommand**, for at indikere til det andet endepunkt, at det ønsker at afbryde kaldet, og dernæst at afslutte transmissionen af H.245-beskeder.
3. Det skal vente på at modtage **endSessionCommand** fra det andet endepunkt, og skal dernæst lukke H.245-kontrolkanalen.
4. Hvis kanalen til kaldssignalering er åben skal en Release Complete-besked sendes, og kanalen skal lukkes.
5. Hvis endepunktet er registreret hos en gatekeeper skal kaldet afsluttes hos gatekeeperen med en DRQ (Disengage Request)-besked til gatekeeperen, der svarer med en DCF (Disengage Confirmation)-besked.

Et endepunkt, der modtager **endSessionCommand** uden først at have sendt den, skal udføre trin 1-5 i ovenstående procedure, bortset fra at det ikke skal vente på **endSessionCommand** i trin 3.

3.3.6 Protokol-fejlhåndtering

Hvis en protokolfejl rapporteres fra den underliggende pålidelige protokol, skal H.245-kontrolkanalen og alle, dermed forbundne, logiske kanaler lukkes, ifølge proceduren for kaldsterminering, som om det andet endepunkt har sendt en **endSessionCommand**. Kanalen for kaldssignalering anvender også en pålidelig transportprotokol. Hvis en protokolfejl rapporteres på denne kanal, kan endepunktet eller gatekeeperen forsøge at genetablere kaldssignalerings-kanalen. Hvis et endepunkt ønsker at bestemme, hvorvidt det andet endepunkt stadig fungerer og har forbindelse, kan det sende en H.245-**roundTripDelayRequest**.

Kapitel 4

Netværksanalyse

Formålet med dette kapitel er at undersøge på hvilke områder IP-netværket har indflydelse på systemet. Der beskrives bl.a. hvilke mekanismer, der kan bruges for at udnytte netværket optimalt i forhold til systemet.

4.1 Netværk

Systemet skal indgå som del af et eksisterende lokal netværk, det betyder, at der skal tages hensyn til forskellige krav. Der skal sikres båndbredde, først og fremmest til Voice over IP systemet, men også til de andre services på lokalnettet.
Da Internettet og intranet er opbygget som pakkebaserede netværk, kan man ikke umiddelbart forudsige hvilken vej gennem nettet en pakke transporteres, ej heller, hvornår pakken når sin destination; hvis den i det hele taget kommer frem.
Til almindelig data-overførsel er dette ikke et større problem, sålænge det foregår indenfor et acceptabelt tidsrum. Derimod er det et langt større problem, når det drejer sig om overførsel af realtids-data, f.eks. multi-media; idet man her behøver en garanti for at data når frem i rette tid og rækkefølge.
En af de mest anvendte løsninger på ovennævnte problematik, er at klassificere trafik, og på den måde prioritere een slags trafik højere end en anden, ved at allokere og reservere båndbredde.
Det kan gøres på flere måder, bl.a. ved brug af specielle reservationsprotokoller.

- RSVP: Resource Reservation Protocol
- SNMP: Simple Network Management Protocol

4.1.1 RSVP

RSVP er en netværks-kontrol-protokol, der bruges til at reservere nødvendige resourcer for en pakke, gennem et netværk, således at den forespurgte båndbredde er til rådighed, når pakken sendes.

4.1.2 SNMP

SNMP er en protokol til netværks-styring, der henvender sig til opsætning og administration af f.eks switche og routere.

4.1.3 Trafik klassificering

En anden måde at styre netværks-resourcer på er ved klassificering af trafik, f.eks. ved brug af ToS/QoS.

4.1.4 Type of Service

Type of Service er, som det ligger i navnet, forskellige service-typer. Disse typer bliver brugt til at specificere, hvordan pakker med data sendes gennem et netværk.
Formålet med ToS er kort og godt at give IP-pakker forskellige prioriteter. Det vil sige, at pakker med en ToS f.eks sendes den hurtigste vej gennem et netværk, mens et andet datagram med en anden ToS sendes den sikreste vej.
Implementationen af ToS i IPv4 optager 8 bit i IP headeren, hvoraf bit 3-6 er af størst betydning, da det er her de 5 forskellige ToS sættes. De 4 ToS bits er organiseret på følgende måde:

Bitmask	**Type of Service**
1000	minimize delay
0100	maximize throughput
0010	maximize reliability
0001	minimize monetary cost
0000	normal service

Figur 4.1: Bitmaske tabel for ToS bit i IP header.

Det er vigtigt at understrege at de forskellige ToS ikke er garantier for den service-type, de står for, men en form for prioritet. For at opnå garantier for services, skal man istedet anvende Quality of Service, se afsnit 4.1.6.

4.1.5 ToS og VoIP

I forbindelse med vores system, er det vigtigt at se i hvilke sammenhænge ToS bruges. Det primære formål med ToS er at finde den rigtige vej gennem et netværk for en IP-pakke. Her er det typisk de routere netværket er opbygget omkring, der bestemmer, hvordan pakken kommer frem. Dermed er det altså routernes opgave at finde den optimale rute, udfra ToS og destination for et IP-datagram. I vores tilfælde har vi ikke et stort og komplekst netværk, med mange forskellige ruter, hvorfor ToS/QoS udelukkende vil blive brugt mht. schedulering af indkommende og udgående pakker på et netværksinterface. Brug af ToS/QoS i systemet må dog afhænge af den pågældende support herfor i det valgte OS.

Winsock 1.x biblioteket til WIN32, stiller funktioner tilrådighed til at sætte ToS bit i IP headeren.

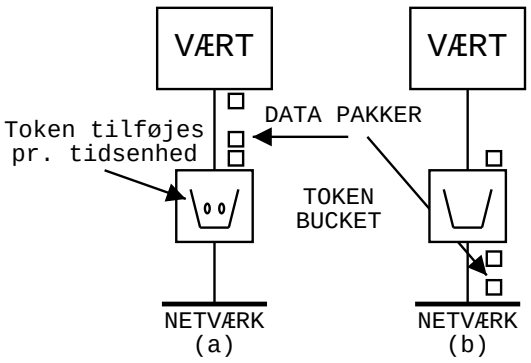
4.1.6 Quality of Service

Quality of Service bruges kort sagt til at bestemme under hvilke kvaliteter en IP-pakke skal/kan sendes gennem et netværk. Med QoS er det f.eks. muligt at give garantier for, hvor stort det maksimale delay ved forsendelse af en IP-pakke må være.
Når det drejer sig om VoIP og Video on Demand, er det vigtigt at have lavt delay, og netop til det formål bør man anvende QoS. Det delay, der forekommer under transmitteringen af en IP pakke, kan opdeles i to områder. Et fast delay for for selve transmitteringen, afhængig af den valgte vej gennem netværket og et delay i forbindelse med (fifo) køer gennem routere, bridges, gateways osv. Det faste delay har på sin vis ikke noget med QoS at gøre, da det bestemmes af den tid det tager at sende en pakke "uhindret" gennem et netværk. Derimod er det delay, der forekommer, når en pakke venter i en kø i f.eks. en router, noget der kan kontrolleres med QoS. En af de væsentligste grunde til disse køer er, at data ofte sendes "bursty", altså i klumper, der kan ophobe sig i dele af netværket. For at komme dette problem til livs er det nødvendigt at se på Congestion Control, f.eks implementeret ved Token Bucket princippet. Congestion Control giver mulighed for at kontrollere netværksressourcerne, på den måde at man kan nedsætte / undgå køtider på netværksenheder, og derved minimere delay.

4.1.7 QoS og VoIP

QoS kan implementeres på flere måder, bl.a ved kontrollering af de underliggende routnings mekanismer. Senest (aug 99) er der eksperimentiel udvikling af en QoS routnings mekanisme [Apostolopoulos, 1999], der skal bruges til at finde og beregne QoS ruter gennem et netværk.
Mere relevant er QoS implementeret som Congestion Control, da winsock 2 biblioteket til WIN32 stiller QoS funktioner til rådighed, der baserer sig på Token Bucket princippet. Token Bucket princippet stammer fra Leaky Bucket princippet. Leaky Bucket skal som navnet antyder opfattes som en spand med et hul i bunden. Hullet gør, at kun en bestemt mængde vand kan forlade spanden per tidsenhed, det vil sige at man har kontrol over, hvor meget vand der forlader spanden, idet man kender hullets størrelse. Skulle der fyldes mere vand i spanden, end der er kapacitet til, vil det der løber over kanten betragtes som tab. Analogien til et netværk ligger i at kontrollere, hvor mange pakker der sendes ud på netværket per tid. På den måde kan man effektivt forhindre "bursts", det vil sige, tilpasse mængden af sendt data efter netværkets kapacitet. Prøver man at sende flere pakker end nettet kan klare, tabes de - og må sendes igen.
Problemet med Leaky Bucket er, at mængde af data, der må sendes på netværket per tid, er statisk, derfor har man opfundet Token Bucket princippet. Token Bucket tilføjer ideen med at ligge Tokens (billetter) i spanden, X billetter per tid. For hver billet bliver hullet i spanden åbnet i et vist tidsrum; på den måde

er det muligt at regulere den aktuelle udsendelse af pakker på netværket. Er der ingen billetter i spanden kan der ikke sendes pakker på netværket. Princippet for Token Bucket er skitseret i figur 4.1.7



Figur 4.2: Skitse af Token Bucket princippet. (a): Værten sender burst-data til netværket, gennem Token Bucket'en, der er i besiddelse af to billetter. (b) De to billetter giver adgang til netværket for to data-pakker, der sendes med et fast mellemrum.

4.2 IP

Som før nævnt er lokalnettet et IP baseret ethernet. IP protokollen er under stadig udvikling og der findes pt. to versioner: 4 og 6. IPv4 er den mest udbredte, hvorimod IPv6 stadig har karakter af en udviklingsversion, og kun implementeret få steder; som regel i test-netværk.
IPv6 har dog en række fordele fremfor IPv4, når det drejer sig om overførsel af realtids-trafik, f.eks. har man en indbygget mekanisme QoS, der gør det mulighed at etablere en højkvalitets rute gennem de lavere netværkslag for f.eks. multimedia-trafik, der kræver stor båndbredde. I IPv4 har man ikke en sådan mekanisme, men gør istedet brug af en protokol til reservation af båndbredde (RSVP), på en rute gennem et netværk, i et vist tidsrum, som omtalt i kapitel 4.1.1
Winsock2 biblioteket til win32 har support for QoS, realiseret udfra tokenbucket princippet. Mere herom i kapitel 4.1.7

4.2.1 TCP/UDP/RTP

IP udgør selve netværkslaget i TCP/IP protokolstakken. Oven på dette findes transport-laget, hvor man benytter transport-protokollerne UDP og TCP.
Oven på UDP bruges RTP. RTP står for Real-time Transport Protokol, der benyttes til transport at multimedie data, såsom audio og video sekvenser. RTP tilbyder kort fortalt, tidsstempling af data, samt en mekanisme til synkronisering af forskellige real-time data. Se appendiks A om RTP for yderligere information.

4.2.2 RTP

RTP er en realtids transport protokol, tænkt til multicast (f.eks. konferencer), men egner sig også til unicast sessions.

- RTP kan ikke selv sikre QoS garantier, men lader underlæggende lag i OSI-stakken tage sig af det.
- Brug af time-strap til synkronisering og rekonstruktion af data, gør RTP egnet til at bære realtids-data, selvom RTP i sig selv ikke garanterer dataoverførsel i realtid.
- RTP besidder pt. ikke nogen form for fejlkorrektion, mht retransmission ved pakketab. Dette begrundes med at en sådan mekanisme, afhænger af pakkens indhold, forstået på den måde at det meste realtids-data ingen værdi har efter et vist tidpunkt.
- RTP er ikke afhængig af specifikke underlæggende protokoller. Dvs. den kan bruges både på IPv4 og IPv6, samt nonIPprotokoller.

4.2.3 RTCP

RTCP bruges sammen med RTP til at "overvåge" RTP sessions. Den forhører sig så at sige om kvaliteten af data-leveringen "i den anden ende". Informationer herom, kan bl.a. bruges til justering af adaptive encodere, filtre osv.
Både IP, TCP, UDP og RTP, består af to hoveddele, en header samt en payload. Ved brugen af RTP via UDP via IP, opnås et vist overhead, idet, headerne for alle protokollerne antager en væsentlig del, af den samlede mængde data, der overføres. Den samlede mængde header-data er 40 bytes, hvilket ikke virker af meget, i forhold til den maksimale pakke-størrelse (MTU) på et ethernet, som er 1500 bytes. Dog er det ikke smart at sende realtids-trafik i så store pakker, som det belyses i 4.3.

4.3 Båndbredde

For at kunne lave overslag over den minimum krævede båndbredde, skal der først ses på, hvilket overhead der måtte være i forhold til payload kontra header. Til beregning af overhead er det nødvendigt at kende den optimale pakkestørrelse, hvormed pakkerne sendes, som passer til vores system. Denne størrelse afhænger bl.a. af den hastighed, hvormed data opsamles fra lydkort. Eksempelvis vil den krævede båndbredden for overførsel af en 20 ms, 64 kbit/s (8 kB/s) PCM audio sample, inclusiv RTP, UDP og IP header (ialt 40 bytes) svare til:

$$Bandwidth = \frac{(8kb/s \cdot 0.02s + 40bytes)}{0.02s} = 80kbit/s \tag{4.1}$$

Ligning 4.1 viser, at data sendt med et delay på 20 ms giver en pakkestørrelse på 200 bytes ($40bytes + 8kbytes \cdot 0.02sec$), hvilket medfører et overhead på:

$$\frac{40bytes}{(200-40)bytes} \cdot 100\% = 25\%$$

Det er derfor nødvendigt at finde det rette forhold mellem delay og optimal pakkestørrelse, herunder overhead forbundet med payload procentdel.
Ifølge har man til ATM netværk fundet frem til, at den optimale payload størrelse bør være mellem 32 og 64 bytes. Payload på 64 bytes giver følgende:

$$Sampletime = \frac{64bytes}{8kb/s} = 8ms$$

$$Bandwidth = \frac{(64 + 40bytes)}{0.008s} = 104kbit/s$$

$$Overhead = \frac{40bytes}{64bytes} \cdot 100\% = 62.5\%$$

Som det ses er der tæt kobling mellem båndbredde, delay, og pakkestørrelse (overhead). Med hensyn til båndbredde vil det ikke give problemer på et lokalnet, der normalt har en maksimal båndbredde på 10 Mbit/s (i nogle tilfælde 100 Mbit/s), altså omkring en faktor 100 (1000) i forhold til hvad en samtale (64 bytes payload) optager af båndbredde. Et pakkekvantiseringsdelay på 8 ms udgør, set i forhold til det, i [Eric Larson, 1999] anbefalede, maksimale delay på 250 ms for end-to-end system, omkring 3 % af det samlede delay. (De 250 ms er ifølge [Eric Larson, 1999], det maksimale delay i en samtale, som et menneske bryder sig om, før det bliver en gene)
Til disse 8 ms skal forsinkelsen på selve netværket lægges til. Da alle enheder i systemet befinder sig på samme LAN, vil et rimeligt estimat over forsinkelsen på netværket kunne foretages ved at sende data-pakker (ICMP) mellem to maskiner. Følgende stammer fra data sendt mellem to maskiner, begge med 100 Mbit netkort, på en 100 Mbit netværk.

```
104 bytes from 10.8.0.202: icmp_seq=0 ttl=128 time=0.3 ms
...
104 bytes from 10.8.0.202: icmp_seq=199 ttl=128 time=0.3 ms
104 bytes from 10.8.0.202: icmp_seq=200 ttl=128 time=0.3 ms

--- 10.8.0.202 ping statistics ---
200 packets transmitted, 200 packets received, 0\% packet loss
round-trip min/avg/max = 0.2/0.3/0.9 ms
```

Som det ses, er det gennemsnitlige round-trip (tiden fra ECHO_REQUEST sendes til ECHO_RESPONSE modtages) omkring 0.3 ms, dvs 0.15 ms for at sende 104 byte fra en maskine til en anden på et 100 Mbit lokalnetværk.

4.4 Opsummering

I ovenstående er aktuelle emner vedrørende netværk i forbindelse med VoIP analyseret. Følgende kan opsummeres: For at sikre minimalt delay gennem et netværk, med lille effektivbåndbredde, kan det være nødvendigt at anvende ToS eller QoS; winsock biblioteket giver bl.a. mulighed for at anvende QoS, implementeret ved Token Bucket princippet. Ydermere kræves der mekanismer til

sikring at korrekt overførsel af data; her er RTP en oplagt protokol, da den eksempelvis giver mulighed for synkronisering og sortering af realtids-data. Sidst har overvejelser om optimal pakkestørrelse vist at data-pakker med 64 byte payload giver et delay på mindre end 10 ms, for både pakkekvantiserings-delay'et og selve netværks-transmissionen på et 100 Mbit lokalnet.

Kapitel 5

Tidsanalyse

5.1 Indledning

Realtidssystemer, som defineret af Stankovic [Stankovic, 1992, s. 1], er systemer, hvor korrektheden af systemet ikke alene afhænger af de beregnede resultater, men også af tiden, hvor resultaterne er færdigberegnede.
Kvaliteten af en beregning afhænger altså i disse systemet ikke udelukkende af præcisionen i resultaterne, men også tiden, det tager at beregne dem. Korrekthed og ydelse er tæt koblede størrelser for i et realtidssystem.
Til realtidssystemer hører også VoIP-systemer. Dette kapitel omhandler de teoretiske overvejelser angående tidens rolle i dette system.

5.2 Realtid

Der skelnes mellem to typer realtidssystemer: bløde og hårde. Bløde realtissystemer, er systemer, hvor resultaterne efter overskreden deadline stadig har en værdi, og beregningerne herfor bør udføres. I hårde realtidssystemer har beregnede resultater ingen værdi efter deadline.
Hurtig afvikling i sig selv er ikke et krav til et realtidssystem. Den korrekte hastighed er givet ud fra krav til deadlines, de enkelte tasks skal overholde. Disse krav stammer fra omgivelserne.
At betragte et system som et realtidssystem, blødt eller hårdt, handler om anskuelse, metode til problemløsning - det er ikke en egenskab ved systemet. Alle beregninger indgår i en eller anden tidslig sammenhæng, hvad enten konsekvensen er kedsomhed eller kædereaktion i en atomreaktor.
Metoden til at vise, at et fremstillet system fungerer i hård realtidssammenhæng, går ud på matematisk at vise, at det til systemet knyttede (worst case) task-set er schedulerbart. Da dette kræver a priori viden (periode, beregningstid o.lign.) om samtlige tasks i et system, er det selvsagt en kompliceret opgave. Tasks gensidige afhængighed gør også forholdene mere komplicerede.
Endvidere har mange tasks mere eller mindre varierende beregningstid.
Man kan vælge at vise et task-sets schedulerbarhed under ideelle forhold. Det

vil for eksempel sige, at man ser bort fra tasks, man ikke har nogen kontrol over og ikke antages at bruge betydelig CPU-tid, såsom tasks, der indgår i operativsystemet.
Et andet problem ved de gængse operativsystemer er i øvrigt anvendelsen af virtuel hukommelse, der introducerer indeterminisme.
Hårde realtidssystemer er deterministiske af natur, da dette forudsættes af de analytiske metoder, der anvendes til at vise, at et task-set er schedulerbart. Realtidsscheduleringsalgoritmer er herfor også deterministiske i modsætning til de normalt anvendte scheduleringsalgoritmer i preemptive multitasking operativsystemer.
Karakteristisk for traditionelle hårde realtidssystemer er også, som en naturlig konsekvens af kompleksiteten ved visningen af schedulerbarhed, at de i høj grad er statiske, dedikerede og infleksible. Præallokering af ressourcer er et eksempel på en meget anvendt teknik, der medfører infleksibilitet og statisk struktur.
En høj grad af pålidelighed er endnu en vigtig egenskab ved realtidssystemer, og dette medfører avanceret fejlhåndtering (specielt fejlkorrektion) i disse systemer. For eksempel ett crash vil medføre overskridelse af et antal deadlines.

5.3 Karakteristik af systemet som realtidssystem

Som sagt spiller tiden en kritisk rolle i vores system. F.eks. en voldsomt forsinket telefonsamtale besværliggører, måske umuliggører, kommunikationen.
Konskvensen af en forsinket samtale eller pakketab er ikke, som i f.eks. en atomreaktor, fatale.
Der kan i dette system blive tale om at prioritere lav latens på bekosning af pakketab, da tab af enkelte datagrammer indeholdende 64 bytes ikke vil have en ødelæggende virknink på en samtale.
Netop graden af konskevensen af overskridelse af deadline er afgørende for, om et system bør anskues som værende et hårdt eller blødt realtidssystem.
Traditionelt er specielt netværksgateway'er betragtet som bløde realtidssystemer og er afviklet i ikke-realtidssystemer, såsom de fleste UNIX-varianter og Windows NT. Konsekvensen ved en overskreden deadline begrænser sig i de fleste tilfælde til irritation. Dette skyldes nok i højere grad de applikationer, man har valgt til afvikling i tidens netværk, netop grundet netværkets egenskaber. Hårde realtidssystemer baseres f.eks. ikke på nutidens internet.
Det er ikke muligt i vores system, som specificeret i kravsspecifikationen, at garantere et specifikt maksimalt delay, grundet tilstedeværelsen af et netværk imellem systemets dele, hvis tidslige egenskaber ikke er givne.
Et IPv4 netværk garanterer ikke overholdelse af tidskrav, og kan herfor kun vanskeligt betragtes som et hårdt realtidssystem.
Netværkets opførsel kan dog anskues som deterministisk ud fra betragtningen, at overførsel af datapakker på et netværk kun kan ske med en mindste periode givet netværkstypen (en art worst-case beregning set fra et scheduleringsmæssigt synspunkt), det være sig Ethernet, Token Ring eller noget helt tredje.
Et LAN i hjemmet, som dette projekt jo bygger på, vil desuden have en inde-

terministisk belastning, grundet andre enheder på nettet, der vil påvirke vores systems ydelse.
En anden måde at anskue netværkets opførsel på er at se det som en stokastisk proces og således anvende statistisk, kø-teoretisk analyse [Stankovic, 1992, s. 10].
Systemet anskues de facto af os som værende et blødt realtidssystem. De facto grundet vores mere eller mindre frivillige valg operativsystem.

5.4 Throughput og latens

To størrelser er afgørende for systemets ydelse med hensyn til audio-signalerne, der udgør en telefonsamtale:

Throughput Antal bytes der behandles per sekund.

Latens Forsinkelsen igennnem systemet.

Et højt throughput og en lav latens er ønskværdige, desværre er disse to størrelser gensidigt udelukkende: Lavere latens giver lavere throughput. Dette kan illustreres med følgende:
Jo større en buffer er, jo større bliver latensen i delen af systemet. Og for throughput: Jo større en buffer er jo mindre bliver plads- eller beregningsmæssig overhead set over et tidsrum ved f.eks. en ISR eller datagram-header, og throughput forøges herved.
Laveste throughput i en del i systemet vil være begrænsende for throughput igennem hele systemet. Latenser akkumuleres igennem systemets dele.
Disse overvejelser er afgørende for valg af buffer- og datagramstørrelser i systemet ved in- og output i forbindelse med lydkort, ISDN og sockets samt disses indbyrdes dataudveksling. Overvejelserne angående datagramstørrelse (UDP) kan findes i 4.3.
I dette system er der krav til throughput i form af krav fra G.711, og systemet dimensioneres ud fra dette. Konsekvensen af dette er en latens af en vis størrelse.

5.5 Rate Monotonic-schedulering

Et eksempel på en realtidsscheduleringsalgoritme er Rate Monotonic-scheduleringsalgoritmen (RMA), der er en af de mest anvendte. Den blev introduceret i 1973 af Liu og Layland [Mercer, 1992, s. 21].
RMA prioriterer tasks efter deres periode, T, kortest periode - højst prioritet. En tasks deadline er typisk sammenfaldende med dens periode. Udførslen af en task tager beregningstid, C. Denne må ikke være større end perioden for samme task.
Udnyttelsesgraden af en processor (utilization), U, er givet ved:

$$U = \frac{C}{T}. \tag{5.1}$$

Følgende ulighed skal være opfyldt for at et task-set med n tasks kan scheduleres med en perfekt scheduleringsalgoritme:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n} \leq 1. \tag{5.2}$$

For rate monotonic-algoritmen er denne grænse dog lavere. Den kan vises at være [Stalling, 1992, s. 439]:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n} \leq n(2^{1/n} - 1). \tag{5.3}$$

Værdien konvergerer mod $ln2 \approx 0,693$ for antallet af tasks, n, gående mod uendelig.
Uligheden beskriver et forhold, der skal overholdes, for at tasks i et taskset overholder deres deadlines, hvis de scheduleres med rate monotonic-algoritmen Ligning 5.2 gælder for Earliest Deadline First-algoritmen (EDF), og denne giver således en større udnyttelsesgrad af processoren. At RMA anvendes mere skyldes bl.a. priority inversion-problematikken, der er gældende for EDF, grundet dennes anvendelse af dynamisk prioritering.
Monotonien i RMA-algoritmen består i det lineære forhold mellem frekvens og prioritet.

5.6 Skaler- og schedulerbarhed

To forholdet er interessante for vores system med hensyn til realiseringen af kravene til systemet: At systemet overholder sine deadlines, dvs. at det givne taskset er schedulerbart, og at systemet er skalerbart.
En forudsætning for at vise, at task-set'et er schedulerbart, er at beregningstid samt periode kendes for hver task. Disse kan måles eller vanskeligt estimeres.
Den øvre grænse for skalerbarheden på en specifik processor vises ved tilføjelsen af en heltalskonstant ud for hver led i ligning 5.3:

$$U_i = A_i \cdot \frac{C_i}{T_i}, \qquad A = 1, 2, 3... \tag{5.4}$$

Her repræsenterer A_i antallet af den i'te task. Denne indgår så som en ubekendt i ligningen.
Denne vil i praksis ikke alene være begrænset af processoren men f.eks også netværk.

5.7 Bestemmelse af beregningstid og periode

Bestemmelsen af beregningstiden og perioden for en given task er en kompliceret sag, men metoden kan antydes med følgende eksempel.

Der ses på transceive-loop'et (sende og modtage et datagram) i netværksdelene. Et datagram indeholder 64 bytes data. a- og μ-law-komprimeringen (G.711) stiller krav om overførsel af $8kB/s$ til netværket. Dette giver transceive-loop'et følgende periode:

$$T = \frac{1}{\frac{2 \cdot 8000B/s}{64B/datagram}} = 0.004s. \tag{5.5}$$

Der divideres med 2, da der skal overføres 8kB/s i begge retninger.
Beregninstiden, c, må bestemmes ved konkrete målinger på eller estimering af transceive-loopet. Estimering er forbundet med visse vanskeligheder, grundet brugen af systemkald (recv/send). Udgøres et loop af assembler-instruktioner kan beregningstiden forholdsvis nemt bestemmes ved opslag i datablad.
Groft set rummer transceive-loopet følgende:

- modtage data fra LAN
- sende data til I/O-objekt
- modtage data fra I/O-objekt
- sende data til LAN

Det er så summen af beregningstiderne for hver enkel af disse operationer, der skal bestemmes, og herefter kan kan udnyttelsesgraden bestemmet efter ligning 5.1.

5.8 Konklusion

Systemet betragtes som værende et blødt realtidssystem. Hovedsageligt på grund af de ikke-fatale konsekvenser ved overskridelse af deadlines, men også på grund de indeterministiske dele, der indgår i systemet, såsom IPv4-netværket, samt kompleksiteten af det samlede system.

Kapitel 6

Operativsystem-analyse

6.1 Formål

Formålet er, på baggrund af en analyse af forskellige operativsystemer, at vælge det operativsystem, der egner sig bedst til afviklingen af en realtidsapplikation. Analysen tager udgangspunkt i kravene til et realtidsoperativsystem, som er beskrevet i [Bruxelles, 1997].
Operativsystemerne, der analyseres med udgangspunkt i kravene til realtidsoperativsystemer, er Windows NT, RTLinux og RTMACH.

6.1.1 Analysen

Et operativsystem kan indgå i forskellige typer af realtidssystemer. Valget af operativsystem afhænger af systemet, det skal indgå i. Der findes to typer realtidssystemer, der som benævnet i afsnit 5.2 klassificeres som hård og blød.
Følgende af de analyserede operativsystemer opfylder kravene til et blød realtidssystem : Windows NT, RTLinux og RTMach.
Er systemet et hård realtidssystem, er der krav til opbygningen og afviklingen af operativsystemet.
Kravene er som følgende:

- Understøttelse af multitrådet samt understøttelse af preemption.
- Prioritetsstyring af tråde.
- Trådsynkronisering af ressourcebrug.
- Prioritetsombytning.

6.1.2 Analyseresultater

Resultaterne er angivet i forhold til de ovenstående krav.

1. Et Realtidsoperativsystem skal være multitrådet og understøtte preemtion.

Windows NT Overholder kravene til multitrådethed og understøttelse af preemption.

RTLinux Operativsystemet er ikke multitrådet, men er derimod et multiproces operativsystem. Kravet til et multitrådet system overholdes ved at lade en proces svare til en tråd. RTLinux understøtter preemption.

RTMach Operativsystemet er ikke multitrådet, men derimod et multiproces operativsystem. Kravet til et multitrådet system overholdes ved at lade en proces svare til en tråd. RTMach understøtter preemption.

2. Styring af trådenes prioritet.

   Windows NT Operativsystemet har to klasser af prioriteter.

   Realtids : Fast prioritet, der kun kan ændres at tråden selv.

   Dynamisk : Dynamisk prioritet, der kan ændre af tråden selv og scheduleren. Problemet er antallet af valgbare prioriteter. Der er kun 5 prioriteter at vælge imellem inden for hver prioriteringsklasse.

   RTLinux Operativsystemet har et område af prioriteter, der ligger mellem hardware-interrupt og Linux-kernen, som er ligestillet med andre realtidsproceser. RTLinux råder over 65536 prioriteter.

   RTMach Operativsystemet giver mulighed for prioritering af processerne.

3. Trådsykronisering af resourcebrug.

   Windows NT Understøtter synkronisering af resourcerne.

   RTLinux Understøtter synkronisering af resourcerne.

   RTMACH Understøtter synkronisering af resourcerne.

4. Prioritetsombytning.

   Windows NT Overholder ikke kravet til prioritetsombytning. Dette skyldes at en tråd med fast prioritet kun kan ændres af tråden selv. Det sker ikke hvis den venter på en resource. Resultatet kan være deadlines, som overskrides.

   RTLinux Kravet overholdes af Operativsystemet, da scheduleren til hver en tid kan ændre prioritet på de enkelte processer.

   RTMACH Understøtter prioritetsnedarvning.

6.1.3 Konklusionen på analysen

Med udgangspunkt i ovenstående resultater kan det konstateres at Windows NT ikke overholder kravene til et hård realtidsoperativsystem. Det er kravet om prioritetsombytning, der ikke overholdes.

De to andre operativsystemer, der indgår i analysen overholder alle de ovenstående krav. Det gør dem velegnet til, at indgå som hård realtidsoperativsystem i et realtidssystem.
Med udgangspunkt i ovenstående analyse, perspektiveres operativsystemerne i forhold til anvendelsen i gateway'en og terminalen.
Fordelene ved RTLinux er følgende :

- God understøttelse af div. standardenheder (fx netkort).
- Godt kendskab til Linux i gruppen.

Ulemperne ved RTLinux er følgende :

- Tilpasning af device driver for realtidsafvikling påkrævet.
- Udsultning af Linuxkernen, hvilket medfører at systemet låser. Dette kan ske, hvis der tildeles for lidt processortid til den tråd, der indeholder linuxkernen.
- Mangelfuld dokumentation af API'et.

Fordelene ved RTMach :

- God dokumentation, samt programmeringseksempler.
- Optimeret til audio, video og netværk.

Ulemper ved RTMach :

- Mindre kendskab til FreeBSD, som RTMach kører under.
- Tilpasning af device driver for realtidsafvikling påkrævet.

Ud fra betragningen af fordele og ulemper ved RTLinux og RTMach, samt at Windows NT ikke er et hård realtidsoperativsystem, undersøges tilgængeligheden af driver til de anvendte kort.

Lydkort Der er understøttelse af lydkortet under alle operativsystemerne.

Netkort Der er understøttelse af netkortet, samt de anvendte protokoller under alle operativsystemerne.

ISDN-2 kort Det eneste operativsystem, der understøtter ISDN-2 kort, er Windows NT med den tilhørende driver. Til de andre operativsystemer skal der først laves driver.

På grund af manglende driver til ISDN-2 kortet vælges Windows NT som operativsystem på gateway'en. Terminalen kan implementeres på alle de analyserede operativsystemer. For at genbruge kildekoden mest muligt, vælges det at lave systemet til et dedikeret windows system.

6.2 Tidsanalyse af Windows.

6.2.1 Formål

Formålet med en tidsanalyse af Windows er, at analysere schedulering og latens under Windows, samt interkommunikationen mellem tråde og processer.
Versionerne af Windows, der analyseres med hensyn til skeduleringstider, er som følgende :

- Windows 98 anden udgave.
- Windows NT ver 4.0 Workstation.
- Windows 2000 professional RC3.

De nævnte versioner af Windows er alle begrænset af antallet af implementeret skeduleringsalgoritmer. Windows anvender Round Robinalgortimen.
De analyserede operativsystemer har et antal forskellige skeduleringsalgoritmer implementeret. Nedenstående er en liste over de algoritmer, de analyserede operativsystemer har implementeret.

Windows NT :

- Round Robin

RTLinux :

- Rate Monotonic.
- Earliest Deadlinie First.
- Brugerdefineret, bruger kan selv designe en algoritme.

RTMach :

- Rate Monotonic.
- First In First Out (FIFO).
- Round robin.
- Earliest Deadlinie First.
- Shortest processing time first.
- Fixed priority preemptive.
- Brugerdefineret, brugeren kan selv designe en algoritme.

6.2.2 Operativsystem-analysen

Tidsanalysen består af en række undersøgelser, der skal anskueliggøre latensen forsaget af operativsystemet, i udvalgte dele af en applikation. Undersøgelserne foretages på de tre tidligere nævnte versioner af Windows.
Undersøgelse af følgende er foretaget:

Trådskifte Undersøger scheduleringsalgortimen for schedulering af tråde inden for den samme proces. Dette gøres ved at måle frekvensen af én trådkørsel i en applikation med tre kørende tråde. De tre tråde udfører i den ene test samme funktionalitet, mens de udfører noget forskelligt i test nummer to. De to test gentages, men med en højere klasseprioritet (prioriteten på processen der afvikler de tre tråde).

Standardafvigelsen findes i forhold til middelværdien af tidsmålingerne.

En nærmere beskrivelse af undersøgelsen findes i punkt 6.2.2.

Processkifte Undersøger scheduleringsalgortimen for skedulering af processer. Dette gøres ved at måle frekvensen af én proceskørsel, hvor tre proces'er i den ene test udfører den samme funktionalitet, mens de udfører noget forskelligt i test nummer to. De to test gentages, men med en højere klasseprioritet på de tre processer.

Der undersøges med hensyn til standardafvigelsen i forhold til middelværdien af tidsmålingerne.

En nærmere beskrivelse af undersøgelsen findes i punkt 6.2.2.

Blokende kald Fem forskellige blokerende kald undersøges, med hensyn til standardafvigelsen i forhold til middelværdien af tidsmålingerne, for de enkelte kald. Følgende blokerende kald undersøges: Sleep, recv, send, Release og WaitForSingleObject. En nærmere beskrivelse af undersøgelsen findes i punkt 6.2.2.

Generelt for alle undersøgelserne sker tidstemplingen med funktionen QueryPerformanceCounter, som returnerer værdien af en hardware-tæller. For at konvertere tælleren til tid benyttes funktionen QueryPerformanceFrequency, der returnerer frekvensen af tælleren.
Tiden, der logges i de enkelte tråde og processer er absolut i forhold til hardware tælleren, der inkrementeres monotont og liniært. Dette giver mulighed for, at undersøge i hvilken rækkefølge trådene og processerne afvikles.
Tiderne gemmes i et dataområde for, efter endt test, at blive gemt i form af en kommasepareret fil. Tiderne gemmes i dataområderne på forskellige måder, alt efter om der er tale om en multitrådet eller en multiproces applikation. Ved multitrådede applikationer har trådene fælles adresseområdet, hvilket gør det muligt, at kommunikere gennem nogle fælles dataområder. Ved en multiproces applikation er funktionaliteten fordelt på flere mindre programmer og dermed separate dataområder. Der kommunikeres gennem pipe's (en system buffer mellem programmer) eller gennem Windows beskeder.

Undersøgelse af trådskifte

Testprogrammet består af tre tråde, der under de forskellige test har forskellig funktionalitet. Trådene er lavet, så der efter udførsel af trådens funktionalitet, skiftes der tråd.

Der foretages fire test, som hænger sammen parvis. Alle med samme funktionalitet i de tre tråde, men med forskellig prioritet. De fire test er beskrevet i figur 6.1.
Tiden logges i millisekunder, med mere en 1000 tidslogninger.
Følgende undersøges:

- Tiden mellem afvikling af en tråd.
- Rækkefølgen af trådudførelserne.
- Trådenes funktionalitets-påvirkning af udførelsesfrekvensen.

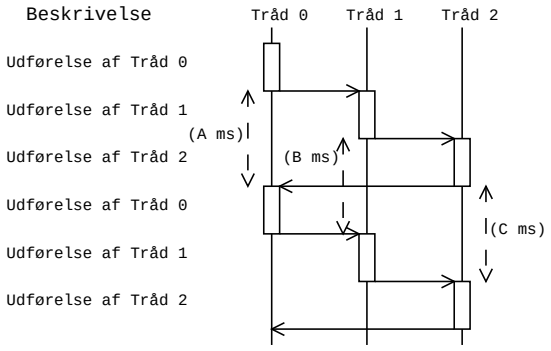
På figur 6.2 ses sekvensdiagram af testforløbet. Tiderne, der logges er A, B og C. For at kunne foretage ovenstående undersøgelser, logges udførelsesfrekvensen af de tre tråde. Tidslogningerne indekseres for at observere den rækkefølge trådene. I afsnit 6.2.3 findes resultaterne for de udførte test.

	Tråd 0	Tråd 1	Tråd 2
Test 0	- Logger tiden og frigiver tråden - Normal prioritet	- Logger tiden og frigiver tråden - Normal prioritet	- Logger tiden og frigiver tråden - Normal prioritet
Test 1	- Logger tiden og frigiver tiden - Normal prioritet	- Logger tiden, skriver 1000 linier @ 30 tegn til en fil og frigiver tråden - Normal prioritet	- Logger tiden, skriver 1000 værdier til et array og frigiver tråden - Normal prioritet
Test 2	- Logger tiden og frigiver tråden - Høj prioritet	- Logger tiden og frigiver tråden - Høj prioritet	- Logger tiden og frigiver tråden - Høj prioritet
Test 3	- Logger tiden og frigiver tråden - Høj prioritet	- Logger tiden, skriver 1000 linier @ 30 tegn til en fil og frigiver tråden - Høj prioritet	- Logger tiden, skriver 1000 værdier til et array og frigiver tråden - Høj prioritet

Figur 6.1: Trådenes funktionalitet og prioritet i testprogrammet for multitråde.

Undersøgelse af processkifte

Testprogrammet består af tre processer, der under de forskellige test har forskellig funktionalitet. Processen er lavet, så efter udførsel af processens funktionalitet, skiftes der proces. Der benyttes én pipe pr. proces til overførelse af tidslogningen; det gør at proces'erne ikke låser pipe'en for hinanden.
Der foretages fire test, som hænger sammen parvis. Det vil sige funktionaliten af de tre processer er den samme, med prioriteten er forskellig. De fire test er beskrevet i figur 6.3.
Tiden logges som tidligere i millisekunder, med mere end 1000 tidslogninger.
Følgende undersøges:



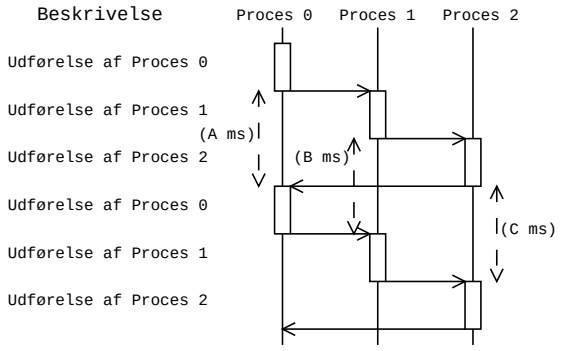
Figur 6.2: Sekvensdiagram for analysen af multitråde.

- Tiden mellem udførelse af en proces.
- Rækkefølgende af procesudførelsene.
- processernes funktionalitets påvirkning af udførelsesfrekvensen.

På figur 6.4 ses sekvensdiagrammet af testforløbet. Tiderne, der logges, er A, B og C. For at kunne foretage ovenstående undersøgelser, logges udførelsesfrekvensen af de tre processer. Tidslogningerne indekseres for at observere rækkefølgen processerne udføres. I punkt 6.2.3 findes resultaterne for de udførte test.

	Proces 0	Proces 1	Proces 2
Test 0	- Logger tiden og frigiver processen - Normal prioritet	- Logger tiden, frigiver processen - Normal prioritet	- Logger tiden og frigiver processen - Normal prioritet
Test 1	- Logger tiden og frigiver processen - Normal prioritet	- Logger tiden, skriver 1000 linier @ 30 tegn til en fil og frigiver processen - Normal prioritet	- Logger tiden skriver 1000 værdier til et array og frigiver processen - Normal prioritet
Test 2	- Logger tiden og frigiver processen - Høj prioritet	- Logger tiden og frigiver processen - Høj prioritet	- Logger tiden og frigiver processen - Høj prioritet
Test 3	- Logger tiden og frigiver processen - Høj prioritet	- Logger tiden, skriver 1000 linier @ 30 tegn til en fil og frigiver processen - Høj prioritet	- logger tiden, skriver 1000 værdier til et array og frigiver processen - Høj prioritet

Figur 6.3: processernes funktionalitet og prioritet i testprogrammet for multiproces.



Figur 6.4: Sekvensdiagram for analysen af multiproces.

Undersøgelse af blokerende kald

Standardafvigelsen i et blokerende kald er med til at give en variabel latens i applikationen. For at anskueliggøre betydningen af et blokerende kald undersøges fem forskellige kald for deres standardafvigelse.

Sleep Sleep bruges til at stoppe tråden en periode, angivet i millisekunder. Dette gøres ved at suspendere tråden.

Testen forgår ved at foretage en tidslogning før og efter kaldet af Sleep. Sleep sættes op til en forsinkelse på 5 ms, 10 ms, 15 ms.

Resulteterne af undersøgelserne findes under afsnit 6.2.3.

WaitForSingleObject Funktionen venter på at et objekt bliver signaleret. Funktionen venter, et specificeret tidinterval, før den melder timeout. Er objektet signaleret inden for den specificeret timeout, overtager funktionen i den aktuelle tråd ejerskabet af objektet. Det betyder at andre tråde, som venter på en signalering af objektet suspenderes.

Testprogrammet er opbygget, som en løkke, der afbrydes hvis objektet signaleres. Tidslogningen sker før objektet signaleres, efter objektet er signaleret, samt efter funktionen WaitForSingleObjekt. Det giver latenserne A og C på sekvensdiagram 6.5. De svarer til tiden fra signalering af et objekt mellem to tråde, samt en overtagelse af objektet.

Resultaterne af undersøgelserne findes i afsnit 6.2.3.

Release Funktionen signalerer frigivelse af et objekt. Funktionen er i sig selv ikke et blokerende kald, men anvendes i sammenhæng med ovenstående funktion WaitForSingleObject. Det vil derfor være naturligt at lade den indgå i undersøgelserne. Latenserne B og D på sekvensdiagram 6.5 angiver tiden for at signalere til et objekt.

Resulteterne af undersøgelserne findes under afsnit 6.2.3.

recv Funktionen blokerer tråden, indtil der er modtaget datapakker på den aktuelle socket. I testprogrammet sendes der til localhost, det vil sige, at datapakkerne ikke kommer ud på det fysiske netværk, men bliver loopet tilbage i TCP/IP-stakken. Latens, der er forårsaget af det fysiske netværk, optræder ikke i målingerne for denne test. Latensen på et netværk, er beskrevet i afsnit 4.1.6. Testprogrammet sender datapakkerne via UDP-protokollen. Protokollen for testprogrammet er valgt på baggrund af de typer af datapakker, der sendes i projektet. Da de største datamængder, der sendes over netværket sendes via UDP-protokollen, er det nærlæggende, at undersøge denne protokol.

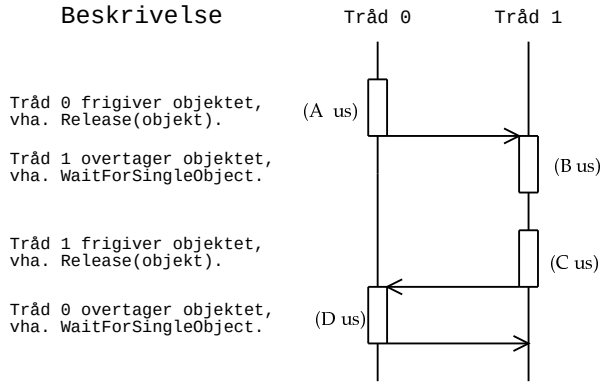
Testen består af to processer, en server og en klient. Serveren startes først; den venter på at klienten sender en UDP-datapakke ved hjælp af send-funktionen. Som det fremgår af sekvensdiagrammet figur 6.6 skiftes klient-tråden og server-tråden til at sende og modtage datapakker. Latenserne N og P på sekvensdiagrammet 6.6 angiver tiden for at modtage en 64 byte UDP-datapakke fra localhost via sockets.

Resultaterne af undersøgelserne findes i afsnit 6.2.3.

send Funktionen sender data ud på et netværk via sockets. Funktonen er i sig selv ikke et blokerende kald, men anvendes i sammenhæng med ovenstående funktion recv.

Resultaterne af undersøgelserne findes i punkt 6.2.3.

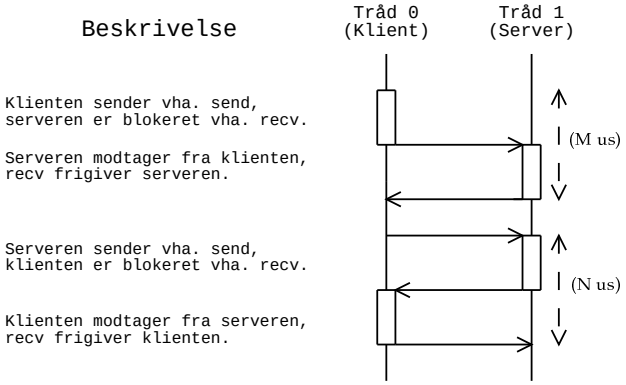
Tiden logges i millisekunder, med mere end 1000 tidslogninger. Dette er gældende for tråd- og processkeduleringstestene, mens de blokerende kald måles i microsekunder og med samme antal tidslogninger.



Figur 6.5: Sekvensdiagram for analysen af signalering mellem 2 tråde.

6.2.3 Resultater af operativsystem-analysen

Resultaterne af undersøgelserne skal vise, hvor stor en latens, der er forårsaget af operativsystemet, samt afvigelsen i forhold til middelværdien. For at kunne



Figur 6.6: Sekvensdiagram for analysen af send og recv funktioner på netværk.

sammenligne de undersøgte versioner af Windows, er alle målingerne foretaget på samme PC. For at simulere betydningen af udforudset kørsel fra operativsystem startes og afsluttes programmerne Word og Excel fra Microsoft.
Da windows benytter sig af scheduleringsalgortimen Round Robin er antallet af tråde og processer bestemmende for tildelingen af CPU-tid. I tabellen figur 6.7 vises antallet af tråde og processer for de anvendte Windowsversioner inden afvikling af testprogrammerne.

Computer beskrivelse: Pentium Celeron 400MHz, 128 MB ram, 128 KB L2 Chache		
	Tråde	Processer
Win 98	25	11
Win NT	71	12
Win 2000	200	16

Figur 6.7: Antallet af proces'er og tråde for et operativsystem i idle tilstand.

Formler til beregning af middelværdi og standardafvigelse

Til beregning af middelværdierne, samt standardafvigleserne for de enkelte test er Excel fra Microsoft anvendt.
Formlen til beregning af middelværdier er angivet i formel 6.1.

$$\mu = \frac{\sum x_i}{N} \tag{6.1}$$

Formlen til beregning af standardafvigelsen er angivet i formel 6.2.

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}} \tag{6.2}$$

Formlerne 6.1 og 6.2 bruges til analysen af resultaterne af undersøgelserne beskrevet i afsnit 6.2.2. For at vurdere resultaterne undersøges hvor meget

målingerne spreder sig i forhold til middelværdien. Der undersøges for en normalfordeling af målingerne, d.v.s. at 95,44 % af målingerne ligger inden for ± 2 gange standardafvigelsen. Det giver en indikation af, hvor ensartet målingerne er. Jo mindre spredningen er, jo mere ensartet/forudsigelig er latensen.

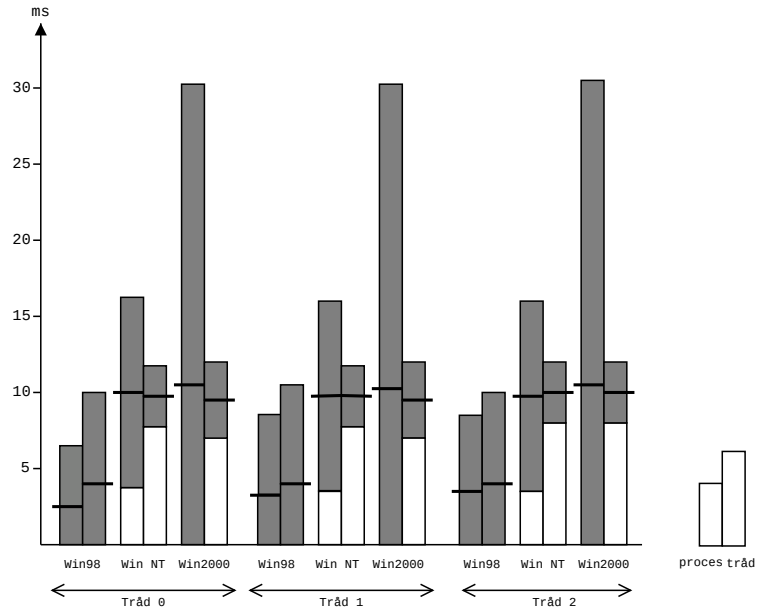
Scheduleringsalgoritmen

Resultaterne af tråd og proces schedulering, analyseres med udgangspunkt i fire test for både tråd- og processkrifte, test 0-3 beskrevet i figur 6.1 og 6.3. Resultatet af hver test vil fremgå i form af det søjlediagram, der sammenligner trådskifte med processkifte.
Fortolkningen af søjlediagrammet er som følger:

- De parvise søjler repræsenterer en sammenhæng af tråd- og processkifte. Søjlen til venstre repræsenterer et processkifte og søjlen til højre et trådskifte.
- De parvise søjler er grupperet i forhold til versionen af Windows, som igen er grupperet i forhold til hvilken tråd, der undersøges.
- Den enkelte søjle er delt op efter middelværdien og 2· standardafvigelsen. De mørke områder af søjlen repræsenterer 2· standardafvigelen, hvilket svarer til 95,44 % af målingerne.
- De brede og tværgående linier repræsenterer middelværdien for målingerne. Standardafvigelserne er indikeret i forhold til denne.
- Processer er angivet som tråde, da en proces altid indeholder minimum 1 tråd.

Analyse af test 0

Søljediagrammet figur 6.8 viser resultatet for test 0 i tabellerne i figur 6.1 og 6.3. Som det fremgår af figur 6.8 er win98 hurtigst til af skifte skifte tråd/proces. Det kan begrundes med at win98 ikke skifter sekventielt mellem trådene, hvorimod det er tilfældet for både winNT og win2000. En anden observation er, at hverken tråd- eller processkifte sker efter en normalfordeling. Det begrundes med, at spredningen ikke ligger jævnt fordelt på begge sider af middelværdien.
For både winNT og win2000 gælder det at trådskifte er hurtigere end processkifte, samt at standardafvigelsen er betydelig mindre for trådskifte. Dette betyder at latensen er mere homogent for trådskifte. Processkifte under win2000 er heller ikke normaltfordelt. Det kan forklares ved, at win2000 fra starten har flere processer kørende end de to andre, hvilket kan give store afvigelser under opstart af Microsoft Word og Excel.
Der er minimalt forskel på middelværdien for de enkelte tråde, hvilket betyder at alle tråde for lige meget CPU-tid.



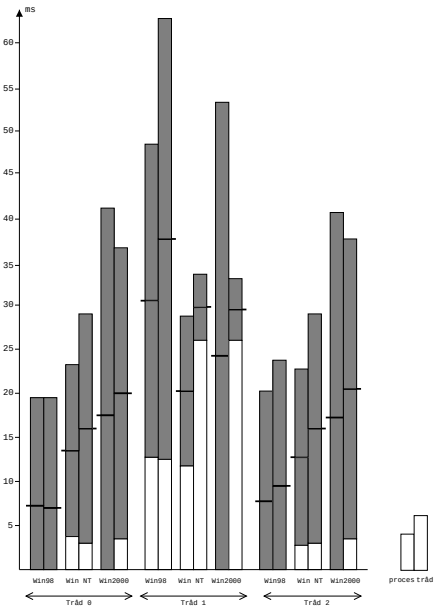
Figur 6.8: Søjlediagram for skedulering af tråde og processer for test 0.

Analyse af test 1

Søjlediagrammet figur 6.9 viser resultatet for test 1 i tabellerne i figur 6.1 og 6.3. Som det fremgår af figur 6.9 er standardafvigelsen betydelig større end ved test 0. Til forskel fra test 0 er der I/O-tilgang i tråd 1, samt hukommelsestilgang i tråd 2. Denne forskel betyder, at middelværdien er generelt højere for alle tråde/proces'er, også for tråd 0, der har samme funktionalitet. Årsagen er, at CPU-tiden fordeles mellem processerne med samme prioritet, og trådene i en proces deler tiden processen har fået tildelt. Alle processer med samme prioritet køres skiftevis efter Round Robin algoritmen. Det illustreres i særdeleshed under processkifte på win2000.
Win98 afviger betydeligt ved I/O-tilgang i forhold til de andre tråde. Win98 og win2000 er som i test 0 ikke normalfordelt.

Analyse af test 2.

Søjlediagrammet på figur 6.10 viser resultatet for test 2 i tabellerne i figur 6.1 og 6.3. Til forskel fra test 0 og test 1 er prioritet på processerne og trådene hævet til en højere prioritetklasse. Det betyder, at de oftere vil blive kørt. Sammenlignes resultaterne fra test 0 med dem fra test 2, ses det tydeligt, at winNT og især win2000 er blevet mere homogene i afviklingen af processerne og trådene. Win98 ligner til forveksling resultaterne fra test 0. Middelværdierne for operativsystemer ligger på samme niveau, hvilket giver en indikation på, at der er flere processer med normal prioritetsklasse, der gør brug CPU'en end processer med en højere prioritet. Ændring af prioriteten har ikke haft nogen indvirkning



Figur 6.9: Søjlediagram for skedulering af tråde og processer for test 1.

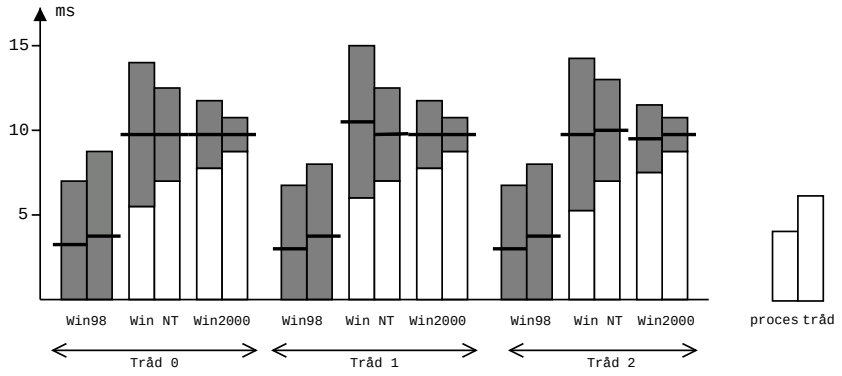
på win98's skedulering, så den er blevet normaltfordelt.
Studeres rækkefølgen af proces-/trådkørslerne viser det sig at win98, som i test 0 ikke forløber sekventielt. Det kan give forsinkelser, hvis en tråd venter på en anden tråd skal signalere et objekt, kan den ventende tråden udsultes.

Analyse af test 3

Søjlediagrammet figur 6.11 viser resultatet for test 3 i tabellerne i figur 6.1 og 6.3. Test 3 tilgår, lige som test 1, I/O. Til forskel fra test 1 er standardafvigelsen for processkifte under winNT og win2000 mindre. Det indikerer, som i resultatet af test 2, at skiftene er mere homogene i det processerne kaldes mere regelmæssigt. Grundet forsinkelsen, der findes i forbindelse med tilgang af I/O, er win98's tråd 1, den indeholdende I/O-tilgangen, blevet normal fordelt. Årsagen til dette kan være at scheduleringen af tråd 1, skal synkroniseres med kernes I/O-håndtering, hvilket gør, at win98 ikke kan udføre tråd 1 to gange i træk.

Analyse af Sleep funktionen

Analysen af funktionen Sleep omfatter de 3 versioner af Windows. Der undersøges for virkemåden, samt den aktuelle tidsforsinkelse funktionen giver.
Søjlediagrammet figur 6.12 viser resultatet for testen af funktionen Sleep, som er beskrevet i afsnit 6.2.2. Som det fremgår af søjlediagrammet sker forsinkelsen, som Sleep giver tråden, i spring. Sleep funktionen suspendere tråden for en givet tidsperiode, hvilket resulterer i at den næste tråd får CPU-tid. I forbindelse med testen af Windows's scheduleringalgortime i afsnit 6.2.3, viste resultatet at der



Figur 6.10: Søjlediagram for skedulering af tråde og processer for test 2.

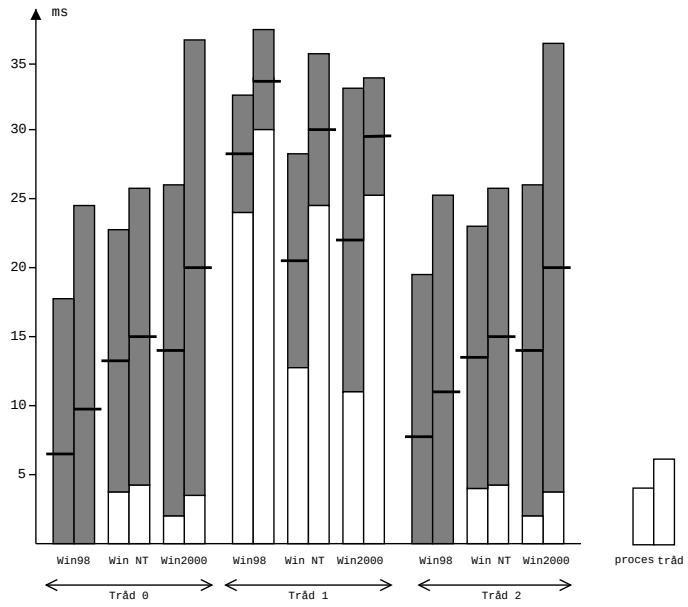
går ca. 10 ms mellem en tråd får CPU-tid. Det er de samme 10 ms spring, som sleep funktionen har som opløsning.

Analyse af WaitForSingleObjekt og Release funktionerne.

I analysen af funktionerne WaitForSingleObject og Release undersøges latensen ikke for de 2 tråde separat. Det vil sige latenserne A og C for Release, samt B og D for WaitForSingleObject indgår i den samme søjle i søjlediagrammet. Forholdet mellem latenserne er vist i figur 6.5.
Søjlediagrammet figur 6.13 viser resultatet for testen af funktionerne WaitForSingleObject og Release beskrevet i afsnit6.2.2. De forskellige versioner af windows afviger ikke meget fra hinanden mht. middelværdien. Dog ser det ud til at win98 ikke håndterer WaitForSingleObject så homogent som de to andre. Det kan skyldes, at trådscheduleringen i win98 ikke er sekventiel. Dette fremgik af analysen af skeduleringen i afsnit 6.2.3. Med middelværdier på ca. 10 μs har opløsningen for hardwaretælleren en indvirkning på resultatet. Den pågældende testcomputer har en opløsning på 1,19 μs.

Analyse af netværksfunktionerne send og recv

I analysen af netværksfunktionerne send og recv, som er en del af socket API'et, undersøges latensen for kommunikation til og fra TCP/UDP-stakken. Latenserne N og M vist på figur 6.6 måles ved, at tidslogningerne for både server- og klientprocessen er absolutte i forhold til hardwaretælleren. Differencen i tid, fra datapakken er sendt til den er modtaget, giver en indikation af latensen fra applikationslaget til transportlaget i OSI-stakken.
Søjlediagrammet figur 6.14 viser resultatet for testen af funktionenerne send og recv beskrevet i afsnit 6.2.2. Resultatet af søjle M og N er afvigende i forhold til hinanden, hvilket er gældende for alle versioner af windows. Det kan skyldes, at Windows' processchedulering har ændret prioritet på en af processerne, hvilket er muligt med en normal prioritetsklasse. En anden mulighed kan være den



Figur 6.11: Søjlediagram for skedulering af tråde og processer for test 3.

rækkefølge processerne udføres i. Det begrundes ved at analysere de data, der ligger til grund for den statiskeberegning. Her fremgår det, at i mindre perioder stiger tiden for at sende datapakker via sockets. Da perioden kun forekommer for enten serveren eller klienten, vil det resultere i en højere middelværdi og en større standardafvigelse og dermed en difference mellem N og M.

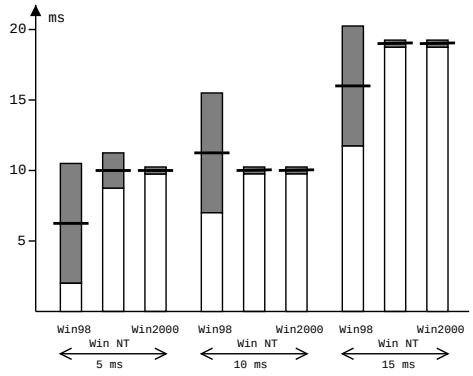
6.3 Konklusion på operativsystem-analysen

Afsnittet indholder en konklusion af de udførte test. Der tages udganspunkt i vurderingerne, der er foretaget udfra resultatet af den enkelte test.

Win98 Windows 98 gør brug af skeduleringsalgoritmen Round Robin. Der er ikke den store forskel på skift mellem tråde og skift mellem processer, dog er processkifte hurtigst. En forhøjelse af prioritetsklassen medfører ikke et hurtigere tråd- eller processkifte. Dette er gældende for tråde og processer med samme funktionalitet.

Ændres funktionaliteten af trådene og processerne til at ombefatte henholdvis hukommelses- og I/O-tilgang. Betyder det en generel stigning i middelværdien af tidsmålingerne for alle tråde og processer. Ændres prioritetsklassen til et højere niveau påvirkes især tråden med I/O tilgangen. Årsagen kan være, at operativsystemets håndtering af I/O-tilgang, giver trådene højere prioritet. Resultatet er en mindre spedning i tidslogningerne.

Spredningen i tidslogningerne måles ved $\pm$ 2 gange standardafvigelsen



Figur 6.12: Søjlediagram for resultatet af testen af funktionen sleep.

i forhold til middelværdien. I alle pånær et eksempel på tråd- og processkifte, ved I/O tilgang, er scheduleringen af skiftene ikke normaltfordelt. Det kan forklares ved analysere rækkefølgen af tråd- og proceskørslerne. For Win98 sker det ikke sekventielt. Det medfører at en tråd kan køres to gange i træk. En anden indvirkende faktor er antallet af kørende tråde og processer, som skal dele CPU-tiden.

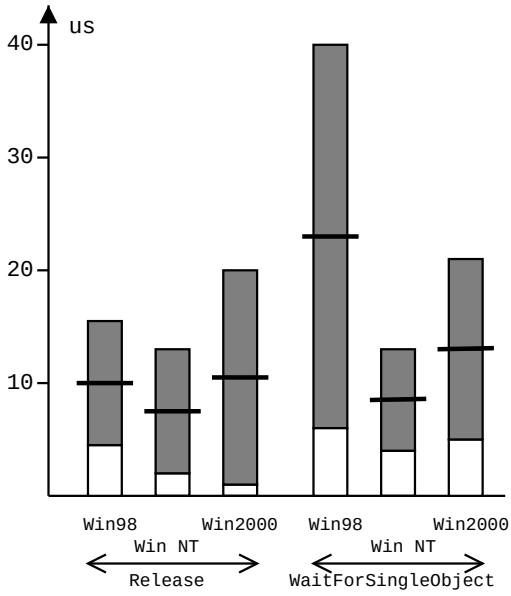
Win98 kontrollerer ikke blokerende kald særlig homogent. Det vises ved, at spredningen er større end for de to andre undersøgte versioner af Windows. Det kan forklares med de tidligere nævnte faktorer. For det blokerende kald Sleep, passer forsinkelsen bedre med den forventede, fordi frekvensen på tråd- og processkifte er mindre end for de to andre versioner.

WinNT Windows NT anvender ligesom Win98 scheduleringsalgotirmen Round Robin. Der er ikke den store forskel mellem scheduleringen af tråde og processer. Det viser sig udtrykt i den ensartethed, der er i middelværdien for tidsmålingerne, hvor trådskifte ikke er hurtigere end processkifte og vice versa. En forhøjelse af prioritestklassen medfører ikke et hurtigere tråd- eller processkifte.

Ændres funktionaliteten af tråde og processerne til, at gøre brug af henholdsvis hukommelses og I/O-tilgang. Betyder det en generel stigning i middelværdien af tidsmålingerne i forhold til de anfre versioner; for alle tråde og processer. Middelværdien af tidsmålingerne stiger også for de tråde og processer der bibeholder sin funktionalitet, hvilket vil sige, at konstrueres en meget kompleks funktionalitet i en tråde eller proces, vil det påvirke kørslen af de andre tråde og processer.

WinNT er den af de undersøgte versioner, der hånterer I/O-tilgang bedst, det er gældende både med normal og høj prioritetsklasse.

Ved schedulering af processer og tråde med både normal og forhøjet prioritetsklasse er skiftet normaltfordelt, og skiftende forløber skeventielt. WinNT er den eneste af de tre versioner af Windows, der i alle skeduler-



Figur 6.13: Søjlediagram for resultatet af testen af funktionenerne WaitForSingleObject og Release.

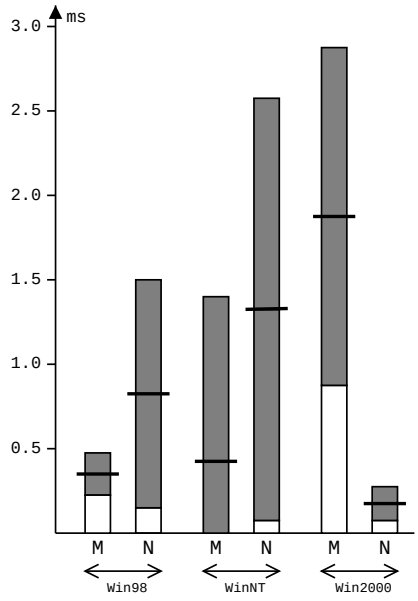
ingstestene er normalt fordelt.

WinNT håndterer blokerende kald homogent, dog er der afvigelser og ikke en normalfordelt udførelse af kommunikation med netværket. Det kan muligvis afhjælpes med forskellige service pakker til WinNT.

Win2000 Windows 2000 gør på samme måde, som de to andre versioner, brug af Round Robin. Der er minimal forskel mellem tråd- og processkifte. Ud fra resultatet af test 0 og test 2 er trådskifte ikke hurtigere end processkifte og vice versa. En forhøjelse af prioritetsklassen medfører ikke et hurtigere tråd- eller processkifte, der er gældende for tråde og processer med samme funktionalitet, hvilket er en tidslogningsløkke (test 0 og 2).

Ændres funktionaliteten af trådene og processerne til, at ombefatte henholdsvis hukommelses og I/O tilgang. Betyder det en generel stigning i middelværdien af tidsmålingerne for alle tråde og processer. Middelværdien af tidsmålingerne stiger også for de tråde og processer der bibeholder sin funktionalitet, hvilket vil sige, at konstrueres en meget kompleks funktionalitet i en tråde eller proces, vil det påvirke kørslen af de andre tråde og processer. Ændres prioritetsklassen til et højere niveau minskes middelværdien for tråden med I/O tilgang.

Ved skedulering af processer med en normal prioritetsklasse er skiftet ikke normaltfordelt, men til forskel fra Win98 forløber skiftene sekventielt. En af de faktorer, der kan være med til at forårsage den afvigelse, kan være antallet af tråde og processer. Hvilket er større end ved Win98, se tabellen



Figur 6.14: Søjlediagram for resultatet af testen af funktionenerne send og recv.

i figur 6.7. En anden faktor kan være opstart af både Word og Excel fra Microsoft, der bruger meget CPU-tid for en hurtig opstart.

Win200 kontrollerer blokerende kald homogent. Dog er der afvigelser i netværkstesten, hvor spredningen i den ene retning afviger i forhold til den anden retning. I de andre test af blokende kald er spredningen minimal.

Kapitel 7

Softwareanalyse

7.1 Indledning

Dette kapitel indeholder en analyse af det problemområde, der er specificeret i kapitel 2.1. På baggrund af overvejelser omkring ressourcer, er der foretaget en afgrænsning, således at analysen ikke omfatter den i Anbefaling H.245 specificerede kontrolkanal. Anbefaling H.323 definerer en Fast Connect procedure (se afsnit 3.3.1), der muliggør kaldsopsætning uden om kontrolkanalen, og analysen er foretaget på grundlag af denne procedure.

7.2 Struktur

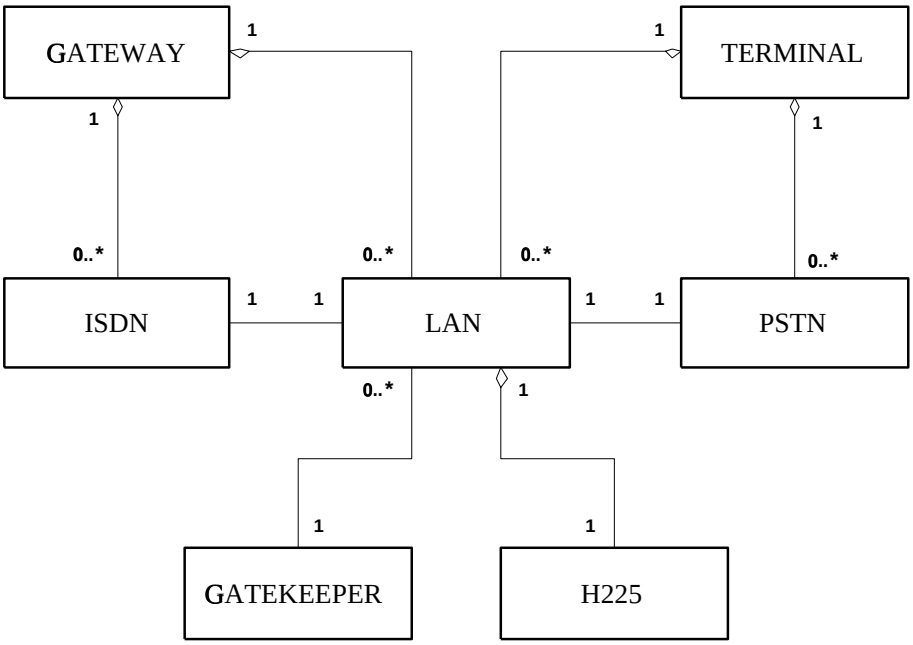
Figur 7.1 viser et samlet klassediagram for objektsystemet. Klassen Gateway er en aggregering af klasserne ISDN og LAN og klassen Terminal er en aggregering af klasserne PSTN og LAN. Klasserne ISDN, LAN og PSTN kan opfattes som repræsentationer af en telefon-forbindelse. Associeringerne mellem ISDN og LAN og LAN og PSTN afspejler at en forbindelse mellem ISDN og PSTN går over LAN. Gatekeeperen (beskrevet i afsnit 3.2.3) er repræsenteret som en autonom enhed, der er associeret med LAN. Klassen H225 (beskrevet i afsnit 3.2.1) er en repræsentation af H.225-protokollen og opfattes som værende indeholdt i LAN.

7.3 Klasser

Dette afsnit indeholder en oversigt over de klasser i systemet, der er blevet fastlagt igennem analysen, samt en beskrivelse af deres formål, attributter og operationer. En systembeskrivelse findes i afsnit 2.2.1.

7.3.1 Gateway

Formålet med klassen Gateway er at forbinde et IP-LAN med en ISDN-linje og omvendt. Figur 7.2 viser en livscyklus for et objekt af klassen Gateway.



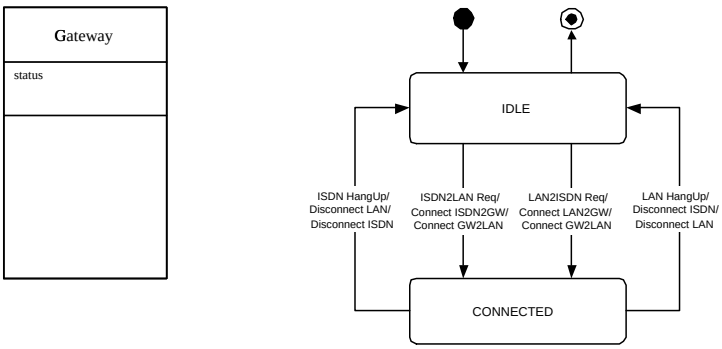
Figur 7.1: Samlet klassediagram for objektsystemet

Attributter

- status.

Operationer

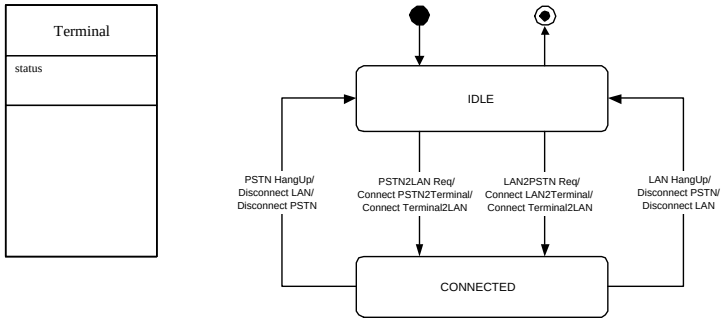
- N/A.



Figur 7.2: Tilstandsdiagram for klassen Gateway

7.3.2 Terminal

Formålet med klassen Terminal er at forbinde et IP-LAN med en PSTN-linje og omvendt. Figur 7.3 viser en livscyklus for et objekt af klassen Terminal.



Figur 7.3: Tilstandsdiagram for klassen Terminal

Attributter

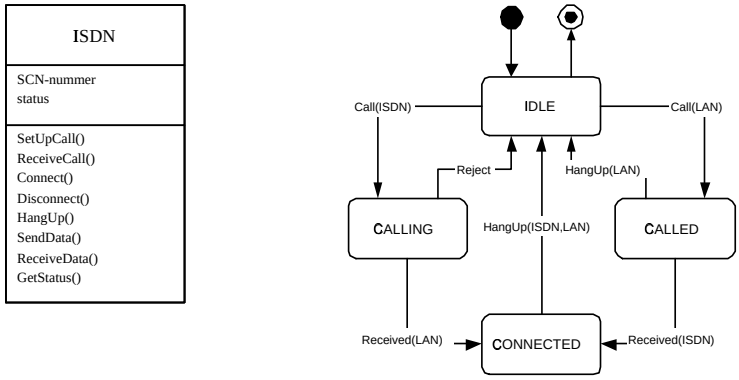
- status.

Operationer

- N/A.

7.3.3 ISDN

Formålet med klassen ISDN er opsætning og terminering af kald til en ISDN-linje, modtagelse af data fra en ISDN-linje og transmission af data til en ISDN-linje samt registrering af kontrol- og status-information vedrørende en ISDN-linje. Figur 7.4 viser en livscyklus for et objekt af klassen ISDN.



Figur 7.4: Tilstandsdiagram for klassen ISDN

Attributter

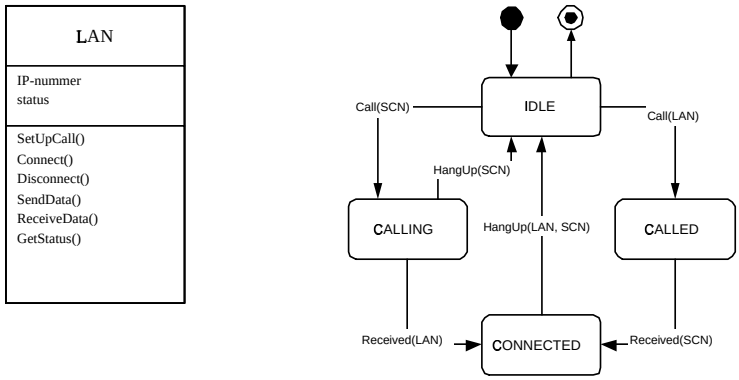
- SCN-nummer, status.

Operationer

- SetUpCall, ReceiveCall, Connect, Disconnect, HangUp, SendData, ReceiveData, GetStatus.

7.3.4 LAN

Formålet med klassen LAN er opsætning og terminering af kald til et IP-LAN, modtagelse af data fra et IP-LAN og transmission af data til et IP-LAN samt registrering af status- og kontrol-information vedrørende et IP-LAN. Figur 7.5 viser en livscyklus for et objekt af klassen LAN.



Figur 7.5: Tilstandsdiagram for klassen LAN

Attributter

- IP-nummer, status.

Operationer

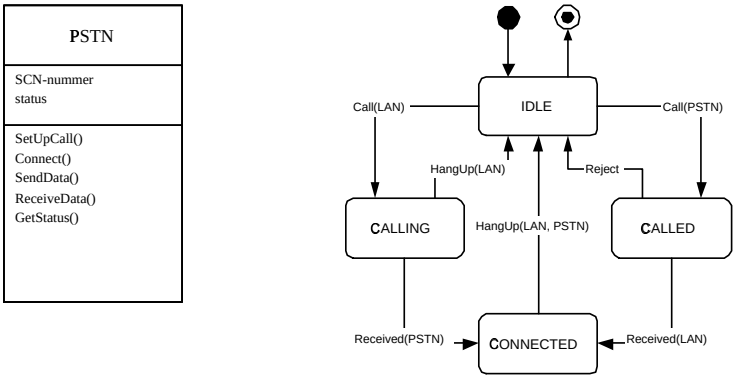
- SetUpCall, Connect, Disconnect, SendData, ReceiveData, GetStatus.

7.3.5 PSTN

Formålet med klassen PSTN er opsætning og terminering af kald til en PSTN-terminal, modtagelse af data fra en PSTN-terminal og transmission af data til en PSTN-terminal samt registrering af status- og kontrol-information vedrørende en PSTN-terminal. Figur 7.6 viser en livscyklus for et objekt af klassen PSTN.

Attributter

- SCN-nummer, status.



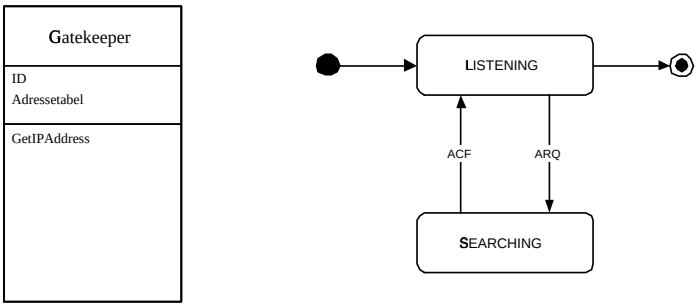
Figur 7.6: Tilstandsdiagram for klassen PSTN

Operationer

- SetUpCall, Connect, Disconnect, SendData, ReceiveData, GetStatus.

7.3.6 Gatekeeper

Gatekeeper'en håndterer adgangskontrol og oversættelse mellem SCN(Switched Circuit Network)-adresser og IP-adresser. Figur 7.7 viser en livscyklus for et objekt af klassen Gatekeeper.



Figur 7.7: Tilstandsdiagram for klassen Gatekeeper

Attributter

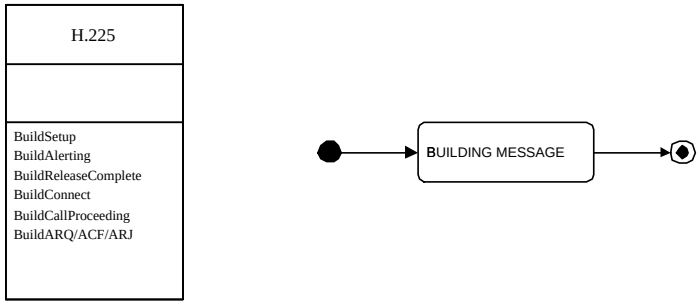
- ID, Adressetabel.

Operationer

- GetIPAddress.

7.3.7 H225

Klassen H225 indeholder de Q.931-kald, der skal anvendes i henhold til Anbefaling H.225.0. Figur 7.8 viser en livscyklus for et objekt af klassen H225.



Figur 7.8: Tilstandsdiagram for klassen H225

Attributter

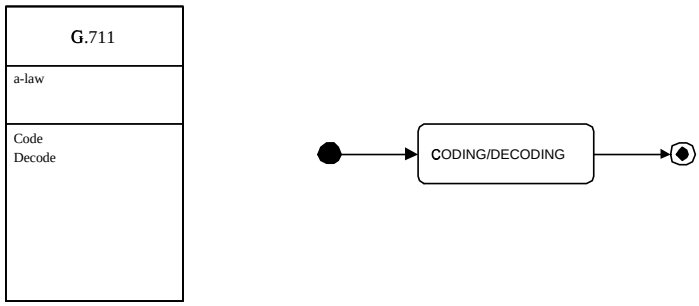
- Q.931-parametre.

Operationer

- BuildSetUp, BuildAlerting, BuildReleaseComplete, BuildConnect, BuildCallProceeding.

7.3.8 G711

Klassen G711 indeholder de operationer til komprimering af lyd, der skal anvendes i henhold til H.323-standarden. Figur 7.9 viser en livscyklus for et objekt af klassen G711.



Figur 7.9: Tilstandsdiagram for klassen G711

Attributter

- A-law

Operationer

- Code, Decode.

7.4 Funktioner

I det følgende findes en oversigt over de funktioner, der skal understøttes af klasserne ISDN, PSTN og LAN. Funktionerne til at sende og modtage data på LAN er ikke medtaget, da vi bruger sockets. Beskrivelsen af hver funktion omfatter den begivenhed, der trick'er funktionen (Event flow input), parametre som funktionen kaldes med (Data input) samt funktionens eventuelle returværdi. En yderligere specifikation af parametre og returværdier kan forekomme i designfasen.

7.4.1 ISDN

Figur 7.10 viser en oversigt over de funktioner, der skal undestøttes af ISDN-klassen.
Setup, Connect, ARJ (Admission Reject) og ReleaseComplete er H.225.0-beskeder. Setup initierer opsætningen af et kald fra LAN, Connect fuldender opsætningen af et kald til LAN og ReleaseComplete afbryder en forbindelse fra LAN. ARJ sendes fra gatekeeperen og indikerer, at adgang til LAN nægtes.
CONNECT_IND, DISCONNECT_IND, DISCON_B3_IND (DISCONNECT_B3_IND), DATA_B3_IND og CON_B3_ACT_IND (CONNECT_B3_ACTIVE_IND) er CAPI-beskeder. CONNECT_IND initierer opsætningen af en fysisk kanal fra ISDN-siden, CONNECT_B3_ACTIVE_IND afslutter opsætningen af en logisk kanal og DISCONNECT_IND og DISCONNECT_B3_IND indikerer afbrydelsen af henholdsvis den fysiske og logiske kanal fra ISDN-siden. DATA_B3_IND indikerer, ankomsten af data fra ISDN-siden.
data repræsenterer en datapakke fra LAN. **data*** er en pointer til **data** og DATA_LENGTH en konstant, der angiver størrelsen af **data**. **status** angiver tilstanden for et objekt af klassen. CalledPartyNumber og CallingPartyNumber er henholdsvis det kaldte og det kaldende nummer.

7.4.2 LAN

Figur 7.11 viser en oversigt over de funktioner, der skal understøttes af LAN-klassen.
DTMF_IN er en indikation af DTMF-toner fra en PSTN-telefon. PSTN_OFF_HOOK indikerer at røret på en sådan telefon er løftet og PSTN_ON_HOOK indikerer at røret er lagt på. ACF er en H.225.0-besked fra gatekeeperen, der angiver, at der er givet adgang til LAN. Se endvidere beskrivelsen af funktionerne i ISDN-klassen

Funktion	Event flow input	Data input	Output
SetupCall()	Setup	CalledPartyNumber CallingPartynumber	**status**
ReceiveCall()	CONNECT_IND	N/A	N/A
Connect()	Connect	N/A	N/A
Disconnect()	ARJ ReleaseComplete	N/A	**status**
HangUp()	DISCONNECT_IND, DISCON_B3_IND	N/A	**status**
SendData()	**data**	**data*** DATA_LENGTH	**data*** DATA_LENGTH
ReceiveData()	DATA_B3_IND	**data*** DATA_LENGTH	**data*** DATA_LENGTH
GetStatus()	CONNECT_IND DISCONNECT_IND CON_B3_ACT_IND	N/A	**status**

Figur 7.10: Oversigt over funktioner i ISDN-klassen

7.4.3 PSTN

Figur 7.12 viser en oversigt over de funktioner, der skal understøttes af PSTN-klassen. Se endvidere beskrivelsen af funktionerne i ISDN- og LAN-klasserne.

7.5 Funktionalitet

Følgende afsnit indeholder en række aktivitetsdiagrammer, der har til formål at give et overblik over flow'et i systemet ved henholdsvis kaldsopsætning og dataoverførsel.

7.5.1 Kaldsopsætning for Gateway

Figur 7.13 viser en model for funktionaliteten for Gateway'en ved kaldsopsætning. To funktioner, isdn.GetStatus og lan.Getstatus kaldes på skift indtil status for enten ISDN eller LAN skifter fra "Idle"til "Calling".
Kommer kaldet fra ISDN-siden kaldes funktionen isdn.ReceiveCall, der svarer på en CAPI_CONNECT_IND (connect indikation) med en CAPI_CONNECT_RESP (connect response), samt en CAPI_ALERT_REQ (alert request), der signalerer, at der arbejdes med at sætte kaldet op på LAN.
Dernæst kaldes gatekeeperen på LAN med gk.GetIPAddress. Gatekeeperen håndterer adgangskontrol til LAN'et, og returnerer enten en ACF(Admission Confirmed)- eller en ARJ(Admission Rejected)besked. Hvis adgang til LAN nægtes

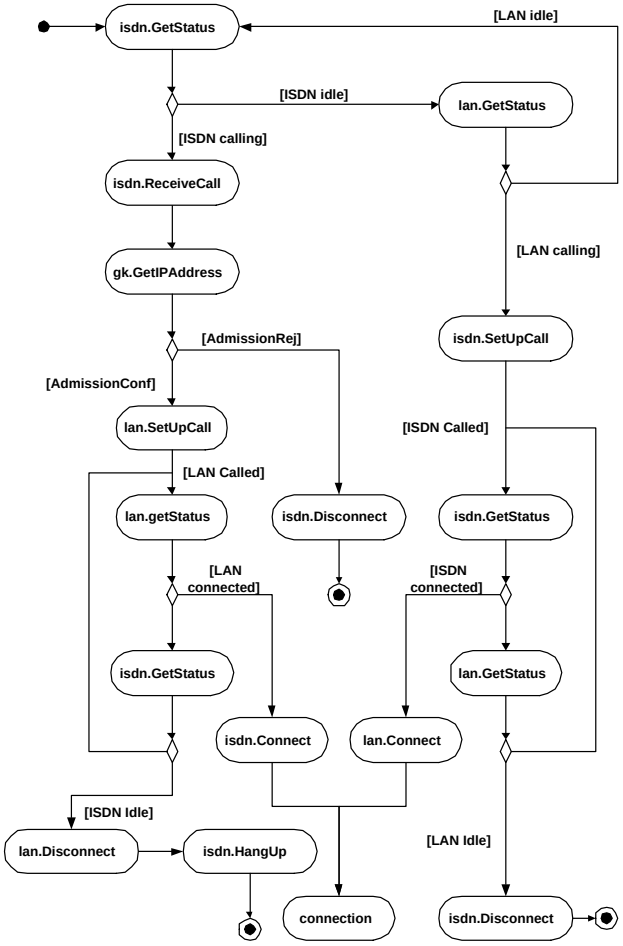
Funktion	Event flow input	Data input	Output
SetupCall()	DTMF_IN ACF	N/A	N/A
Connect()	PSTN_OFF_HOOK CON_B3_ACT_IND	N/A	**status**
Disconnect()	Setup Connect PSTN_ON_HOOK	N/A	**status**
GetStatus()	CONNECT_IND DISCONNECT_IND ReleaseComplete	N/A	**status**

Figur 7.11: Oversigt over funktioner i LAN-klassen

Funktion	Event flow input	Data input	Output
SetupCall()	Setup	CalledPartyNumber CallingPartynumber	**status**
Connect()	Connect	N/A	N/A
Disconnect()	ARJ ReleaseComplete	N/A	**status**
SendData()	**data** (fra LAN)	**data*** DATA_LENGTH	**data*** DATA_LENGTH
ReceiveData()	**data** (fra PSTN)	**data*** DATA_LENGTH	**data*** DATA_LENGTH
GetStatus()	DTMF_IN PSTN_ON_HOOK PSTN_OFF_HOOK	N/A	**status**

Figur 7.12: Oversigt over funktioner i PSTN-klassen

afbrydes forbindelsen til ISDN med isdn.Disconnect. Når der gives adgang til LAN returnerer gatekeeperen sammen med ACF den pågældende terminals IP-adresse.
Gateway'en sender nu en H.225.0-Setup-besked til Terminalen, og venter derefter i en løkke på at forbindelsen er åben. Samtidig checkes der for hang up fra ISDN-siden. Afbrydes forbindelsen herfra kaldes lan.Disconnect og isdn.HangUp. Sidstnævnte svarer på en CAPI_DISCONNECT_IND (disconnect indikation). Når forbindelsen er åben på LAN-siden kaldes isdn.Connect, der afslutter åbningen af forbindelsen på ISDN-siden.
Hvis kaldet udgår fra LAN kaldes isdn.SetUpCall, der indleder kaldsopsætning til ISDN. Ligesom i ovenstående tilfælde ventes der nu på at forbindelsen åbnes. Når det sker kaldes lan.Connect. Hvis forbindelsen til LAN afbrydes før kaldet er sat op til ISDN kaldes isdn.Disconnect.



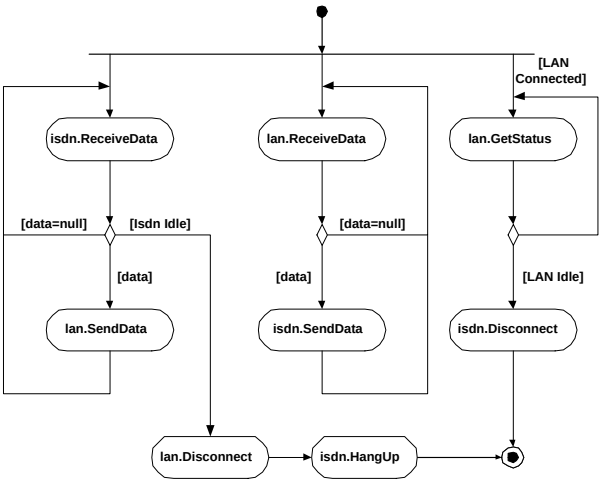
Figur 7.13: ISDN/LAN-opsætning af kald

7.5.2 Dataoverførsel for Gateway

Figur 7.14 er en model af funktionaliteten for Gateway'en ved dataoverførsel. Efter at forbindelsen er oprettet forkes 3 processer, en proces der modtager data fra ISDN og sender til LAN, en proces der modtager data fra LAN og sender til ISDN og en proces, der checker status på signaleringskanalen på LAN'et, og dernæst afbryder ISDN-linjen, hvis forbindelsen afbrydes fra LAN'et. I den førstnævnte proces checkes der for en CAPI_DISCONNECT_IND, der tricker lan.Disconnect og isdn.HangUp. Er der ingen data (**data**=NULL) kaldes ReceiveData igen.

7.5.3 Kaldsopsætning for Terminal

Figur 7.15 viser funktionaliteten for Terminalen ved kaldsopsætning. Funktionaliteten er i store træk den samme som i Gateway'en, hvis man erstatter ISDN-



Figur 7.14: ISDN/LAN-dataoverførsel

forbindelsen med en PSTN-forbindelse. Den eneste forskel er situationen, hvor der ventes på forbindelse til LAN ved opkald fra PSTN, idet der her ikke er brug for nogen procedure der afslutter åbningen af PSTN-linjen (den er åben fra det øjeblik, hvor der laves et opkald). Endvidere er det ikke nødvendigt at svare med HangUp, når forbindelsen afbrydes fra PSTN.

7.5.4 Dataoverførsel for Terminal

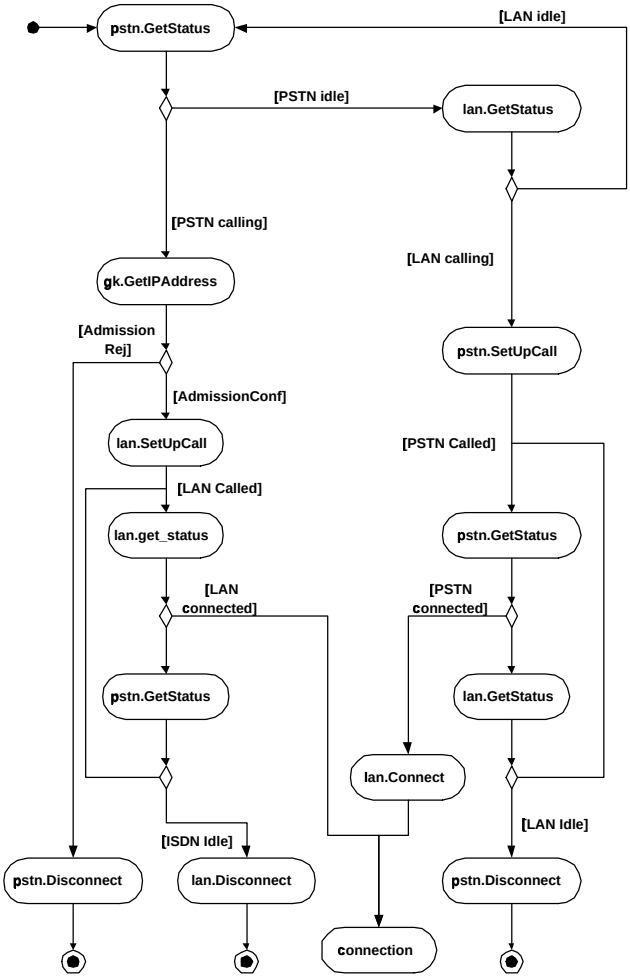
Figur 7.16 er en model af funktionaliteten for Terminalen ved dataoverførsel. Som ved kaldsopsætning er den næsten indentisk med modellen for dataoverførsel for Gateway'en, hvis man erstatter ISDN med PSTN. Undtaget er check for afbrydelse fra PSTN, idet dette check finder sted i samme proces, hvor der checkes for afbrydelse fra LAN.

7.6 Adfærdsanalyse

I det følgende findes en række sekvensdiagrammer, der angiver hvorledes klasserne Gateway, Gatekeeper og Terminal interagerer med hinanden i en række typiske scenarier. Beskederne, der udveksles imellem Gateway og Gatekeeper, imellem Terminal og Gatekeeper og imellem Gateway og Terminal er H.225.0-beskeder.

7.6.1 Kaldsopsætning ved opkald fra ISDN til PSTN

Figur 7.17 viser sekvensen af beskeder, der udveksles imellem Gateway, Gatekeeper og Terminal under kaldsopsætning, ved opkald fra ISDN til PSTN over LAN.



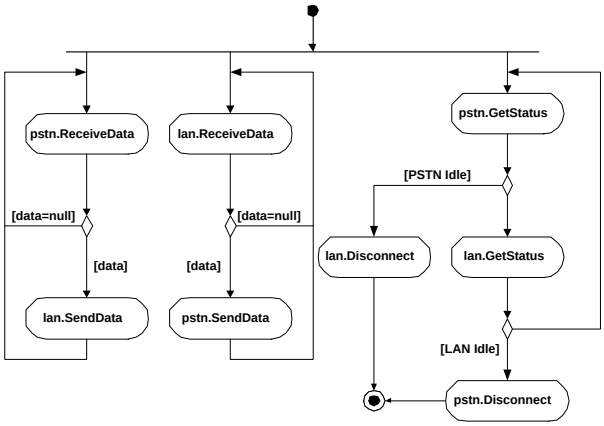
Figur 7.15: PSTN/LAN-opsætning af kald

7.6.2 Terminering af kald fra ISDN til PSTN

Figur 7.18 viser sekvensen af beskeder, der udveksles imellem Gateway og Terminal ved terminering af kald fra ISDN til PSTN.

7.6.3 Kaldsopsætning ved opkald fra PSTN til ISDN

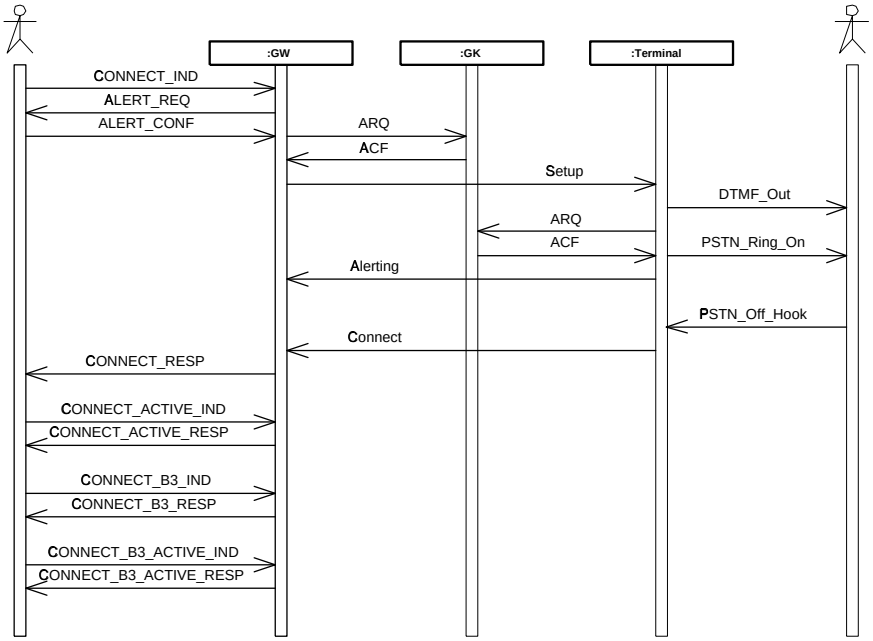
Figur 7.19 viser sekvensen af beskeder, der udveksles imellem Gateway og Gatekeeper, imellem Terminal og Gatekeeper og imellem Gateway og Terminal under kaldsopsætning ved opkald fra PSTN til ISDN.



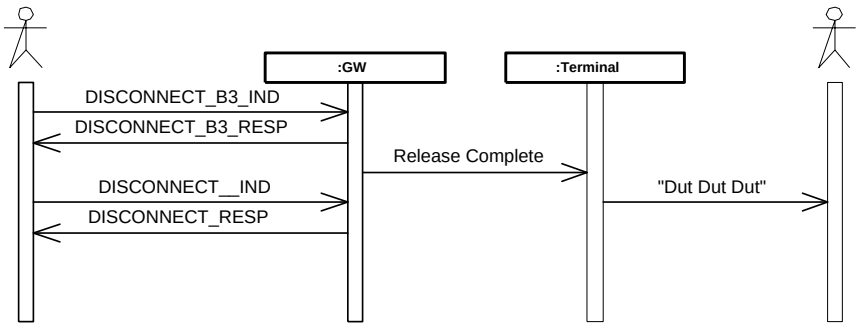
Figur 7.16: PSTN/LAN-dataoverførsel

7.6.4 Terminering af kald fra PSTN til ISDN

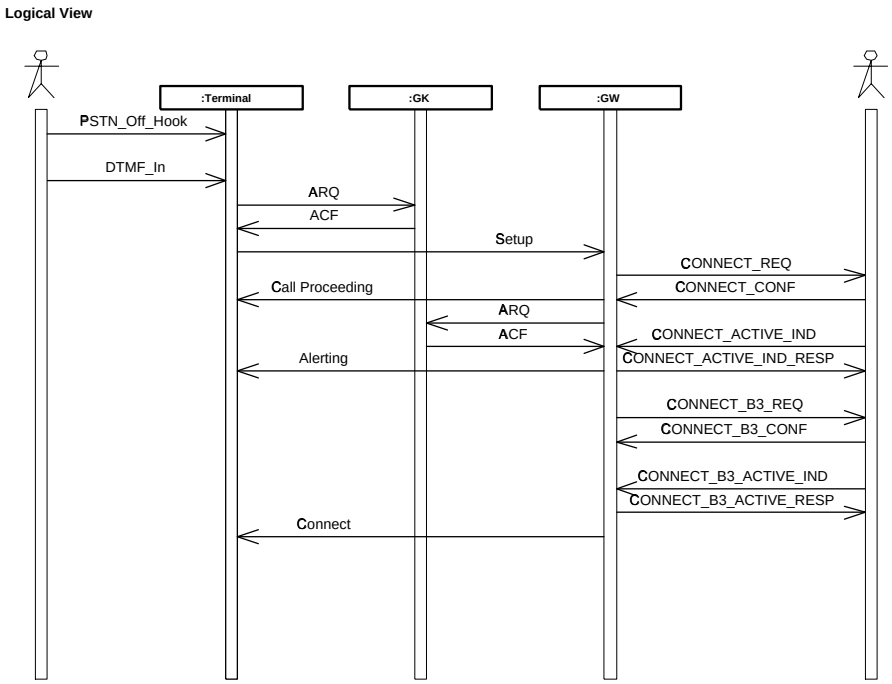
Figur 7.20 viser sekvensen af beskeder, der udveksles imellem Gateway og Terminal ved terminering af kald fra PSTN til ISDN.



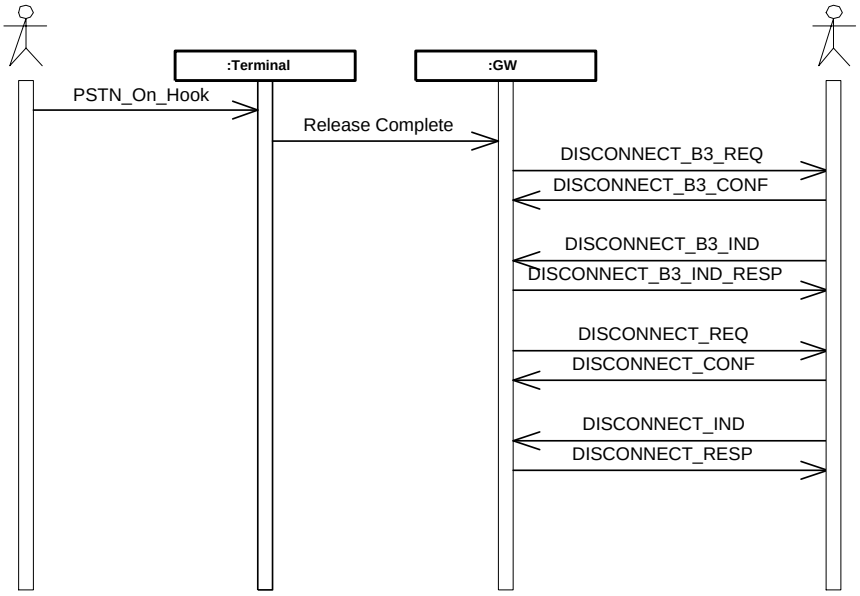
Figur 7.17: Sekvensdiagram for opsætning af kald fra ISDN til PSTN



Figur 7.18: Sekvensdiagram for terminering af kald fra ISDN til PSTN



Figur 7.19: Sekvensdiagram for opsætning af kald fra PSTN til ISDN



Figur 7.20: Sekvensdiagram for terminering af kald fra PSTN til ISDN

Kapitel 8

Softwaredesign

8.1 Indledning

8.1.1 Afgrænsning

I forhold til analysen er der foretaget en yderligere afgrænsning af systemet, således at design ikke omfatter udvekslingen af de i Anbefaling H.225 definerede beskeder til kaldsopsætning, registrering, adgangskontrol og angivelse af status. Funktionaliteten af de enkelte dele, f.eks. LAN-delen, er dog stadig den samme. Denne afgrænsning er foretaget på baggrund af manglende ressourcer. At få et minimumssystem op at køre har haft en høj prioritet.
Konsekvensen er den, at H.323 således ikke overholdes, og at det fremstillede produkt ikke vil være i stand til at kommunikere med andre H.323-enheder.

8.1.2 Prototyper

En række prototyper er blevet fremstillet forud for designfasen med det formål dels at kunne lave et detaljeret, bedre design og dels for at komme problemer i forbindelse med implementationen i forekøbet.
Eksempler på sådanne prototyper er sockets, DirectSound, programmering af parallelport og CAPI. Et eksempel på et problem, vi er kommet i forekøbet, er problemerne i forbindelse med tråde og objektorienteret programmering under Win32 beskrevet i afsnit 8.2.1.

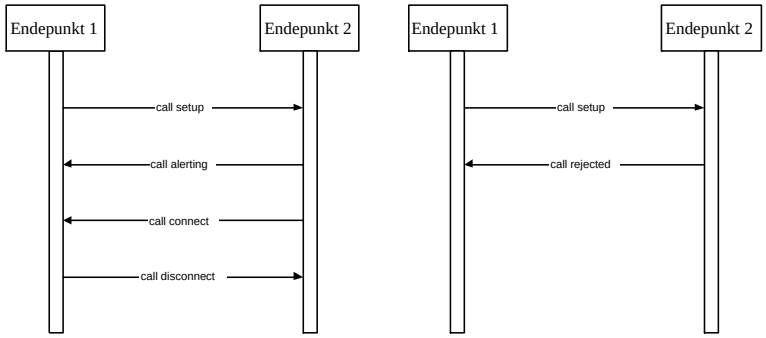
8.1.3 Metode

Designet er set som et raffinement af analysens klassediagrammer og klassespecifikationerne, desuden er beslutningen om konkret procesarkitektur og opdeling i logiske subsystemer (en art komponentarkitektur) (CAPI, PSTN og LAN) taget. Disse subsystemer er i dette kapitel designet hver for sig, og koblingen mellem disse er behandlet i afsnit 8.2.
Ingen designmetode er fuldt slavisk. Elementer fra og overvejelser i forhold til en række udviklingsmetoder indgår i designet, blandt andet CODARTS, OOA&D

samt SPU.

8.2 Generelt

De 2 delsystemer, PSTN- og ISDN-gateway'en, har grundlæggende den samme opbygning. I begge systemer viser analysens sekvensdiagrammer, at en samtale har forløb som vist på figur 8.1



Figur 8.1: Sekvensdiagram af kommunikation mellem to endepunkter. A: succesfuld forbindelse, B: modtageren afviser kommunikation

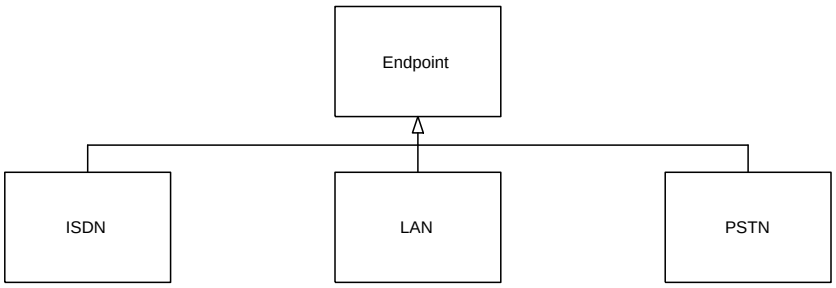
Forløbet er således:

1. Senderen sender en call setup besked med kilde- og destinationstelefonnummer.
2. Modtageren svarer enten med en call alerting eller call rejected besked.
3. Hvis alerting, så sender modtageren en call connect besked når forbindelsen opnåes.
4. Data overføres.
5. Samtalen afsluttes ved at enten senderen eller modtageren sender en call disconnect besked.

Figur 8.1 viser at det ville være hensigtsmæssigt, at give de 3 dele (ISDN, LAN og PSTN) en fælles grænseflade, således at man simpelt kunne udskifte de forskellige dele (f.eks. kunne man forestille sig et system, der kørte over en anden protokolstak, eller gateways til andre systemer end ISDN). Dette implementeres ved at lave en fælles baseklasse for alle endepunkterne, som de hver især nedarver fra. Grafisk ville klassehiearkiet se ud som på figur 8.2.

8.2.1 Samtidighed

Da kommunikationen imellem de forskellige endepunkter (og de forskellige samtaler) foregår asynkront, er det ønskværdigt at designe programmet multitrådet.



Figur 8.2: Endpoint-klassehiearki

Grunden til at der ønskes at bruge multiple tråde, og ikke multiple processer, er den relativt store kommunikation mellem de enkelte programdele.
Desværre er det ikke simpelt at kombinere multitrådethed og OOP under Win32, idet man ikke direkte kan starte eksekveringen af metoder med de indbyggede trådkald. Dette skyldes dels, at startfunktionen i trådkaldet skal kaldes efter _stdcall kaldskonventionen, og at det kun er muligt at overføre én parameter af typen void* til funktionen.
Man kan løse problemet ved at lave en static metode, der kan kaldes ud fra _stdcall kaldskonventionen, idet en static funktion ikke automatisk får overført this-pointeren som første parameter, som er normal member funktion gør. Man kan så overføre this-pointeren som parameter til denne funktion, og i denne funktion typecaste pointeren (der igennem kaldet er blevet til en void pointer) til en pointer til klassen. Ud fra denne pointer kan man så kalde den ønskede metode.
Skal man ændre member variabler inden metoden kaldes, kan man, hvis variablerne er public, ændre disse ud fra this-pointeren. Hvis variablene ikke er public, kan man istedet kalde en anden metode, der har adgang til de private data, og derefter kalde den ønskede metode. F.eks kunne man forestille sig at have en boolsk variabel, der fortalte om en tråd var aktiv. Denne ville man sætte til sand lige inden den ønskede metode blev kaldt, og falsk lige efter den returnede.
Det sidste problem ved trådfunktionerne under Win32 er, at det kun er muligt at overføre een parameter til den kaldte funktion, og da denne bruges til at overføre this-pointeren, er der ingen mulighed for at overføre parametre. Dette problem løses ved at kalde en eller flere metoder, som sætter objectet op, inden man starter tråden.
Problemerne løses ved at lave en trådklasse, som indkapsler det nødvendige arbejde for at kunne starte tråde, der udfører kode i member funktioner. I denne klasse er metoden der udføres når tråden startes op (run metoden) ikke implementeret, og skal derfor implementeres i underklasser af denne. De dele af programmet der skal køre som selvstændige tråde, er derefter realiseret som nedarvninger af denne klasse.

Trådkommunikation

For at de enkelte tråde skal kunne kommunikere med hinanden, er det nødvendigt at hver tråd har en beskedkø, således at man kan signalere en besked fra en tråd til en anden. Denne beskedkø skal have gensidig udelukkelse, idet multiple tråde kan tilgå denne beskedkø samtidig. Problemstillingen med beskedkøen er på mange måder den samme som ved dataoverførsel, så der er brugt den samme buffermekanisme ved begge dele. I det følgende er Dataoverførslen beskrevet.

8.2.2 Dataoverførsel

Der er flere faktorer der spiller ind i design af dataoverførslen.

- For at et multitrådet system skal være effektivt, skal trådsynkronisering holdes på et minimum, idet synkronisering mindsker paralleliteten.
- Der skal være gensidig udelukkelse.
- Idet f.eks. netværket ikke garanterer, at data ankommer i rækkefølge eller med fast rate, er det nødvendigt med en buffermekanisme, for at minimere variationer i datastrømmen.

Denne buffer kan realiseres på flere måder:

- Cirkulær buffer (array)
- Linket liste

Der er fordele og ulemper ved begge muligheder.
Cirkulær buffer:

- Lavt overhead i bufferhåndtering.
- Svært at understøtte variabel pakkestørrelse (til forskellige komprimeringsstandarder).
- Overhead ved flytning af data i forbindelse med buffermekanisme.

Linket liste:

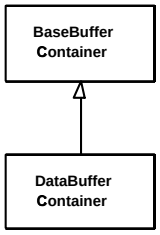
- Større overhead i bufferhåndtering end ved en cirkulær buffer.
- Relativt simpelt at understøtte variabel pakkestørrelse.
- Simpelt at flytte (indsætte datapakke imellem andre datapakker) data i buffer.

Det er ønskværdigt at bufferen har lavt overhead ved indsættelse og flytning af data pga. at man ønsker at smide den ældste datapakke væk hvis bufferen er fyldt. Dette skyldes at de ældste data er de mest forsinkede, og dermed ville forårsage den største latens, hvis den blev sendt videre.

På baggrund af dette er en linket liste valgt. Denne er designet således:
Listen består af to klasser. En containerklasse og en bufferklasse. Containeren er et led i den linkede liste, og bufferen er aggregeringen af alle containerne, dvs. hele listen.

Containeren

Linkede lister kan grundlæggende realiseres på to måder, som dobbelt- eller enkeltlinkede lister. Den enkeltlinkede liste har færre referencer som skal opdateres ved indsættelse eller udtagelse af køen, men da man ikke har en reference til den forrige i køen, bliver man nødt til at søge listen igennem for hver indsættelse eller udtagelse. Da det er ønskværdigt at minimere overhead ved indsættelse og udtagelse, er der valgt et dobbeltlinket design.
For at adskille listehåndteringen fra datarepræsentationen er containeren todelt. Listehåndteringen er implementeret i klassen basebuffercontainer, og datarepræsentationen er implementeret i databuffercontainer-klassen, som er en nedarving af basebuffercontainer'en. Figur 8.3 viser opbygningen.

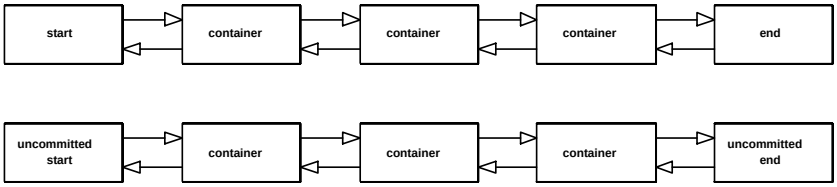


Figur 8.3: buffercontainer-klassehiearki

Basebuffercontainer-klassen indeholder metoder til at insætte og fjerne fra en liste, og manipulere med referencerne fremad og tilbage. Databuffercontainer-klassen indeholder desuden metoder til at manipulere med indsætte og udtage data fra containeren.

Bufferen

Bufferklassen holder styr på listen af containerobjecter. For at kunne bruge de generelle funktioner i basebuffercontainer-klassen, uden at lave specialtilfælde ved enderne af listen, er listen hægtet sammen med specielle endepunktscontainere, istedet for at have en pointer til den første og den sidste container i listen. På denne måde har en container altid en forrige og en næste, og man skal ikke tjekke for null-pointere ved indsættelse og udtagelse. Desuden kan man have en tom liste, hvilket ellers er problematisk ved et konventionelt listedesign.
Skematisk er listen opbygget som på figur 8.4:
Af figur 8.4 ses det, at buffer-klassen indeholder to lister: Den egentlige liste og en liste over ledige containere. Dette er nødvendigt, fordi en liste bliver initialiseret med et fast antal containere, som placeres i listen over ledige containere.



Figur 8.4: Opbygning af linket liste

Ligeledes bliver en container indsat i listen over ledige containere, når den er blevet tømt.

Gensidig udelukkelse

Da et buffer-objekt kan blive tilgået af to eller flere tråde på samme tid, er det nødvendigt at sikre gensidig udelukkelse. Dette kan sikres på 3 niveauer:

1. For hele buffer-objektet.
2. For hver liste for sig.
3. For hver container for sig.

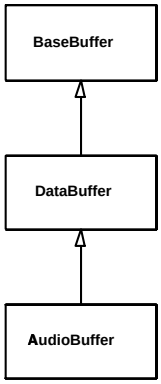
Løsning 1 giver lav parallelitet, idet kun en tråd kan tilgå hele buffer-objektet af gangen. Løsning 3 er meget kompleks, idet mange kritiske regioner skal arbejde sammen, med risiko for deadlocks og synkroniseringsproblemer. Løsning 2 er en fornuftig mellemting, og er derfor valgt. Til at opnå gensidig udelukkelse er anvendt kristiske regioner, der stilles til rådighed af Win32.

Deadlocks

På grund af gensidig udelukkelse er der risiko for deadlocks. For at forhindre dette i at opstå forhindres en af de fire betingelser for at deadlocks kan opstå [Stalling, 1992, s. 260].
Den simpleste betingelse at undgå er cirkulær ventning. Dette kan undgås ved at definere en fast rækkefølge for ressourceallokering, eksempelvis listen af ledige containere førend den anden liste. Hvis denne rækkefølge altid overholdes kan der ikke opstå deadlocks.

Opbygning

Ligesom buffercontainer-klasserne er buffer-klasserne opdelt i et hiearki som vist på figur 8.5.
Øverst er BaseBuffer-klassen, der analogt med BaseBufferContainer-klassen står for det grundlæggende listesystem, inklusiv håndtering af gensidig udelukkelse. Nedenunder denne klasse findes DataBuffer-klassen, der står for håndtering af data ind og ud af listen. Metoden for indsættelse af data vil fejle hvis der ikke er nogle ledige containere til rådighed. Under denne klasse er AudioBuffer-klassen,

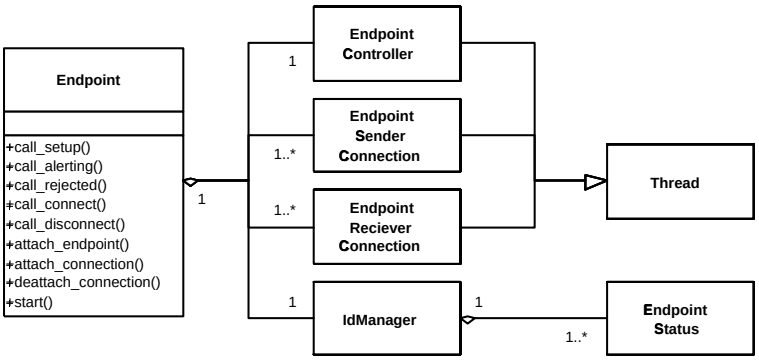


Figur 8.5: buffer-klassehiearki

der redefinerer indsættelsesmetoden, således at den ældste container vil blive kasseret, i tilfælde af at bufferen bliver fyldt. Databuffer-klassen bliver brugt til beskedkø i trådklassen, og audiobuffer-klassen benyttes til dataoverførslen.

8.2.3 Endpoint-klassen

Ud fra designet af trådkommunikationen og dataoverførslen kan endpointklassen designes. Figur 8.6 viser klassediagrammet for endpointklassen.



Figur 8.6: endpointklasse-diagram

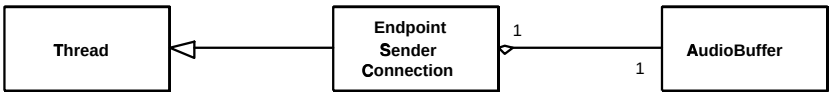
ID

Som beskrevet i afsnit 8.1, foregår kontrolkommunikationen mellem to endepunkter via beskeder. For at kunne have flere samtaler kørende på samme tid, er det nødvendigt at kunne skelne mellem, hvilke samtaler de forskellige kontrolbeskeder er til. Til dette benyttes sender- og modtagernummer, idet dette er unikt for hver samtale. Dog bør starttidspunktet for samtalen også tages

med, idet senderen kan have hemmeligt nummer, og man derfor ikke vil kunne skelne mellem samtidige opkaldsforsøg fra forskellige telefoner til samme nummer. Istedet for at sende nummeret med igennem hele sekvensen, bliver der, hos modtageren af call setup beskeden, lavet et unikt id for samtalen, som benyttes i resten af sekvensen. Dette id placeres i IdManageren, således at controlleren kan finde de tilknyttede connectionobjekter til en samtale.

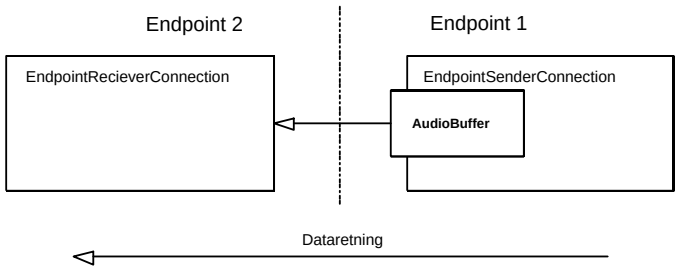
Opbygning

Endpointklassen er overordnet en aggregering af fire dele, en EndpointController, EndpointSenderConnection, EndpointRecieverConnection og en IdManager. Desuden er der to underdele, CodecManageren og EndpointStatus. Disse dele har hvert sit ansvarsområde:
EndpointController: Modtager og reagerer på kontrolbeskeder fra det andet endepunkt og mellem de enkelte dele i endpoint-klassen (se figur 8.6). Controlleren skal kunne håndtere opsætning / nedlægning af flere samtidige samtaler på een gang i samme tråd. Denne løsning er valgt for at minimere antallet af tråde, idet disse altid vil give et vist overhead.
Controlleren nedarver fra trådklassen, og har dermed også en beskedkø. Controlleren skal hente beskeder fra beskedkøen, identificere hvilken samtale beskeden tilhører, og sende beskeden videre til det pågældende EndpointStatus-objekt, som behandler beskeden (se senere). Internt i endpoint-klassen skelnes mellem de forskellige samtaler ud fra det tilknyttede telefonnummer, hvorimod beskeder fra et andet endepunkt skelnes på det tilknyttede id.
Da Controller-objekterne i nedarvningerne til endpoint (ISDN,PSTN og LAN) ikke kommunikerer igennem samme grænseflade (parallelport, CAPI og socket), er EndpointController-klassen ikke implementeret, idet der ikke findes nogen fælles kodebasis eller interface.

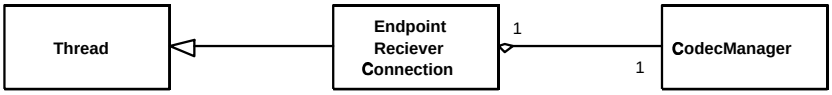


Figur 8.7: EndpointSenderConnection klassehiearki

EndpointSenderConnection: Står for overførslen af en enkelt samtale's data fra dette endepunkt til et andet endepunkt. Som figur 8.7 viser, arver klassen af trådklassen, og har derfor en beskedkø, hvorigennem signaleringen foregår. Hvert EndpointSenderConnection-objekt indeholder en AudioBuffer (se afsnit 8.2.2). Denne buffer benyttes som mellemled mellem et EndpointSenderConnection-objekt og en EndpointRecieverConnection-objekt i det andet endepunkt. Figur 8.8 viser en skematisk oversigt af denne kobling.
EndpointRecieverConnection: Står for overførslen af en enkelt samtale's data til dette endepunkt. EndpointRecieverConnection-objektet's tråd henter data fra Audiobufferen placeret i det tilknyttede EndpointSenderConnection-objekt. Inden dataene sendes videre bliver de, hvis nødvendigt, konverteret af



Figur 8.8: EndpointSenderConnection til EndPointRecieverConnection forbindelse



Figur 8.9: EndpointRecieverConnection klassehiearki

en CodecManager, som hvert reciever-objekt indeholder, hvilket kan ses af figur 8.9. grunden til, at CodecManager'en er placeret i EndpointRecieverConnection-klassen, og ikke i EndpointSenderConnection-klassen, er at det ville være formålsløst at bruge tid på konvertering af lyddata, hvis det blev kasseret i AudioBuffer (hvilket vil ske, hvis recieveren ikke er hurtig nok til at udtage data, se afsnit 8.2.2). For at oprette forbindelsen til EndpointSenderConnection-objektet, kaldes den modstående endepunkt's attach_buffer metode med samtalens id.

CodecManager: CodeManager-klassen står for konvertering mellem forskellige audioformater, hvilket i vores system begrænser sig til konvertering mellem G.711 og 16bit upakket lyddata. Denne konvertering sker ved hjælp at et enkelt opslag i en tabel per sample.

Figur 8.10: IdManager klassehiearki

IdManager: Står for oversættelse mellem id og telefonnummer eller EndpointRecieverConnection/EndpointSenderConnection-objekt. Desuden er der en metode til at indsætte et nyt id i listen ud fra et specificeret telefonnummer. Denne metode vil fejle hvis den valgte linje er optaget. IdManager-klassen nedarver af Databuffer-klassen, og opnår derfor samme mutual exclusion som denne (se figur 8.10). IdManageren indeholder et EndpointStatus-objekt per linje, således at den kender status for hele endepunktet.

EndpointStatus: EndpointStatus-klassen er en form for en statemaskine for en samtale. Når den modtager et event (dvs. en besked), behandles beskeden, og et tilstandsskift kan forekomme. Klassen kan spørges om hvilken tilstand den er i, hvilket benyttes af IdManageren.

Grænseflade

Grænsefladen til endpointklassen ses på figur 8.6, og vil i det følgende blive beskrevet:

call_setup(source_number,destination_number): Når call_setup-metoden kaldes beregnes et id for samtalen, id'et forsøges indsat i IdManager'ens liste, og hvis linjen ikke er optaget, insættes beskeden i controlleren's beskedkø. Hvis linjen er optaget, kaldes istedet call_reject på det andet endepunkt. Følgende pseudokode viser sekvensen:

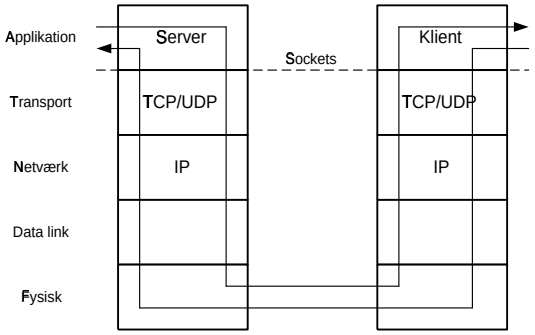
```
call_setup(source_number,destination_number)
{
  id = create_id(source_number,destination_number,current_time);
  if (IdManager.insert_id(id,destination_number))
        controller.send_message(MSG_CALL_SETUP,id);
  else
        endpoint.call_reject(id);

  return id;
}
```

call_alerting(id),call_rejected(id), call_connected(id), call_disconnected(id): Disse funktioner har det til fælles at de laver en besked, som indsættes i controlleren's beskedkø.
attach_endpoint(endpoint*): Denne metode sætter referencen til det andet endepunkt op. Dette bliver ikke gjort i constructoren, fordi man så ville have en cirkulær reference, hvor endpoint 1 krævede endpoint 2 for at kunne startes, og hvor endpoint 2 krævede endpoint 1.
attach_connection(id), deattach_connection(id): Attach_connection metoden finder EndpointSenderConnection-objektet der svarer til id'et ud fra IdManageren. Herefter hentes adressen på AudioBuffer-objektet i EndpointSenderConnection-objektet. Deattach_connection metoden fortæller EndpointSenderConnection-objektet at der ikke længere er referencer til bufferen, og man derfor godt kan deallokere denne hvis nødvendig.
start(): Start metoden er beregnet til at starte de nødvendige tråde op. Da dette er forskellige for hver underklasse af endpoint-klassen, er denne metode ikke implementeret i baseklassen.

8.3 LAN

8.3.1 Indledning

Det er netværksdelen af systemets opgave, at opsætte kald over netværk samt at transmittere audiosignaler over netværket fra et I/O-objekt til et andet I/O-objekt.



Figur 8.11: TCP/IP/OSI-referencemodel hybrid-repræsentation af dataflow i netværksdelen af systemet.

Netværksdelen grænser, som det ses på figur 8.11 nedadtil op mod TCP/IP-stakken.
En række prototyper er blevet fremstillet som en del af foranalysen med henblik på at kunne lave et socket-nært design, der ville minimere tilpasning under implementationen.

8.3.2 Klient-server

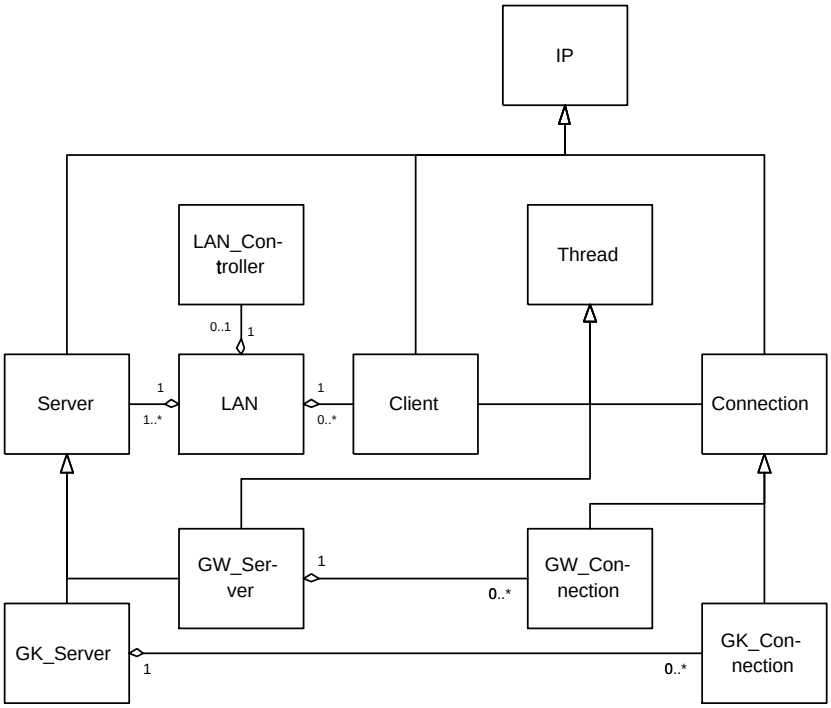
Netværksapplikationer anvender klient-server paradigmet [Comer, 1999, s. 325], og dette har afgørende betydning for designet af netværksdelene af systemet.
En server, det stiller en service til rådighed, venter på kontakt fra en klient. Hvis serveren, som i dette tilfælde, skal være i stand til at håndtere flere forbindelser samtidigt, er der tale om en samtidig server (concurrent server).
Der skal altså være en server for hver enhed på netværket, der skal være i stand til at reagere på forespørgsler over netværket. Nærmere bestemt en for hver H.323-enhed, dvs. gateway, gatekeeper og terminal.
Klienterne instantieres for hvert opkald fra PSTN eller ISDN, altså "udefra". Disse kontakter så serverne.

8.3.3 Sockets

Grænsefladen til TCP/IP-stakken, vi har valgt, er socket-API'en, og implementationen, der er anvendt, er Winsock 1.1.
Alle socket-kald er anvendt blokerende.

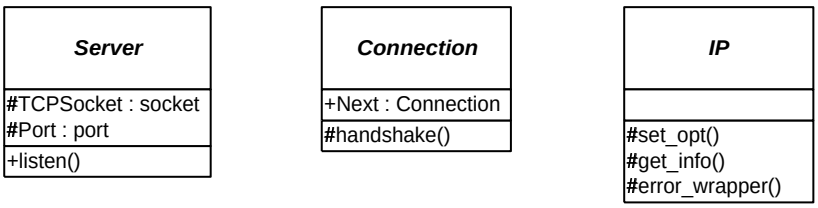
8.3.4 Klasser

Figur 8.12 viser et raffineret klassediagram for netværksdelene af systemet.



Figur 8.12: Raffineret klassediagram.

De to Server-specialiseringer, GK_Server og GW_Server, rummer hver en række Connection-objekter (GK_Connection og GW_Connection henholdsvis), der repræsenterer en tråd, der håndterer en forbindelse mellem en server og en klient.
Thread-klassen (se afsnit 8.2.1) gør det muligt for en specialiseret klasses metoder at køre som en selvstændig tråd.
De tre klasser IP, Server og Connection er abstrakte klasser, dvs. de instatieres ikke i sig selv. Disse ses på figur 8.13.



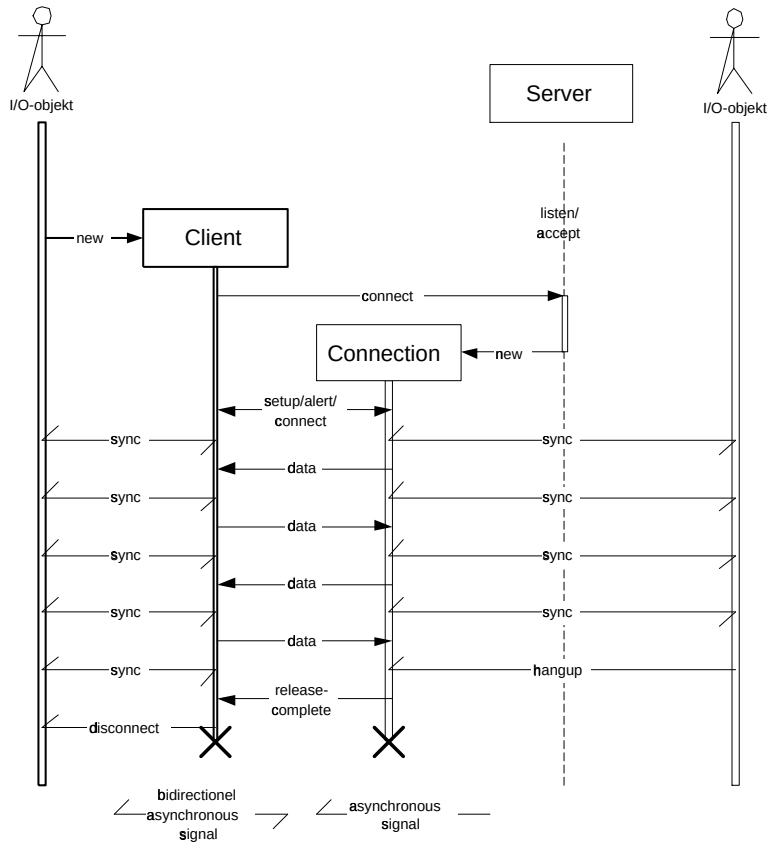
Figur 8.13: Abstrakte klasser.

IP er en hjælpeklasse, der stiller en række metoder til rådighed, og er som

sådan ikke en logisk klasse. Metoderne anvendes i en række klasser igennem nedarvning.
set_opt bruges til at sætte pågældende socket-options, f.eks. ToS eller QoS. Det er således kun i denne metode, at ændringer i forhold til anvendt version af winsock skal foretages.
Serverklassen rummer en metode (listen) til at lytte på en TCP-socket og herefter fork'e en tråd, repræsenteret ved klassen Connection, for hver forbindelse. Serverklassen udgør skallen af de samtidige servere, der indgår i en H.323-zone.

8.3.5 Kobling og instantiering

Figur 8.14 viser sekvensdiagram for netværksdelene af systemet. Det viste diagram tager udgangspunkt i et opkald fra ISDN.
Alle detaljer angående kaldsopsætning (handshake) og kobling mellem LAN-delene og PSTN/ISDN-delene er ikke medtaget. Disse er angivet i analysen.



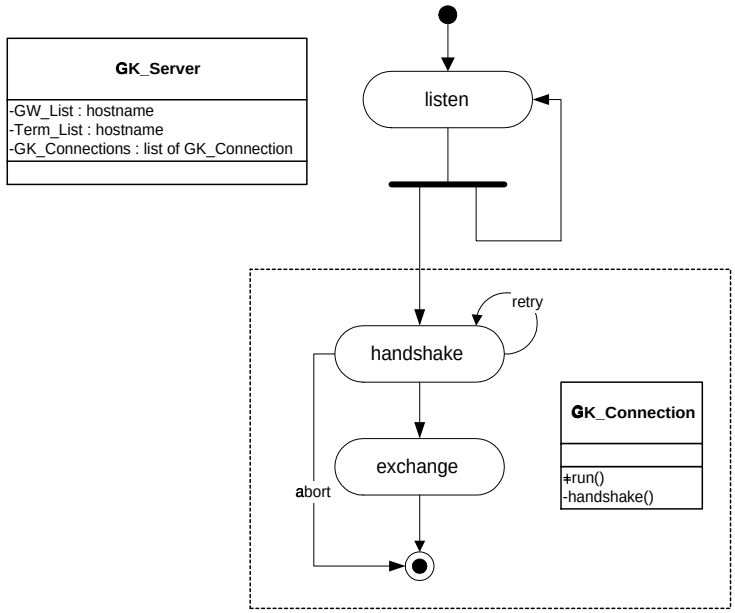
Figur 8.14: Sekvensdiagram visende instantiering af objekter og koblingen mellem disse.

Diagrammet viser principperne for instantiering af objekter af klasserne specifi-

ceret i dette dokument samt hvilke objekter, der er sammenkoblede. Bemærk at kommunikationen (adresseoversættelse/adgangskontrol) med Gate Keeper'en er udeladt.
Et opkald indløber fra ISDN-siden (venstre I/O-objekt). Et klient-objekt instantieres og dette kontakter PSTN-Serveren over netværket. Denne instantierer et Connection-objekt (en af specialiseringerne heraf), hvis metode run() afvikles som en selvstændig tråd. Connection-objektet er koblet til PSTN-I/O-objektet. Connection og Client-objekterne destrueres efter endt samtale.
Koblingen mellem I/O-objekterne og henholdsvis Client og Server/Connection er virtuel. Den sker i praksis igennem instanser af LAN_Controller og PSTN-Controller-klassen.
Al kaldsopsætning og lignende foretages over TCP, hvorimod transmission af audio-signalerne sker over UDP (se afsnit 4.2.1).

8.3.6 Tilstande

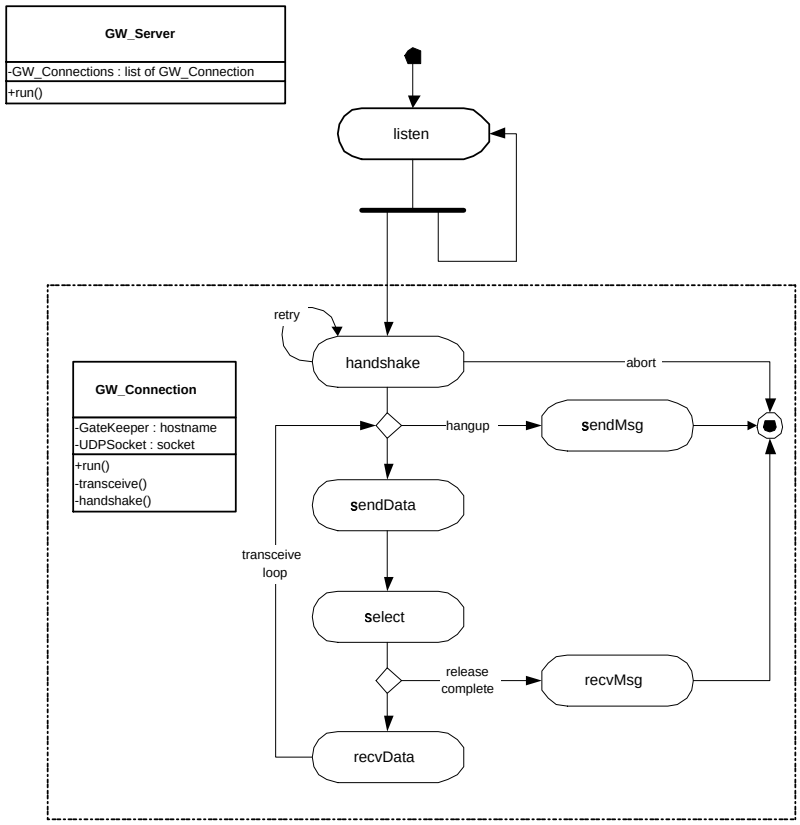
Aktivitetsdiagrammet for Gate Keeper'ens Server- og Connection-klasser ses på figur 8.15.



Figur 8.15: Aktivitetsdiagram for GK_server::listen-metoden og dennes brug af GK_Connection::run samt yderlige specifikation af klasserne Gate Keeper-Server og -Connection.

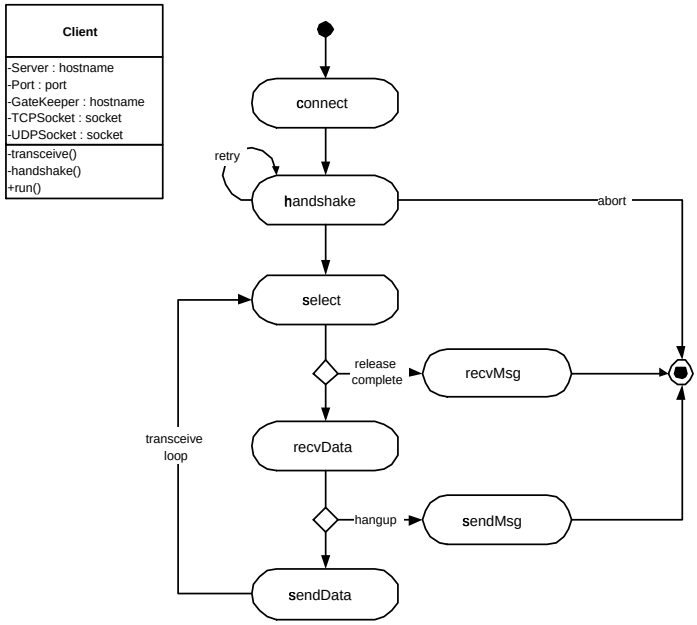
run-metoden i GK_Connection-klassen er metoden der gennem nedarvningen af Thread-klassen afvikles som ny tråd ved kald af start.
handshake-tilstanden imellem Client-klassen samt de to Connection-specialiseringer udgør udvekslingen af f.eks. Setup-, Alerting- og Connect-beskeder.

Gate Keeper'eren kommunikerer udelukkende ved hælp af TCP.
På figur 8.16 ses aktivitetsdiagrammet for GW_Connection- og GW_Server-klassen.



Figur 8.16: Aktivitetsdiagram for GK_Server::run og GK_Connection::run samt yderlige specifikation af Gateway Server- og Connection-klasserne.

sendData, recvMsg, recvData og sendMsg-løkken udgør transceive-metoden i GW_Connection og Client-klasserne. recvData- og sendData-tilstandene udgør overførslen af audiosignaler over UDP.
recvMsg/sendMsg-tilstandene repræsenterer udvekslingen af kontrol-beskeder, f.eks. ReleaseComplete i forbindelse med hangup på ISDN- eller PSTN-siden.
Anvendelsen af select-tilstanden bevirker, at applikationen ikke blokerer evigt i forbindelse med ophør i transmissionen af audiosignaler, da en kontrol-besked altid vil blive modtaget. Dette opnåes implementationsspecifikt ved hjælp af select-kaldet, som er en del af Socket API'et.
Yderligere specifikation af samt aktivitetsdiagram for Client-klassen er at finde på figur 8.17.
Bemærk at tilstandene recvData og sendData i Client-klassen indtræffer i modsat rækkefølge af Server-klassen.



Figur 8.17: Aktivitetsdiagram for Client::connect samt yderlige specifikation af Client-klassen.

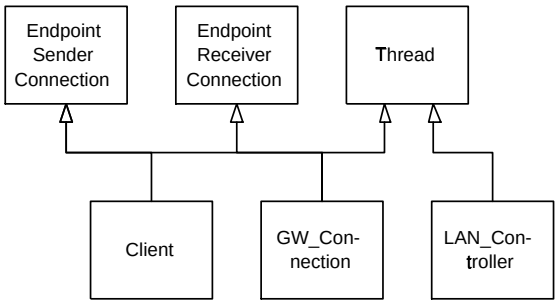
8.3.7 Tråde

En række problemstillinger introduceres ved anvendelsen af parallelitet i form af multi-trådede programmer.
accept-kaldet returnerer en ny socket, og denne skal så videreregives som parameter til en ny-instantieret Connection, der kører som en selvstændig tråd. For at sikre, at denne socket ikke overskrives af en ny connect fra en klient, placeres denne i en integer, der allokeres vha. new for hver connect. Pointeren til denne gives så som parameter til den nye Connection-instans, hvori pladsen frigives.
Der er et en til en forhold mellem instanser af Connection-specialiseringerne og kørende tråde. Connection-instanserne ordnes i en linket liste. Dette er nødvendigt for at de enkelte Connection-instanser ikke havner i garbage collection eller pladsen ikke frigives efter brug. For hver tilføjelse af en ny connection gennemløbes de eksisterende og de inaktive objekter slettes. Tilføjelse og gennemløb (check for inaktive objekter) foregår under mutex ved hjælp af CRITICALSECTION.

8.3.8 IPC

Beskeder og audiosignaler skal sendes mellem de enkelte dele af systemet. Klasserne til håndteringen af dette er specificeret i 8.2.3.
Server-objekter og de tilhørende instanser af specialiseringerne af Connection-

klasserne kommunikerer ikke med hinanden men instantieres og nedlægges af Server-objekterne.
Figur 8.18 viser nedarvningerne, der ligger til grund for IPC'en i systemet.



Figur 8.18: Nedarvningerne der muliggører IPC.

Beskedkøerne, der udgør kanalerne for udveksling af beskeder imellem delene af en H.323-zone, og operationer i forbindelse med disse, nedarves igennem Thread-klassen.
Alle beskederne til LAN-delen (fra PSTN eller ISDN-delene) modtages af LAN_Controller-instansen indeholdt i LAN-objektet. LAN_Controller-klassens opgave er så ud fra opkalds-ID'en at fordele beskederne ud til henholdsvis instanserne af GW_Connection- og Client-klassen. Designet af Thread-klassen kan findes i afsnit 8.2.1.
Nedarvningen af EndpointReceiverConnection- og EndpointSenderConnection-klasserne sætter GK_Connection- og Client-klassen i stand til at modtage og sende audiosignaler.

8.4 RTP design

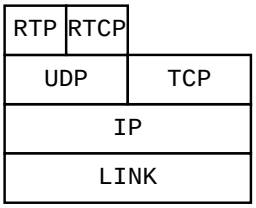
Følgende indeholder en beskrivelse af, hvordan RTP/RTCP designes og tænkes implementeret. Først gives en kort gennemgang af et ideelt design, hvorefter det aktuelle design beskrives, med henblik på hvilke dele der er implementeret og hvilke, der er udeladt.

8.4.1 RTP/RTCP protokol

RTP er en transport protokol for real-tids data, forstået på den måde at RTP er velegnet til at transportere real-tids data. Egnetheden ligger i RTP's indbyggede mekanismer til bl.a. tidsstempling, synkronisering, detektion at pakke tab mm. RTP er ikke selv istand til direkte at transportere data, hvilket klares af, det i OSI stakken, underlæggende lag, f.eks TCP og UDP.
I forbindelse med H.323 implementationer benytter RTP sig af UDP som transportprotokol. Et IP datagram, med RTP data ser ud som figur 8.19.
Selve TCP/IP stakken, med RTP over UDP, ser ud som vist i figur 8.20, hvor RTP somsagt transporteres via UDP, der ligger oven på selve IP, som er det

Figur 8.19: Data pakke bestående af IP header, UDP header, RTP header samt payload.

nederste lag inden linklaget i TCP/IP stakken.

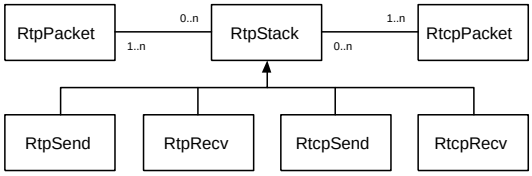


Figur 8.20: TCP/IP protokol stak set i forhold til RTP og RTPC .

Den ideelle tilgangsvinkel til design og implementation af RTP må derfor basere sig på stak princippet. Det vil sige at RTP designes som et lag til OSI stakken, der ligger lige over UDP. RTP laget skal dermed stille den nødvendige funktionalitet til rådighed for at kunne sende og modtage RTP data mellem 2 maskiner. Dermed bliver det RTP lagets opgave at snakke med UDP og IP, formentlig gennem sockets (WINSOCK).

8.4.2 Ideel design

En mulig ideel opbyging af RTP protokollen kan laves som vist på klassediagrammet (figur 8.21).



Figur 8.21: Klassediagram for en mulig ideel RTP/RTCP implementation.

Overordnet beskrevet er RTP protokollen opbygget af følgende klasser.

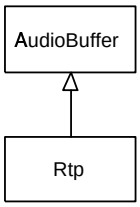
- RtpStack: Superklasse, som primært implementere basale kald til de underlæggende lag i OSI stakken, gennem sockets biblioteket (WINSOCK).
- RtpSend og RtcpSend: Specialiseringer af RtpStack klassen, der implementerer RTP/RTCP pakke sender.
- RtpRecv og RtcpRecv: Specialiseringer af RtpStack klassen, der implementerer RTP/RTCP pakke modtager.

- RtpPacket og RtcpPacket: Associationer til RtpStack, der stiller metoder til selve formateringen af RTP pakkerne til rådighed. Eksempelvis hukommelsesallokering, opsætning af header, men også formateringen af sender og modtager rapport (RTCP) se evt. appendiks A om RTP.

Den specifikke implementation af RTP protokollen, er ret omfattende, hvorfor der i det følgende ses på et "hack", der stiller de allermest nødvendige elementer fra den samlede RTP protokol til rådighed.
I første omgang ses der på kontrol delen af RTP, nemlig RTCP. RTCP er jævnfør appendiks A om RTP en kontrol protokol, der har til formål at sende tilstands rapporter om modtage- og sendeforhold for RTP data pakker. Disse rapporter kan benyttes til f.eks. adaptiv ændring af kodec, men det er ikke et krav. Derfor vælger vi at undlade implementation af RTCP, for istedet at fokusere på de dele af RTP der er mere nødvendige.

8.4.3 Aktuel design

RTP's vigtigste elementer er tidsstempling og synkroniserings mekanismerne. Informationerne herfor ligger i headeren, hvilket betyder, at der princippielt kun er brug for metoder til at pakke en header om en mængde data, samt pakke samme ud igen. Udpakningen af data skal dog ske til en slags jitterbuffer, hvor det er muligt at sortere datapakkerne, idet der ikke er garanti for at IP datagrammer bliver transporteret samme vej gennem et netværk, og derfor ikke kommer til modtageren i samme rækkefølge som de forlod afsenderen.
Klassebeskrivelsen for RTP protokollen indeholder dermed en RTP klasse, der arver generelle buffermekanismer fra AudioBuffer klassen (se afsnit 8.2.2. AudioBuffer klassen redefineres, således at den funktion, der lægger data i bufferen bliver i stand til at sortere data efter sekvensnummeret i RTP headeren. Udover sortering skal funktionen også sørge for at forsent ankommet data ikke afspilles. Figur 8.22 viser det aktuelle klassediagram for RTP. Selve RTP klassen stiller



Figur 8.22: Klassediagram for RTP protokol.

to grundlæggende funktioner til rådighed. En til at pakke en RTP header på en mængde data, og en til at pakke "payload" ud af en RTP pakke.
På den måde bliver RTP implementationen en form for formaterings funktion, der kan kaldes på en mængde data, man ønsker at sende som RTP data eller har modtaget som RTP data. Ved at foretage designet på denne måde, er det ikke umiddelbart muligt at anskue RTP som et lag i OSI stakken.

Legitimeringen af ovenstående skal findes i omfanget af en fuld RTP implementation, der efter projektgruppens mening er for stor, set i forhold til udbyttet heraf.

Kapitel 9

PSTN design

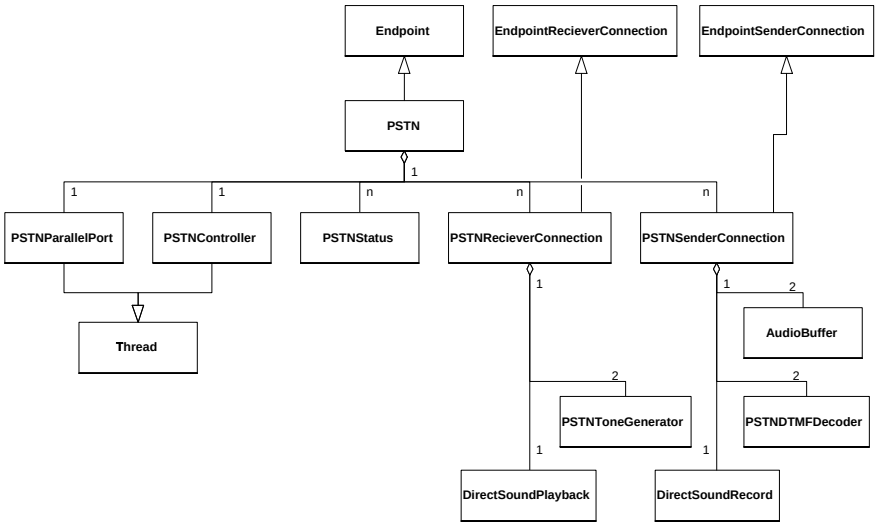
PSTN-delen af systemet er opbygget udfra den i afsnit 8.2.3 beskrevne endpoint-struktur. Den består yderst af et PSTN-objekt, som nedarver Endpoint-klassens funktionalitet (IdManageren samt attach- og beskedkø-funktionerne). PSTN-objektet indeholder følgende:

- PSTNParallelPort: Sørger for at sende ringetone samt overvåge telefonrørs tilstand fra PSTN-enhederne. (kører i egen tråd)
- PSTNController: Som EndpointController. Tager beskeder ud af sin beskedkø, identificerer beskederne vha. IdManageren og sender dem videre til det rigtige PSTNStatus-objekt. (kører i egen tråd)
- PSTNStatus: Som EndpointStatus. Der er et PSTNStatus-objekt for hver PSTN-enhed. Objektet skal sørge for at sende beskeder til andre objekter afhængigt af PSTN-enhedens status og den modtagne besked.
- PSTNRecieverConnection: Der er eet PSTNRecieverConnection-objekt for to PSTN-enheder. Dette skyldes brugen af stereolydkort. Objektet kan hente lyddata og afspille disse vha. DirectSoundPlayback-objektet. Objektet kan også sættes til at generere PSTN-toner vha. et PSTNToneGenerator-objekt for hver kanal. Klassen arver fra EndpointRecieverConnection-klassen.
- PSTNSenderConnection: Objektet kan indspille lyddata vha. DirectSound-Record-objektet og sætte dem i en AudioBuffer. Objektet kan også sættes til at dekode DTMF-taster vha. et PSTNDTMFDecoder-objekt for hver kanal. Klassen arver fra EndpointSenderConnection

Opbygningen ses ilustreret i klassediagrammet på fig 9.1.

9.1 PSTN-klassen

Start er implementeret, således at PSTNControlleren, alle PSTNSenderConnection- og PSTNRecieverConnection-objekter bliver allokeret og deres tråde startet.



Figur 9.1: PSTN-klassens struktur vist i et klassediagram.

Resten af PSTN-klassen nedarves direkte fra Endpoint-klassen.

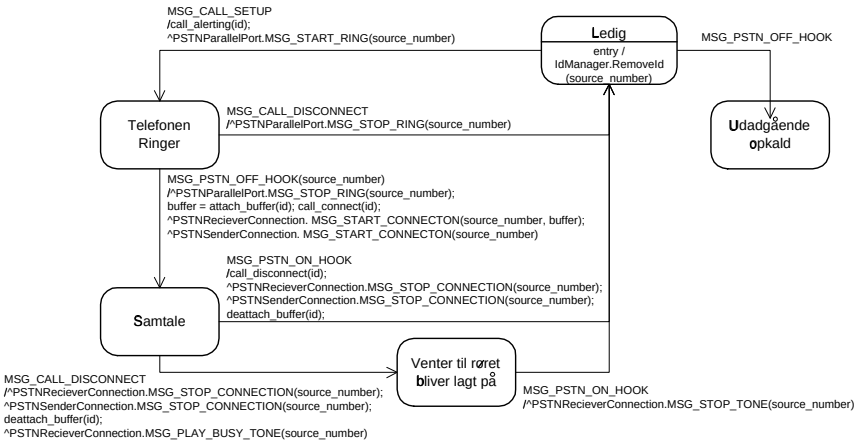
9.1.1 PSTNController-klassen

PSTNController-klassen, har samme funktionalitet, som beskrevet i afsnit 8.2.3. Beskeder ud af beskedkøen og via IdManageren identificeres det til beskeden hørende PSTNStatus-objekt, og beskeden sendes videre til dette objekt.

9.1.2 PSTNStatus-klassen

PSTNStatus-klassen fungerer som beskrevet i afsnit 8.2.3 som en tilstandsmaskiner for hver samtale.

Tilstandsmaskinens tilstande er vist i to tilstand-diagrammer for at øge overblikket. Figur 9.2 viser tilstande for indgående opkald, figur 9.3 viser tilstande for udadgående opkald. På diagrammerne er kan handlinger forårsaget af tilstandsskift have to former. Den ene form er et kald til det modsatte endepunkt eller et kald til et objekt i PSTN-endepunktet. Den anden form er beskeder, som indsættes i den relevante instans af den angivne klasses beskedkø. Den relevante instans kendes, da der er en direkte mapning et mellem PSTNStatus-objekt og de objekter det skal snakke med. source_number på diagrammerne er det telefonnummer, som PSTNStatus-objektet tilhører. Nogle af beskederne på figurene hører til klasser, som ikke har været omtalt endnu, men bliver omtalt i de følgende afsnit.



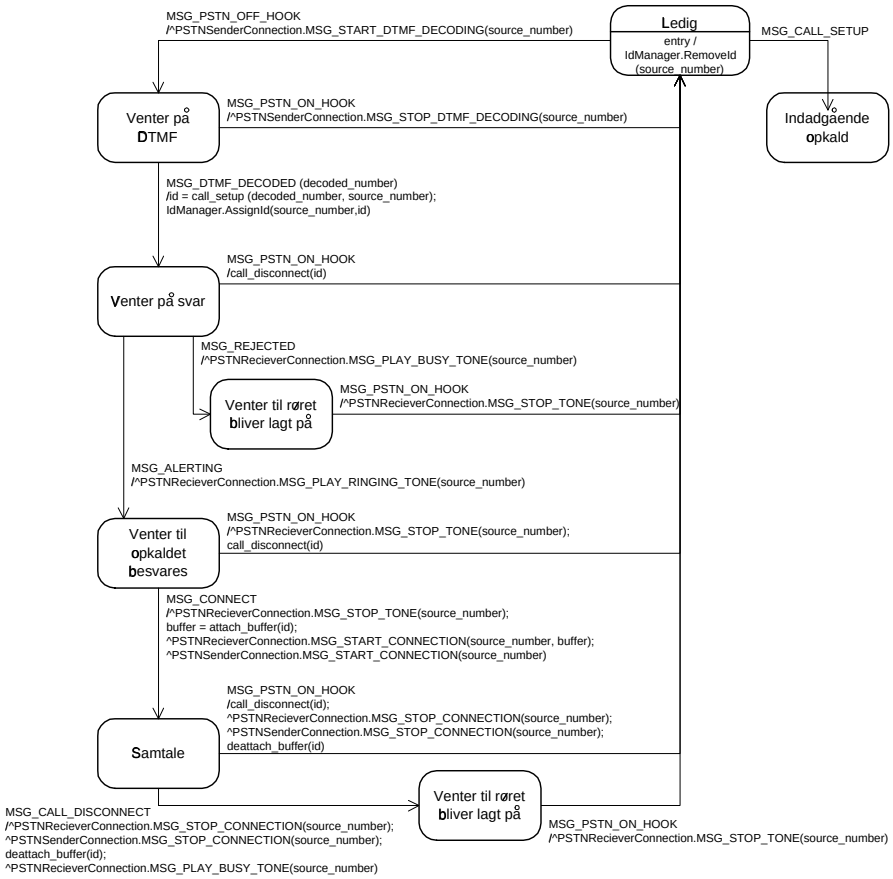
Figur 9.2: Statechart for indadgående opkald.

9.1.3 PSTNRecieverConnection-klassen

PSTNRecieverConnection-klassen arver fra EndpointRecieveConnection-klassen, og køres derfor i en tråd. Eet PSTNRecieverConnection-objekt skal håndtere datainput/lydoutput fra (op til) to samtaler. Dette skyldes anvendelsen af stereolydkort; begge kanaler i lydkortet leverer data på samme tid.
PSTNRecieverConnection-klassen kommunikerer vha. følgende beskeder:

- MSG_START_CONNECTION(source_number,buffer)
- MSG_STOP_CONNECTION(source_number)
- MSG_PLAY_DIAL_TONE(source_number)
- MSG_PLAY_RINGING_TONE(source_number)
- MSG_PLAY_BUSY_TONE(source_number)
- MSG_STOP_TONE(source_number)

PSTNRecieverConnection-klassens primære formål er at hente lyddata fra det modsatte endepunkts buffer, konvertere data via CodecManageren, og afspille dataene med lydkortet. Denne funktionalitet igangsættes af beskeden MSG-_START_CONNECTION(source_number,buffer) og stoppes af beskeden MSG-_STOP_CONNECTION(source_number). Parameteren source_number bruges til at bestemme hvilken kanal beskeden henvender sig til. Parameteren buffer er en reference til det bufferobjekt dataen skal hentes fra.
Klassen er også beregnet til afspilning af PSTN-toner. Tre forskellige toner bruges i systemet, og afspilningen af disse igangsættes af beskederne MSG-_PLAY_RINGING_TONE (source_number), MSG_PLAY_BUSY_TONE-(source_number og MSG_STOP_TONE(source_number). Afspilningen stoppes med beskeden MSG_STOP_TONE(source_number).



Figur 9.3: Statechart for udadgående opkald.

Klassen PSTNToneGenerator bruges til at generere disse toner. Afspilning gennem lydkortet sker ved hjælp af DirectSoundPlayback-klassen, som er beskrevet i afsnit E.

9.1.4 PSTNSenderConnection-klassen

PSTNSenderConnection-klassen køres ligesom PSTNRecieverConnection-klassen i en tråd, og skal håndtere lydinput/dataoutput for (op til) to samtaler. Det primære mål er at hente lyddata fra lydkortet og sætte det i to lydbuffere (som håndteres af klassen AudioBuffer), hvorfra det modsatte endepunkt kan hente dataen. Klassen kommunikerer vha. følgende beskeder:

- MSG_START_CONNECTION(source_number)
- MSG_STOP_CONNECTION(source_number)
- MSG_START_DTMF_DECODING(source_number)
- MSG_STOP_DTMF_DECODING(source_number)

- MSG_DTMF_DECODED(source_number,decoded_number) (udadgående besked)

MSG_START_CONNECTION(source_number) igangsætter en overførsel af lyddata fra lydkortet til den ene lydbuffer, og MSG_STOP_CONNECTION-(source_number) stopper en overførsel. Parameteren source_number bruges til at bestemme hvilken kanal beskeden henvender sig til.
PSTNSenderConnection-klassen skal også bruges til igangsættelse af dekodning af DTMF-toner. Dekodningen sker vha PSTNDTMFDecoder-klassen og igangsættes med MSG_START_DTMF_DECODING(source_number). Når dekodningen igangsættes, sættes PSTNRecieverConnection-klassen til at afspille "DIAL TONE", og når det første DTMF-tegn er dekodet, stoppes afspilningen af denne tone. DTMF-dekodningen stopper automatisk, når en gyldig telefonnummerstræng er dekodet, eller når brugeren trykker på tegnet #. En endt DTMF-dekodning sendes det dekodede nummer med beskeden MSG_DTMF-_DECODED(source_number,decoded_number) til PSTNController-objektet. Der mulighed for at afbryde en DTMF-dekodning ved at sende en MSG_STOP_-DTMF_DECODING (source_number) besked.
Algoritmen for DTMF-dekodning er beskrevet i afsnit C. Indspilning gennem lydkortet sker ved hjælp af DirectSoundRecord-klassen, som er beskrevet i afsnit E.

9.1.5 PSTNParallelPort-klassen

Formålet med denne klasse er at overvåge PSTN-enhedernes tilstand og at styre PSTN-enhedernes ringetone. Dette sker via et parallelportsinterface, som er specificeret i kravspecifikationen, afsnit 2.4.2.
Denne klasse køres i en tråd, som skiftevis tjekker sin beskedkø og parallelportens status. Kommunikationen med den klassen foregår via følgende beskeder:

- MSG_START_RING(source_number)
- MSG_STOP_RING(source_number)
- MSG_PSTN_ON_HOOK(source_number)
- MSG_PSTN_OFF_HOOK(source_number)

Parameteren source_number angiver den pågældende PSTN-enhed.

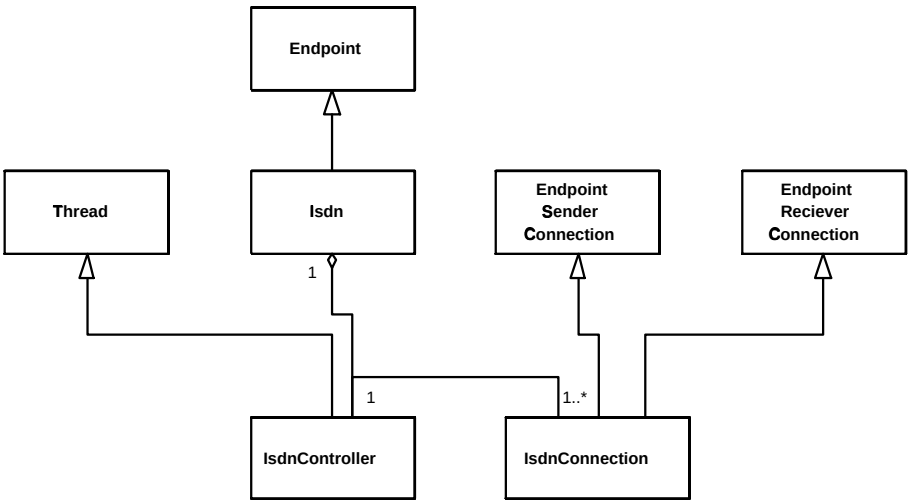
9.2 ISDN

ISDN klassen består ligesom Endpoint-klassen (se afsnit 8.2.3), overordnet af et kontrol-objekt (IsdnController), connection-objekter for hver ISDN kanal (IsdnConnection) og IdManageren, der nedarves fra Endpoint-klassen. Desuden indeholder IdManageren et IsdnStatus objekt for hver kanal.
Kontrol-objektet skal under eksekveringen af programmet sørge for den overordnede styring af samtalerne.

Ved opstart instantieres IsdnConnection-objekterne for hver ISDN kanal. Disse objekter's tråde bliver dog først aktiveret, når IsdnControlleren modtager en anmodning om en samtale. Efter at samtalen er ophørt, bliver IsdnConnection-objektet igen stoppet.
Metoderne i Endpoint-klassen nedarves direkte, pånær call_setup beskeden, idet metoden i Endpoint-klassen indsætter id'et i IdManageren ud fra destinationstelefonnummeret. I Isdn-klassen er destinationstelefonnummeret nummeret på den, der ønskes opkaldt, og kildetelefonnummeret det nummer der er tilknyttet ISDN-kanalen. Derfor skal id'et indsættes ud fra kildetelefonnummeret. Dette giver følgende pseudokode:

```
call_setup(source_number,destination_number)
{
  id = create_id(source_number,destination_number,current_time);
  if (IdManager.insert_id(id,source_number))
        controller.send_message(MSG_CALL_SETUP,id);
  else
        endpoint.call_reject(id);

  return id;
}
```

Den overordnede struktur i ISDN-klassen kan ses på figur 9.4.

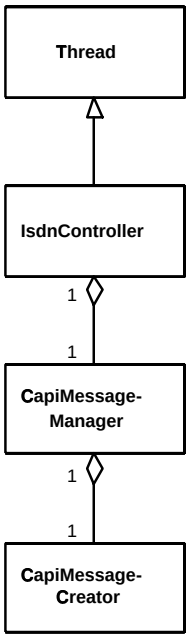


Figur 9.4: ISDN klassens overordnede struktur

9.2.1 Kontrolobjektet

Ligesom i Endpoint-klassen styrer IsdnControlleren formidling af kontrolbeskeder mellem CAPI, det modsatte endepunkt, og status-objekterne.

Til kommunikation med modsatte endepunkt i denne forbindelse anvendes de generelle kontrolbeskeder (Se kapitel 8.2.3). Disse placeres i det modsatte endepunkts kontrol-objekts beskedkø via metoderne beskrevet i afsnit 8.2.3. Ligeledes bliver kontrolbeskeder fra modsatte endepunkt indsat i kontrol-objektets beskedkø.
Kommunikationen på ISDN siden foregår gennem CAPI ved at bruge beskeder, som sendes til CAPI (Se appendiks B). CAPI-beskederne har variabel længde, hvilket gør, at man ikke kan lave statiske strukturer. Der er derfor designet to klasser, som IsdnController- og IsdnConnection-klasserne indeholder. Den ene er CapiMessageManager. Objektet af denne klasse sætter CAPI-beskederne sammen ved at kalde et andet objekt, som er et object af klassen CapiMessage-Creator.
På figur 9.5 er den overordnede struktur i kontrolobjektet afbilledet.



Figur 9.5: Den overordnede struktur i kontrolobjektet

9.2.2 IsdnStatus-objekter

Opkald til ISDN

Modtager IsdnStatus objektet en call_setup-besked fra modsatte endepunkt bliver der, hvis en samtale allerede er i gang, sendt en call_rejected besked retur. Ellers sendes en call_alert besked, som indikerer, at den ønskede forbindelse er godkendt og der skiftes tilstand. Det tilknyttede IsdnConnection-object, som håndterer dataoverførelsen mellem ISDN-objektet og det andet endepunkt, ak-

tiveres. Herefter påbegyndes opsætningen af forbindelsen på ISDN. Dette sker ved at udføre følgende:

- Lav connect_req ved at kalde CapiMessageManager
- Send connect_req til CAPI
- Modtag connect_conf fra CAPI
- Modtag connect_active_ind fra CAPI
- Lav connect_active_resp ved at kalde CapiMessageManager
- Send connect_active_resp CAPI
- Lav connect_b3_req ved at kalde CapiMessageManager
- Send connect_b3_req til CAPI
- Modtag connect_b3_conf fra CAPI
- Modtag connect_b3_active_ind fra CAPI
- Lav connect_b3_active_resp

Når ovenstående sekvens er udført, er der etableret forbindelse til ISDN, og der sendes en call_connect-besked til det andet endepunkt. Denne besked angiver til det andet endepunkt, at forbindelsen er blevet oprettet.

Opkald fra ISDN

Når IsdnController'en via. CAPI registrerer, at der er opkald fra ISDN (modtager connect_ind), sendes beskeden til det tilknyttede IsdnStatus-objekt, og følgende sekvens udføres:
Det modsatte endepunkt kaldes med beskeden call_setup. Herefter sendes beskeden alert_req vha. CAPI til den kaldende part. Dette gøres, da det ikke kan garanteres at modtageren på PSTN siden har løftet røret, inden der er gået fire sekunder. Herefter kan man enten modtage et call_reject eller et call_alert fra det andet endepunkt. Hvis man modtager et call_reject lukkes forbindelsen ned på ISDN siden ved at sende et connect_resp vha. CAPI, som afviser opkaldet. Modtages beskeden call_alert aktiveres IsdnConnection-objektet, som håndterer dataoverførelsen mellem ISDN objektet og det andet endepunkt.
Efter at forbindelsen til PSTN er blevet sat op, modtager IsdnControlleren call_connect fra det andet endepunkt. Herefter udføres følgende:

- Lav connect_resp ved at kalde CapiMessageManager
- Send connect_resp til CAPI
- Modtag connect_active_ind fra CAPI
- Lav connect_active_resp ved at kalde CapiMessageManager

- Send connect_active_resp til CAPI
- Modtag connect_b3_ind fra CAPI
- Lav connect_b3_resp ved at kalde CapiMessageManager
- Send connect_b3_resp til CAPI
- Modtag connect_b3_active_ind fra CAPI
- Lav connect_b3_active_resp

Der er nu oprettet forbindelse mellem ISDN og det andet endepunkt.

Nedlukning af forbindelser

Efter endt samtale skal forbindelsen lukkes.
Hvis forbindelsen afbrydes fra LAN modtager IsdnControlleren signalet call_disconnect. Dette sendes til IsdnStatus-objektet for samtalen. hvorefter status-objektet signalerer til Connection-objektet, at det skal stoppe overførsel af data. Herefter lukkes ISDN forbindelsen ned med følgende sekvens:

- Lav disconnect_b3_req ved at kalde CapiMessageManager
- Send disconnect_b3_req til CAPI
- Modtag disconnect_b3_conf fra CAPI
- Modtag disconnect_b3_ind fra CAPI
- Lav disconnect_b3_resp ved at kalde CapiMessageManager
- Send disconnect_b3_resp til CAPI
- Lav disconnect_req ved at kalde CapiMessageManager
- Send disconnect_req til CAPI
- Modtag disconnect_req fra CAPI
- Modtag disconnect_ind fra CAPI
- Lav disconnect_resp ved at kalde CapiMessageManager
- Send disconnect_resp til CAPI

Efter endt terminering af forbindelsen opdateres status-objektes tilstand.
Hvis forbindelsen afbrydes fra ISDN, modtages signalet disconnect_b3_ind fra CAPI. Status-objektet sender derefter signalet call_disconnect til LAN og IsdnConnection-objektet's overførsel af data stoppes. Herefter lukkes ISDN-forbindelsen ned:

- Lav disconnect_b3_resp ved at kalde CapiMessageManager

- Send disconnect_b3_resp til CAPI
- Modtag disconnect_ind fra CAPI
- Lav disconnect_resp ved at kalde CapiMessageManager
- Send disconnect_resp til CAPI

Efter endt terminering af forbindelsen opdateres tilstanden.

CapiMessageManager

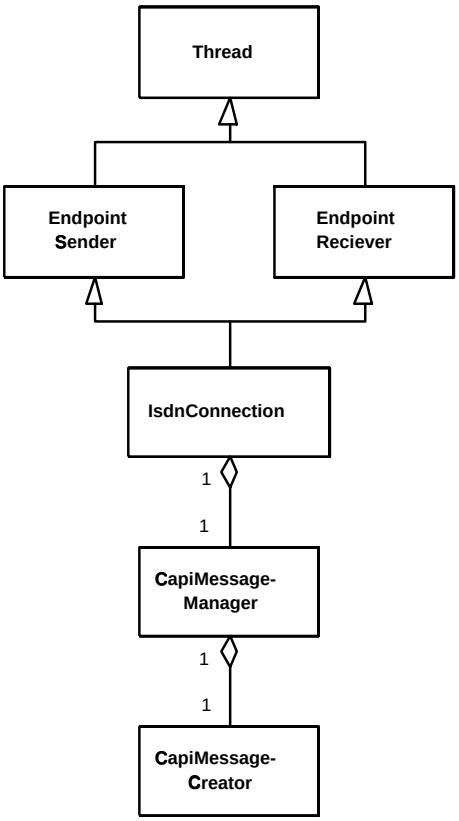
Da CAPI-beskederne har variabel længde, har vi valgt at generere beskederne vha. klasserne CapiMessageManager og CapiMessageCreator.
Herved undgåes at skulle implementere beskedernes parametere flere gange og risikoen for fejl i programmet mindskes.
CapiMessageManagerens metoder bestemmer ud fra parameterne i sit kald, headerens format og hvilke parametere, som skal være med i beskeden. Den kalder CapiMessageCreators funktioner, hvorefter CapiMessageCreator genererer en besked, som kan sendes til CAPI. Denne hentes efter at CapiMessageManageren har tilføjet headeren og alle parameterne til beskeden.

CapiMessageCreator

CapiMessageCreator sætter en besked sammen ud fra nogle kald til klassen.
Beskeden laves i en buffer med længden 1024 bytes. Denne længde er større end længden af den største mulige CAPI-besked, og der vil derfor ikke være risiko for overflow i bufferen.
CapiMessageCreator har en metode kaldet reset, som resetter bufferen (sætter længden lig nul, og sætter pointeren i begyndelsen af bufferen). Denne metode er altid den første metode der kaldes i forbindelse med generering af en CAPI-besked.
Den første metode, som skal kaldes efter, at bufferen er blevet resat, er make_message_header. Denne metode genererer en header ud fra parameterne, som den bliver kaldt med, og tilføjer den først i bufferen.
Efterfølgende tilføjes de nødvendige parametere vha. kald til CapiMessageCreator fra CapiMessageManageren.

9.2.3 Connection-objekter

IsdnConnection-klassen nedarver af både EndpointSenderConnection- og EndpointRecieverConnection-klassen, og indeholder derfor både en Audiobuffer og en CodecManager. Dette er gjort fordi CAPI-forbindelsen er bidirektionel. Når En forbindelse er oprettet varertager et IsdnConnection-objekt overførslen af data mellem CAPI og modsatte endepunkt, og den modsatte vej efter tur. Da IsdnConnection-objekterne også skal snakke med CAPI, indeholder de også et CapiMessageManager- og et CapiMessageCreator-objekt, ligesom IsdnController-objektet.
På figur 9.6 ses den overordnede struktur i IsdnConnection-objektet.



Figur 9.6: Den overordnede struktur i kontrolobjektet

Overførsel af data

Overførsel af data til og fra CAPI sker henholdsvis med beskederne data_b3_req og data_b3_ind. Disse beskeder laves vha. CapiMessageManager efter princippet beskrevet i afsnit 9.2.2 og 9.2.2. For hver besked gives besked tilbage af modtageren om, at pakken er modtaget (handshake). data_b3 beskederne indeholder længden af datapakkerne og en pointer til dem.
Overførelsen af data mellem ISDN klassen og modsatte endepunkt foregår som beskrevet i afsnit 8.2.3 om overførelse af data fra endpoint til endpoint.
Overførslen af data stoppes, når det tilknyttede IsdnStatus-objekt anmoder om dette.

Kapitel 10

Accepttest

10.1 Accepttestspecifikation

10.1.1 Formål

Formålet med accepttesten er, at teste det samlede system i form af en black-boxtest. Testen opfatter systemet på samme måde som brugeren af systemet. Accepttesten deles op i nogle mindre test, de er som følgende :

- Opstart af systemet (afsnit 10.1.3).
- Modtage opkald fra ISDN (afsnit 10.1.3).
- Foretage et opkald på PSTN (afsnit 10.1.3).
- Latensen i systemet (afsnit 10.1.3).

Inden accepttesten foretages en række testfaser, som er beskrevet i afsnit 10.1.2.

10.1.2 Indledende test

Accepttesten er en test, der skal godkende systemet i forhold til kravene i kravspecifikationen. Inden test udføres er der gennemføres en række forskellige testfaser, der bevirker at acceptesten ikke bliver en "Big Bang" test.
Følgende testfaser gennemføres:

- Modultest.
- Procestest.

Modultest

Modultesten er en del af integrationstest. Modultesten har til formål, at teste de enkelte moduler efter whitebox test princippet.
En whitebox test gennemløber alle uafhængige stier gennem et modul, f.eks. en metode i en klasse. Stierne checkes op mod det specificerede design af modulet.

Overholder alle stierne specifikationerne, godkendes modulet til videre integration. Denne test foretages på alle modulerne i systemet.

Procestest

Procestesten er en del af integrationstest på samme måde som modultesten. Procestesten tester interproceskommunikationen mellem trådene i systemet.
Til analyse af rækkefølgen trådene spawnes, logges tiden som det første i tråden. Det giver en indikation på antallet af tråde, samt deres indbyrdes spawntider. Interproceskommunikationen mellem trådene testes ved at indsætte nogle teststubbe, som dels sender og modtager nogle specifikke testbeskeder. Systemet sættes til, at sende en beskred gennem alle trådene. Samt en logning af tiden og tråden, der behandler beskeden. Rækkefølgen og hvilke tråde beskeden sendes igennem kortlægges på denne måde. Er observationerne i overensstemmelse med designet godkendes procestesten.

10.1.3 Specifikation af accepttesten

Acceptesten er delt op i fire dele, de er som følgende :

Opstart af systemet : Beskriver accepttesten af systemopstarten. Den tidslige definition af systemopstart, er fra start af program, til systemet er i idle og klar til, at tage imod opkald.

Modtage opkald fra ISDN : Beskriver accepttesten af et opkald modtaget fra ISDN-nettet til PSTN. Testene udføres først for en forbindelse. Testene gentages med en anden aktiv samtale kørende.

Foretage opkald på PSTN : Beskriver accepttestene af et opkald foretaget på PSTN til ISDN-nettet. Testene udføres først for en forbindelse. Testene gentages med en anden aktiv samtale kørende.

Latens i systemet : Beskriver acceptesten af systemets lydforsinkelse. Testen udføres uden andre aktive samtaler, samt gentages med en aktiv samtale.

Opstart af systemet

ISDN :

Input : Ingen.

Gyldigt resultat : Ingen fejlbeskeder.

Ugyldigt resultat : Fejlbeskederne : Ingen ISDN-kort og ingen CAPI driver installeret.

LAN :

Input : Ingen.

Gyldigt resultat : Ingen fejlbeskeder.

UGyldigt resultat : Fejlbeskeden : Netværksfejl.

DirectSound :

Input : Ingen.

Gyldigt resultat : Ingen fejlbeskeder.

Ugyldigt resultat : Fejlbeskeden : DirectSound ikke installeret, bruges af en anden applikation, eller lydkortet opfylder ikke kravet: 2 kanaler og fuld duplex.

Resourcer :

Input : Ingen.

Gyldigt resultat : Ingen fejlbeskeder.

Ugyldigt resultat : Fejlbeskeden : Ikke nok resourcer.

Parallel port :

Input : Ingen.

Gyldigt resultat : Ingen fejlbeskeder.

Ugyldigt resultat : Fejlbeskeden : Driver ikke installeret.

Modtage opkald fra ISDN

Check for ringeindikation :

Input : Opkald til PSTN-telefon fra ISDN.

Gyldigt resultat : Telefonen ringer.

Ugyldigt resultat : Telefonen ringer ikke.

Opkalder afslutter, inden telefonen er besvaret Input : Opkald til PSTN-telefon. Efter ringeindikation på PSTN-telefonen afsluttes opkaldet.

Gyldigt resultat : Telefonen stopper med at ringe.

Ugyldigt resultat : Telefonen ringer videre.

Samtale :

Input : Opkald til PSTN-telefon. En samtale påbegyndes ved, at røret på PSTN-telefonen løftes.

Gyldigt resultat : Lyden fra ISDN-nettet afspilles i PSTN-telefonen, og lyden fra PSTN-telefonen afspilles på ISDN-nettet.

Ugyldigt resultat : Der afspilles ingen lyd på ISDN-nettet og/eller PSTN-telefonen.

Opkalder afslutter samtale :

Input : Efter påbegyndt samtale, afsluttes den fra ISDN-nettet.

Gyldigt resultat : Lyden afbrydes, der bliver stille i PSTN-telefonrøret.

Ugyldigt resultat : Lyden afbrydes ikke, der afspilles lyd i PSTN-røret.

PSTN-telefonen afslutter samtalen :

Input : Efter påbegyndt samtale, afsluttes den af PSTN-telefonen.

Gyldigt resultat : Lyden afbrydes, der bliver stille på ISDN-nettet.

Ugyldigt resultat : Lyden afbrydes ikke, der afspilles lyd på ISDN-nettet.

Foretage opkald på PSTN

Check af klartone :

Input : Røret på PSTN-telefonen løftes.

Gyldigt resultat : Der afspilles en klartone i PSTN-telefonrøret. Klartonen er specificeret i afsnit D.

Ugyldigt resultat : Der afspilles ingen klartone.

Check af ringingtone :

Input : Røret på PSTN-telefonen løftes og et gyldigt telefonnummer ringes.

Gyldigt resultat : Der afspilles en ringingtone. Ringingtonen er specificeret i afsnit D.

Ugyldigt resultat : Der afspilles ingen klartone.

Check af busytone :

Input : Røret på PSTN-telefonen løftes. Et gyldigt og optaget telefonnummer ringes.

Gyldigt resultat : Der afspilles en busytone. Busytonen er specificeret i afsnit D.

Ugyldigt resultat : Der afspilles ingen busytone.

Samtale :

Input : Røret på PSTN-telefonen løftes. Et gyldigt telefonnummer ringes. En samtale påbegyndes ved, at ISDN-nettet svarere på opkaldet.

Gyldigt resultat : Lyden fra ISDN-nettet afspilles i PSTN-telefonen, og lyden fra PSTN-telefonen afspilles på ISDN-nettet.

Ugyldigt resultat : Der afspilles ingen lyd på ISDN-nettet og/eller PSTN-telefonen.

Opkalder afslutter samtale :

Input : Efter påbegyndt samtale, afsluttes den fra ISDN-nettet.

Gyldigt resultat : Lyden afbrydes, der bliver stille i PSTN-telefonrøret.

Ugyldigt resultat : Lyden afbrydes ikke, der afspilles lyd i PSTN-røret.

PSTN-telefonen afslutter samtalen :

Input : Efter påbegyndt samtale, afsluttes den af PSTN-telefonen.

Gyldigt resultat : Lyden afbrydes, der bliver stille på ISDN-nettet.

Ugyldigt resultat : Lyden afbrydes ikke, der afspilles lyd på ISDN-nettet.

Latens i systemet

Latensen i systemet :

Input : Forudsætningen er, at der ikke foregår samtaler på systemet. Testen simulerer et opkald fra ISDN-nettet. Lyden i PSTN-telefonen er sløjfet, så lyden fra line-out sendes direkte ind i line-in. Tiden logges inden in-bufferen på ISDN-siden fyldes med en kort tone. Tonen sendes gennem system og tilbage til out-bufferen på ISDN-siden. Når out-bufferen modtager data loges tiden igen. Differencen mellem de to tidslogninger giver den dobbelte latens i systemet.

Gyldigt resultat : En latens < 250 ms.

Ugyldigt resultat : En latens > 250 ms vil give afbrydelser i lyden [Eric Larson, 1999].

Latens i systemet med en aktiv samtale :

Input : Forudsætningen er, at der foregår en aktiv samtale på systemet. Testen simulerer et opkald fra ISDN-nettet. Lyden i PSTN-telefonen er sløjfet, så lyden fra line-out sendes direkte ind i line-in. Tiden logges inden in-bufferen på ISDN-siden fyldes med en kort tone. Tonen sendes gennem system og tilbage til out-bufferen på ISDN-siden. Når out-bufferen modtager data loges tiden igen. Differencen mellem de to tidslogningerne giver den dobbelte latens i systemet.

Gyldigt resultat : En latens < 250 ms.

Ugyldigt resultat : En latens > 250 ms vil give afbrydelser i lyden [Eric Larson, 1999].

Kapitel 11

Studierapport

11.1 Indledning

Følgende kapitel indeholder en studierapport, der har til formål at dokumentere projektrelaterede emner som overvejelser vedrørende projektvalg, opgavefordeling i gruppen, planlægning af projektforløbet, erfaringer med gruppearbejde og samarbejde med vejleder, samt udbytte af PE-kurser. Da der ikke foreligger overordnede krav til at studierapporten skal indgå i den samlede rapport, laves den udelukkende på projektgruppens eget initiativ, idet den giver mulighed for at reflektere over projektforløbet og samle erfaringer.

11.2 Projektvalg

Da alle medlemmer i projektgruppen har arbejdet sammen i tidligere projekter, har vi opnået et vist kendskab til hinandens arbejdsmetoder, samt erfaring i hvorledes projektarbejdet bør struktureres, for at udnytte gruppens ressourcer. Projektgruppen har i tidligere projekter haft en svaghed, hvad angår afgrænsning af projektets omfang, hvorfor der var stor enighed om, at dette projekt fra starten skulle være så afgrænset og veldefineret som muligt.

11.2.1 Projektforeslag

Temaet for dette 5.semesters projektet "Sandtids-kommunikations-systemer"har affødt 8 projektforeslag.

- WEB-baseret fjernundervisningssystem.
- WWW-index
- Modulært proceskontrolsystem.
- Real-time scheduling
- WEB-cam.

- Video on demand
- VoIP på ADSL og CATV
- Styring af elevator.

Efter gennemlæsning af forslagene og mundtlig oplæg af forslagsstillerne, var der spredte meninger i gruppen om hvilket projekt vi ønskede at arbejde med. Desværre var den mundtlige fremlæggelse af projektforslagene lidt kaotisk og flere projektforslag var ikke blevet fremlagt mundtligt.
Gruppen var enige om at vælge et projekt, der indebar arbejde med netværk. Det resulterede i en prioriteret liste med WEB-cam som første prioritet og VoIP på ADSL og CATV som anden prioritet. Grunden til at nærværende projekt var et anden prioritets ønske, var manglende præsentation af emnet, og hvad det ville indebære.
På grund af stor interesse for WEB-cam projektet blev der foretaget lodtrækning blandt grupperne, hvorfor gruppen fik tildelt projekt svarende til anden prioritet.

11.3 Opgavefordeling

For at gøre projektarbejdet mere effektivt, besluttede vi ret tidligt i projektforløbet at dele gruppearbejdet op i undergrupper. Meningen med denne opdeling var i starten at skaffe bred baggrundsviden til hele gruppen. Senere blev disse grupper/enkeltpersoner ansvarlige for enkelte områder af projektet.
På grund af denne opdeling bestemte vi følgende: Vi skulle have koordineringsmøder en gang om ugen, så alle gruppemedlemmerne kunne følge med i, hvor langt vi var nået med de forskellige dele. Der skulle udfærdiges arbejdsblade om de forskellige emner, der blev arbejdet med til information for andre.
En opdeling af gruppen kan have følgende konsekvenser:

- Fordel: Projektet kan blive dybt og samtidig bredt
- Ulempe: Ikke alle i gruppen kan få samme kendskab til alle dele af projektet.

Vi føler i gruppen, at vi i dette projekt alle har haft meget at lave. Tiden er ikke blevet spildt med store diskussioner mellem alle gruppemedlemmerne, som i nogle tidligere projekter. Undergrupperne har selv foretaget de fleste beslutninger. Kun vigtige beslutninger med konsekvenser for resten af projektet er blevet taget op på fælles møder. Alt dette har resulteret i en langt mere effektiv arbejdsproces, som har gjort projektet både bredt og dybt.
Dette kan dog resultere i, at det enkelte gruppemedlem ikke kender alle dele af projektet fuldt ud. Dog har koordination og information været med til at udjævne disse kløfter, der er uundgåelige i en projektgruppe med 7 medlemmer.
Hvad kunne gøres bedre:

- Arbejdsblade skulle have blevet udarbejdet tidligere.

- Arbejdsblade kunne have været fremlagt tidligere.

11.4 Planlægning

I forbindelse med opdelingen af gruppen i undergrupper, blev der lavet en overordnet tidsplan.
Formålet med den overordnede tidsplan var dels at have et redskab til at planlægge en række milestones, samt at koordinere indsatsen i de forskellige undergrupper, herunder fordeling af ressourcer. Detaljeringen af den overordnede tidsplan blev overladt til undergrupperne.
Fællesmøder blev holdt med jævne mellemrum, både med og uden vejleder, hvor status i undergrupper blev rapporteret til resten af gruppen.
Milestones blev brugt til at overveje status i forhold til tidsplanen, og afvigelser blev konstateret. På et tidspunkt blev afvigelserne på visse områder så store at en korrektion var nødvendig. En sådan korrektion var mulig, fordi den oprindelige tidsplan havde indbygget elasticitet, med henblik på at tage højde for uforudsete forsinkelser.

11.5 Gruppearbejde

Internt i gruppen har vi benyttet følgende til at koordinere og informere hinanden om arbejdet i gruppen:

- Pligtfordeling
- Mødeteknik
- Arbejdsblade

I det følgende er beskrevet, hvorledes vi har effektueret disse punkter.

11.5.1 Pligtfordeling

I gruppen uddelte vi fra starten de organisatoriske pligter til de enkelte gruppemedlemmer.
Mødekoordinatoren har haft til opgave at styre både interne møder i gruppen og eksterne møder med vejleder. Dette blev gjort ud fra en dagsorden, som er blevet mailet til de implicerede senest dagen før. Al kommunikation med vejleder er gået igennem mødekoordinatoren for at undgå multiple modstridende aftaler.
Referanten har skullet tage referat af møderne i en detaljegrad, så alle vigtige oplysninger, valg og aftaler kan gendannes. Disse referater er blevet indskrevet af referanten og givet til webmasteren.
Webmasteren har løbende haft til opgave at opdatere gruppens hjemmeside med de sidste nye informationer, så de andre i gruppen løbende har kunnet holde sig orienteret om de forskellige dele af projektet på internettet.

Papirkoordinatorens opgave har været at sørge for at al udefrakommende post er blevet kopieret og fordelt til alle gruppens medlemmer. Endvidere har han haft til opgave at informere de andre i gruppen om disse.
Den hardwareansvarlige har haft til opgave at finde ud af, hvilke komponenter som var til rådighed. Desuden har han indsamlet information om de anvendte komponenter.
Den CVS-repository-ansvarlige har haft til opgave at strukturere og opretholde orden i CVS-repository, samt at stille softwarehjælpemidler, såsom Makefiles, til rådighed.
Den rapport-ansvarliges opgave har bestået i at varetage en central styring af hovedstrukturen i rapporten, det vil f.eks. sige udarbejde en disposition, bibliography, synopsis/abstract og forord.
De organisatoriske pligter er af alle gruppens medlemmer blevet udført ansvarligt. I enkelte tilfælde har der dog været små forsinkelser i udførelsen af pligterne. De enkelte medlemmer har taget mere ansvar end tidligere.

11.5.2 Mødeteknik

I forbindelse med møderne har der været en bestemt arbejdsgang.
Møderne er blevet ledet "løst" af mødekoordinatoren. Senest dagen før mødet har alle gruppens medlemmer modtaget en mail fra mødekoordinatoren med dagsordenen for mødet. Herefter var det op til de enkelte medlemmer at læse den relevante information på gruppens hjemmeside (Referat fra tidligere møde, ny dokumentation osv.) eller de udfærdigede arbejdsblade.

11.5.3 Arbejdsblade

Under udviklingen af de forskellige dele af projektet har de implicerede lavet arbejdsblade om emnerne. Disse har været været til rådighed til læsning på gruppens hjemmeside. I enkelte tilfælde er der foretaget præsentation af relevante dele af projectet som supplement til arbejdsbladene.

11.6 Samarbejde med vejleder

Samarbejdet med vores to vejledere har været tilfredsstillende. Vi har arbejdet selvstændigt, hvilket er blevet respekteret. Samtidig har vores to vejledere med deres forskellige baggrund suppleret hinanden godt.

11.7 Kurser

I projektforløbet har vi deltaget i en række kurser med det formål at støtte os i projektarbejdet i form af projektenhedskurser, der udgør projekt-pensum og evalueres igennem projektevalueringen. SE-kurserne har denne gang ikke været relevante i forhold til projektet.

Desværre er en del af PE-kurserne været afholdt for sent i projektforløbet til at projektgruppen har kunne opnå fuldt udbytte heraf, f.eks. afholdes de sidste minimoduler af Netværk og datakommunikation ugen efter afleveringsdatoen.
Indholdsmæssigt har PE-kurserne være relevante, dog har kurset i Algoritmer og datastrukturer ikke været direkte anvendeligt i projektet men både interessant og relevant uddannelsesmæssigt.
I kurset i Sandtidssystemer var en stor stofmængde presset net på et enkelt modul, og detaljeringsgraden må siges klart at overstige det i projektet mulige.

11.8 Konklusion

Når vi skal vurdere vores erfaringer med dette projekt, er vi nødt til at se tilbage på vores tidligere erfaringer. For flertallet af gruppemedlemmernes vedkommende er det vort 5. semester sammen. Vi har derfor mulighed for at vurdere vores fremskridt på baggrund af et længere udviklingsforløb.
Vort 4. semester var i mange henseender tilfredsstillende. Projektet var udfordrende og vi fik løst nogle problemer vedrørende fordelingen af arbejdsbyrde, som vi havde med os fra tidligere projekter. På et afgørende punkt var dette projekt dog kilde til stor frustration. Vi formåede ikke at afgrænse stofmængden, med det resultat at arbejdsbyrden voksede over hovedet på os og vi mistede kontrollen med projektforløbet. Dettte projekt startede derfor med den udtalte intention, at vi denne gang ikke skulle "slå større brød op end vi kunne bage". Denne intention gik dog hånd i hånd med intentionen om et projektforløb, der var fagligt udfordrende, og denne intention var baggrunden for, at vi valgte et projekt, hvor vi ikke fra start kunne overskue, hvor stor en arbejdsbyrde projektet indebar.
Resultatet er da også blevet at vi i projektforløbet er løbet ind i problemer med en stofmængde, der blev vildtvoksende og uhåndterlig. Det gælder først og fremmest arbejdet med at overskue de standarder som anvendes i et system som vores. Vi har imidlertid formået at tage nogle beslutninger vedrørende afgrænsninger på forskellige kritiske tidspunkter i projektforløbet. Konsekvensen har været, at vi frem til skrivende stund har fastholdt kontrollen med projektet, i modsætning til tidligere projekter, der som regel har haft en kaotisk afslutning.
Projektet har på en række områder budt på nogle spændende udfordringer. I software-analyse og -design har vi forsøgt at forene inflydelserne fra SPU, OOAD, COBRA og CODARTS. Dette har givet anledning til mange frugtbare overvejelser, der har medført en klarere og dybere opfattelse af betydningen analyse- og designfasen, herunder også den anvendte UML-notation.
Et andet punkt, der bør nævnes er, at vi i dette projekt, i modsætning til tidligere, har fremstillet prototyper på forskellige områder, hvilket har vist sig at være en klar fordel.
Generelt kan vi konkludere, at det har været et tilfredsstillende projektforløb, hvor det eneste større minus er, at vi i skrivende stund endnu ikke har et system oppe at køre.

Kapitel 12

Konklusion

12.1 Indledning

Målet for dette projekt har været at analysere, designe, implementere og teste et VoIP-system bestående af en ISDN/IP-gateway og en IP/PSTN-gateway, der kan kommunikere med hinanden via et IP-baseret netværk i henhold til Anbefaling H.323. Der er i kravspecifikationen foretaget en afgrænsning i forhold til Anbefaling H.323, idet multipoint-konferencer og videotelefoni ikke understøttes.

12.2 Analyse

Analysen har omfattet en række punkter:

- En undersøgelse af hvilke krav der er gældende for et system, der skal følge Anbefaling H.323.
- En undersøgelse af et TCP/IP-baseret netværks egnethed til overførsel af realtidsdata.
- En tidsanalyse hvor realtidsproblematikker er analyseret på et generelt niveau.
- En undersøgelse af forskellige operativsystemers egnethed til realtids-systemer.
- En software-analyse

12.2.1 Undersøgelse af krav til et H.323-system

En generel beskrivelse af opbygningen af et H.323-system er lavet. Denne beskrivelse omfatter terminaler, gatekeeper og gateways samt de protokoller, der definerer beskederne, som udveksles mellem disse. Desuden er procedurerne for kaldssignalering beskrevet.

12.2.2 Netværksanalyse

Gennerelle netværksprocedurer for overførsel af data på et pakkebaseret netværk er beskrevet. I denne beskrivelse er der lagt vægt på de problemer, der er forbundet med overførsel af realtidsdata. Netværks- og transportprotokoller, herunder RTP og RTCP er kort beskrevet. Derudover er kravene til båndbredde analyseret. På grundlag af en undersøgelse, beskrevet i [Black, 1999] konkluderes det, at den optimale datagram-størrelse er 64 byte. Dette medfører en forsinkelse på 8 ms. Da summen af konstante forsinkelser i et VoIP-system ifølge [Eric Larson, 1999] er 71-129 ms, hvilket medfører, at der kan tolereres et variabelt delay på 120 ms, uden at tollerancen for forsinkelse på 250 ms overskrides, vurderes forsinkelsen, for den optimale datagramstørrelse, til at være acceptabelt.

12.2.3 Tidsanalyse

På baggrund af den generelle analyse af kravene til et realtidssystem konkluderer vi, at det er mest hensigtsmæssigt at betragte systemet som et blødt realtidssystem. Endvidere er principperne for test af systemets schedulerbarhed beskrevet.

12.2.4 OS-analyse

En analyse af Windows NT, RTLinux og RTMACH er udført med henblik på at vælge et egnet operativsystem. Windows NT 4.0 understøtter ikke DirectSound, som anvendes til optagelse og afspilning af lyd. Til gengæld understøtter det, som det eneste af de tre analyserede operativsystemer, ISDN-2 kortet, der skal anvendes i henhold til kravspecifikationen.
Da Windows 2000 og Windows 98 understøtter DirectSound 7.0, som anvendes til optagelse og afspilning af lyd, kan disse operativsystemer anvendes som platform for terminalen. På denne baggrund er der foretaget en sammenlignende analyse af Windows NT 4.0, Windows 2000 og Windows 98. Denne analyse omfatter undersøgelse af henholdsvis trådskifte, processkifte og blokerende kald, samt af scheduleringsalgoritme.
På baggrund af analysen vælges Windows NT 4.0 som operativsystem for gateway'en, da det er det af operativsystemerne, som har den mest homogene styring af trådskifte. Som operativsystem til terminalen vælges Windows 2000, da den understøtter DirectSound og da den har en bedre styring af trådskifte end Windows 98.

12.2.5 Software-analyse

En software-analyse af systemet er udført. Forud for software-analysen er der foretaget en yderligere afgrænsning af systemet, således at software-analysen ikke omfatter, den i Anbefaling H.245 specificerede, kontrolkanal.

12.3 Softwaredesign

Et software-design af systemet er gennemført. Forud for design er der foretaget en yderligere afgrænsning af systemet, således at design ikke omfatter udvekslingen af de i Anbefaling H.225 definerede beskeder til kaldsopsætning, registrering, adgangskontrol og angivelse af status.
Designet indbefatter design af subsystemer til udveksling af beskeder og data med henholdsvis en ISDN- og en PSTN-linje, samt over et LAN. Disse subsystemer er designet med relativ lav kobling, således at en applikation baseret på en H.323-protokolstak senere vil kunne designes og implementeres på LAN-delen af systemet.

12.4 Implementation

Forud for implementationen er der foretaget en yderligere afgrænsning af systemet, således, at den i Anbefaling H.323 definerede gatekeeper ikke vil blive implementeret. I skrivende stund er status for implementation som følger:

- Generel del: Thread-klassen og bufferhierarkiet er implementeret. Endpoint-klassen er delvist implementeret. EndpointSenderConnection, EndpointReceiverConnection, EndpointStatus, CodecManager og IdManager er ikke implementeret.
- LAN: LAN-delen er implementeret bortset fra koblingen til ISDN- og PSTN-klasserne.
- ISDN: CapiMessageCreator er delvist implementeret. CapiMessageManager, IsdnController, IsdnConnection og IsdnStatus er ikke implementeret.
- PSTN: PstnToneGenerator, DirectSoundRecord og DirectSoundPlayback er implementeret. PstnDtmfGenerator og PstnParallelport er delvist implementeret.

12.5 Test

Der er lavet modultest på de implementerede dele. Der er foretaget integrationstest af en LAN-til-LAN-forbindelse. Det har endnu ikke været muligt at foretage integtationstest for andre dele af systemet eller accepttest for det samlede system.

12.6 Obligatoriske krav

Ifølge kravspecifikationen skal systemet være transparent for brugeren. Dette vil ifølge designet blive opfyldt i den endelige version af programmet. Endvidere er det i kravspecifikationen specificeret, at antallet af forbindelser skal være skalerbart. Dette vil være opfyldt i programmet, grundet den dynamiske opbygning.

På afleveringstidspunktet for rapporten er systemet ikke fuldt ud implementeret, hvorved det ikke kan konstanteres, om systemet er i stand til at leve op til kravet om realtidsafvikling.
Overholdelsen af de angivne standarder er ikke realiseret, da det er prioriteret højere at få et minimumsystem op at køre.
Kravet til frekvensområde ($300Hz$ - $3,1kHz$) er overholdt.

Litteratur

et al Apostolopoulos. *RFC-2676, QoS Routing Mechanisms and OSPF Extensions.* 1999. URL `http://sunsite.auc.dk/RFC/rfc/rfc2676`.

Uyless D. Black. *Voice Over IP.* Prentice Hall, 1999. ISBN 0-13-022463-4.

Bruxelles. *Real Time Magazine Publication.* Bruxellse 5, 1997.

capi.org. *CAPI 2.0 Documentation.* Telekom ROLAND, 1999. URL `http://www.capi.org`.

Douglas E. Comer. *Computer Networks and Internets.* Prentice Hall, 2nd edition, 1999. ISBN 0-13-084222-2.

et al Eric Larson. *Voice Technilogies for IP and Frame Relay Networks.* Motorola, 1999. URL `http://www.mot.com/MIMS/ISG/mnd/papers`.

H.225.0. *Call signalling protocols and media stream packetization for packet-based multimedia communication systems.* ITU-T, 1998. 02/98.

H.246. *Interworking of H-Series multimedia terminals with H-Series multimedia terminals and voice/voiceband terminals on GSTN and ISDN.* ITU-T, 1998. 02/98.

H.323. *Packet-based multimedia communications systems.* ITU-T, 1998. 02/98.

H.450.1. *Generic functional protocol for the support of supplementary services in H.323.* ITU-T, 1998. 02/98.

at al Hideyuki Tokuda. Real-time mach: Towards a predictable real-time system. 1995.

Cliffor W. Mercer. An introduction to real-time operating systems: Scheduling theory. 1992.

Q.931. *ISDN user-network interface layer 4 specifikation for basic call control.* ITU-T, 1998. 05/98.

Q.932. *Digital subscriber signalling system no. 1 (DSS 1) - generic procedures for the control of ISDN supplementary services.* ITU-T, 1993. 03/93.

et al Schulzrinne. *RFC-1889, A Transport Protocol for Real-Time Applications.* 1996. URL `http://sunsite.auc.dk/RFC/rfc/rfc1889`.

Alok K. Sinha. *Network Programming in Windows NT.* Addison-Wesley Publishing Company, 1st edition, 1996.

William Stalling. *Operating Systems: Internals and Design Principles.* Prentice Hall, 3rd edition, 1992.

John A. Stankovic. Real-time computing. 1992.

W. Richard Stevens. *UNIX Network Programming Volumne 1 - Networking APIs: Sockets and XTI.* Prentice Hall PTR, 2nd edition, 1998. ISBN 0-13-490012-X.

Viktor Yodaiken. The rt-linux approach to hard real-time. 1998.

Akronymer

ACF AdmissionConfirm

ARJ AdmissionReject

ARQ AdmissionRequest

EDF Earliest Deadline First

IP Internet Protocol

IPC Inter Proces Communiation

ISDN Integrated Services Digital Network

ISR Interrupt Service Routine

LAN Local Area Network

MCU Multi Control Unit

PSTN Public Switched Telephone Network

QoS Quality of Service

RAS Registration, Admission and Status

RFC Requets For Comment

RMA Rate Monotonic

RTCP Real-Time Transport Control Protocol

RTP Real-Time Transport Protocol

SCN Switched Circiut Network

SPP Standard Parallel Port

TCP Transport Control Protocol

ToS Type of Service

UDP User Datagram Protocol

Appendiks A

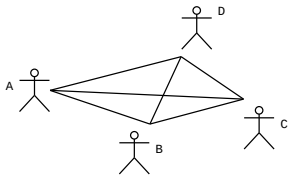
RTP

Dette afsnit omhandler de tekniske specifikationer vedrørende RTP. Afsnittet bygger hovedsaglig på IETF's RFC 1889 [Schulzrinne, 1996]. Sidst i afsnittet findes er række definitioner, der kan bruges som nomenklatur.
Da dele af RTP headeren's opbygning benyttes under implementetionen, gennemgåes denne mere detaljeret end RTCP headerens opbygning.

A.1 Introduktion

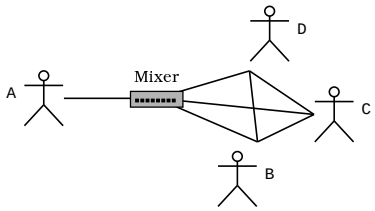
RTP er en Internet-standard protokol til transport af real-time data, som f.eks audio og video. Primært er det udviklet til brug for interaktive services som bl.a. internet telefoni.
RTP består af to dele, selve RTP pakken, til data overførsel, samt RTCP, der er en kontrol protokol. Data delen gør det muligt at dedektere pakke tab, rekonstruere rækkefølgen af en pakke sekvens, samt justere timing for data, der sendes kontinuert (f.eks audio og video). Kontrol delen gør det muligt at sende quality-of-service rapporter til kilden, der fortæller hvordan modtagerforholdene er på et givent tidspunkt.
RTP er udviklet til at være uafhængig af typen på det underlæggende transportlag, men benytter i mange implementationer UDP/IP.
En vigtig detalje ved RTP er at den ikke kan sikre real-tids levering af data. Principielt kan ingen ende-til-ende protokol sikre dette, det kan kun de underlæggende netværkslag, da det disse, der reelt har kontrollen over ressourcer i f.eks. routere og switche.
RTP er heller ikke istand til alene, at sikre fejlfri overførsel af data. Konsekvensen af det, må nødvendigvis afhænge af karakteristikken af det overførte data, da f.eks. retransmission af pakker i nogle tilfælde vil være spild af ressourcer, fordi data uanset hvad kommer for sent. I andre tilfælde vil det ikke spille nogen rolle at retransmittere data, og her tilbyder RTP udemærkede mekanismer for detektering af pakketab, der kan danne grundlag for en beslutning om retransmission.
En af forcerne ved RTP protokollen, er at den er udviklet til at håndtere multi-sessions, f.eks audio/video konferencer. Denne feature realiseres med bl.a mixere

og translators.
Følgende scenario beskriver kort mulighederne i en RTP session, bl.a. med brug af translator og mixer.
En audio konference, med 4 deltagere (se figur A.1). Gennem en allokerings mekanisme indsamles information om de deltagende parter, og de tilføjes til en multicast gruppe, denne information distribueres til alle deltagende parter. Audio data sendes nu i små enheder svarende til f.eks. 20 ms. Hver lille enhed præsenteres med en lille header, der bl.a beskriver kodec typen, hermed er det muligt at skifte kodec type, under selve sessionen / konferncen. Det kunne eksempelvis være nødvendigt, hvis en ny deltager slutter sig til fra en lavhastigheds forbindelse.
Dette kan dog undgåes ved brug af en mixer, som det kan ses i det følgende: Deltager A sidder ved et lavhastigheds opkobling, og tvinger derfor alle andre



Figur A.1: Audio konference med 4 deltagere. Deltager A sidder som den eneste ved en lavhastigheds opkobling.

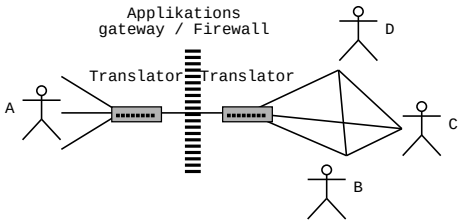
i konferencen til at benytte samme lave båndbredde, hviklet medfører kvalitets forringelse. Ved at indsætte en mixer, i overgangen mellem høj- og lavhastigheds nettet, bliver det muligt for deltagerne B, C og D at opretholde høj kvalitet, samtidig med at A kan udnytte sin forbindelse bedre, da han kun skal kommunikere med mixeren, fremfor de tre andre deltagere, (se figur A.2). Ydermere kan mixeren om nødvendig ændre kodec, af det data, der sendes til og fra A. I et



Figur A.2: Audio konference med 4 deltagere. Deltager A (med lavhastigheds opkobling) benytter nu en mixer.

andet tilfælde sidder deltager A bag en applikations gateway / firewall, der ikke direkte tillader IP data at passere. For at omgå dette problem installeres der en translator på begge sider af firewallen (se figur A.3). Translatoren på yder-

siden af firewallen, modtager alle multicasts, i forbindelse med RTP sessionen, og transporterer dem som unicast gennem en krypteret tunnel til translatoren indenfor firewallen, der konverterer unicast data til multicast data igen.

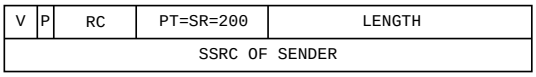


Figur A.3: Audio konference med 4 deltagere. Deltager A sidder bag en firewall / applikations gateway, der gør brug af to translator'e

Selvom RTP er udtænkt til at være en multisessions protokol, kan man uden problemer benytte den til unisessioner, som det ofte bliver i forbindelse med H.323 systemer, uden MCU'er.

A.2 RTCP

RTCP er som sagt den kontrolprotokol, der benyttes sammen med RTP. Den primære funktion for RTCP er at sende tilbagemeldinger til den aktuelle sender om, hvordan kvaliteten er hos de forskellige modtagere. Disse tilbagemeldinger kan f.eks. bruges til adaptiv ændring af kodec.
RTCP pakke formatet er opbygget på stortset samme måde som RTP, med en fast minimum header, efterfulgt af en struktur af forskellig indhold, med variabel længde. RTCP pakker bygges så de kan "stakkes", det vil sige enkeltdele af RTCP pakker kan stykkes sammen til en såkaldt compound pakke. RTCP headeren er opbygget på følgende måde (A.4):



Figur A.4: Bit formatet for RTCP headeren

Første segment på to bit beskriver RTCP versions nummeret, næste bit (padding) bruges til at fortælle om der er ekstra data, som ikke er en del af kontrol informationen. Næste segment (RC) angiver antallet af modtager rapporter, der findes i RTCP pakken. Herefter findes et 8 bit segment, der beskriver hvilken type RTCP der sendes (se indhold af RTCP pakker i A.2). Næste segment fortæller hvor lang RTCP pakken er. Sidste del af den faste header er en SSRC

identifikation, der fortæller hvem der har genereret denne RTCP pakke.
Indholdet i en RTCP pakker dækker bl.a. følgende:

- SR: Sender rapport, fra nuværende / aktive sender, med statistik for overførsel af RTP.
- RR: Modtager rapport, fra nuværende / aktive modtagere, med statistik for overførsel og modtagelse af RTP.
- SDES: Kilde beskrivelse, indeholdende info om sender.
- BYE: Farvel besked, der indikerer at deltager forlader konferencen.

Resten af RTCP pakken indeholder afhængig af type forskellige informationer, der ikke specificeres nærmere her.

A.3 Definitioner

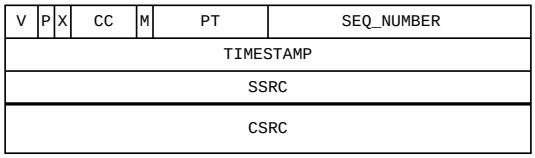
Følgende indeholder en række definitioner, der beskriver område specifikke enheder, når der snakkes om RTP.

- RTP Payload Den mængde data, der transporteres i en RTP Packet, f.eks. audio samples.
- RTP Packet En data pakke indeholdende en RTP header (med fast længde) og RTP Payload. Normalt vil RTP Packets blive transporteret enkeltvis af en underlæggende protokol, men der kan også indkapsles og transporteres flere sammen.
- RTCP Packet En kontrol pakke indeholdende en RTPC header (med fast længde), efterfulgt af en række data, afhængig af RTPC Packet typen. Normalt sendes flere RTPC pakker i en enkelt af de underlæggende pakker.
- Transport Address En kombination af en netværksadresse og port, der tilsammen identificere et endepunkt i OSI stakens transportlag. Eksempelvis en IP adresse og en TCP port. RTP pakker sendes fra senderens transport adresse til modtagerens transport adresse.
- RTP Session En association mellem en mængde enheder, der kommunikere via RTP. Ved multimedia sessioner, transporteres hver medium over seperate RTP sessioner, med hver deres RTPC pakker.
- Synchronization source (SSRC) Identifikation, der beskriver kilden (afsender) for en strøm af RTP pakker. Benyttes af modtager til at "gruppere" pakker ud fra "afsender". SSRC vælges vilkårligt, med intentioner om at ikke 2 "afsendere" inden for den samme RTP session har samme SSRC identifikation.

- Contributing source (CSRC) En liste til identifikation af bidragsydere, for en kombineret strøm af RTP pakker, genereret af en RTP mixer. Mixeren indsætter SSRC identifikationer for hver bidragsyder i CSRC listen. Dermed bliver det i en audiokonference muligt for en modtager at kende den pågældende taler, selvom SSRC identifikationen peger på mixeren.
- End System En applikation, der generere indhold, som skal sendes i RTP pakker, og/eller konsumerer indholdet, som modtages i RTP pakker.
- Mixer Enhed der modtager RTP pakker fra en eller flere kilder, og kombinerer pakkerne, eventuelt med timings tilpasning, synkronisering og ændring af data format.
- Translator Enhed der videresender RTP pakker, med SSRC identiteten intakt; f.eks som en applikations gateway i en firewall. Kan om nødvendig konvertere mellem forskellige kodeks. Yderligere er der mulighed for konvertering mellem unicast og multicast sessions.

A.4 RTP header information

Selve RTP pakkens header består af minimum 12 bytes, som vist på figur A.5.



Figur A.5: Bit formatet for RTP headeren

Første byte består af 4 segmenter, hvor de første 2 bit udgør versions nummer, der efter specifikationerne er 2. Næste bit er en padding bit, der bruges til at gøre opmærksom på, om der er yderliger data efter selve payload, f.eks i forbindelse med kryptering. Fjerde bit, en extension bit kan sættes i forbindelse med brug af headerextensions. Sidste segment i første byte er CSRC count, et tal for hvor mange CSRC identifikationer, der følger efter headeren.
Anden byte består af 2 segmenter, første bit heri er en marker bit, der fortolkes forskellig afhængig af profil (payload type). De sidste 7 bit angiver payload typen.
Herefter følger et 16 bit segment, der udgør et sekvens nummer, for den enkelte RTP session. Dette sekvens nummer tælles en op for hver RTP pakke der sendes, på den måde kan modtageren dedektere pakketab samt sortere pakker, så de kommer i rækkefølge, hvis de under netværksoverførslen kommer i uorden. Sekvens nummeret initialiseres random.
De næste 32 bit udgør en timestamp. Timestampen skal reflektere en samplings instans, der styres af en monoton og lineær inkrementerende klokke. Timestampen bruges bl.a til at beregne jitter og synkronisering. Ved periodisk genereret

RTP data, benyttes sampeltiden som klokke. Det vil sige at timestampen inkrementeres med en for hver sample. Også timestampen initialiseres random. De sidste 32 bit i den obligatoriske del af headeren angiver SSRC identifikationen. Denne værdi er statisk for den enkelte session, og vælges random for at undgå at 2 kilder får samme SSRC identifikation.
Herefter følger 0 til 15, 32 bit CSRC identifikationer, der hver opbygges som SSRC identifikationen.

Appendiks B

CAPI

B.1 Indledning

CAPI er et applikation programmerings interface (API), som giver en standardiseret adgang til ISDN udstyr, der er forbundet til enten en BRI eller en PRI ISDN forbindelse. CAPI bygger på ITU-T standarden Q.931, som er udviklet til at håndtere opkalds kontrol over ISDN.
CAPI tillader programmøren at abejde på et højere abstraktionsniveau, hvilket gør, at der er færre beskeder at holde styr på. Det er lettere at programere til CAPI end direkte til Q.931.

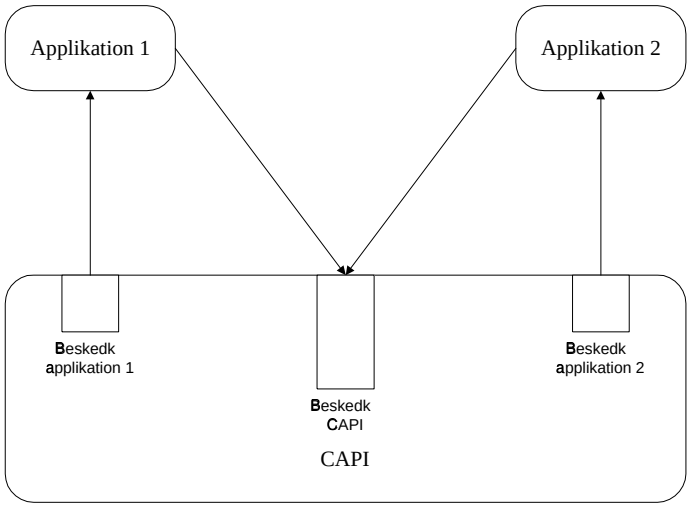
B.2 Den overordnede struktur i CAPI

B.2.1 Beskedkøer

CAPI er beskedorienteret, hvilket vil sige, at kommunikationen mellem CAPI og en given applikation foregår vha. beskeder. Beskederne, som sendes mellem CAPI og applikationerne gemmes i fifo buffere. Der er således en fifo buffer for hver applikation, som er registreret ved CAPI, hvor CAPI placerer sine beskeder til den pågældende applikation. Der er dog kun en buffer til beskeder til CAPI, hvor applikationerne kan placere deres beskeder til CAPI. Denne struktur i beskedkøerne, kan ses på figur B.1.

B.2.2 CAPI operationer

CAPI operationer anvendes til kommunikation mellem applikationer og CAPI. Inden en applikation kan sende beskeder til CAPI skal den registreres hos CAPI. Dette gøres med operationen CAPI_REGISTER, hvorefter CAPI giver applikationene et unikt nummer, som senere bruges til at identificere beskeder til og fra applikationen. Herudover laver CAPI en beskedkø til applikationen, hvor den senere vil placere sine beskeder til applikationen.
Beskederne sendes til CAPIs beskedkø fra applikationen med operationen CAPI-



Figur B.1: Beskedkøerne ved kommunikation mellem CAPI og applikationer.

_PUT_MESSAGE og hentes fra applikationens beskedkø med operationen CAPI_GET_MESSAGE.
Efter endt anvendelse af CAPI, frigøres applikationen ved at anvende operationen CAPI_RELEASE. Dette angiver til CAPI, at den pågældende applikation ikke længere vil anvende CAPI. CAPI vil derfor fjerne applikationens beskedkø, når applikationen sender CAPI_RELEASE.
Endelig er der operationer til at få information om versionen af den anvendte CAPI, serienummer m.m..
Alle operationerne, som er defineret for CAPI, er angivet i figur B.2.2.

Operation	Description
CAPI_REGISTER	Register an applikation
CAPI_RELEASE	Release an applikation
CAPI_PUT_MESSAGE	Transfer message to CAPI
CAPI_GET_MESSAGE	Get message from CAPI
CAPI_SET_SIGNAL	Register call-back function
CAPI_WAIT_FOR_SIGNAL	Wait for new message to be available
CAPI_GET_PROFILE	Get capabilities of CAPI implementation
CAPI_GET_MANUFACTURER	Get manufacturer identification
CAPI_GET_VERSION	Get CAPI version numbers
CAPI_GET_SERIAL_NUMBER	Get serial number
CAPI_INSTALLED	Check wheter CAPI is installed
CAPI_MANUFACTURER	Manufacturer-specific function

Figur B.2: Operationer i CAPI.

B.2.3 CAPI beskeder

I forbindelse med udvekslingen af beskeder mellem applikationer og CAPI anvendes handshake. For hver besked, som sendes til modtageren, sender denne en besked tilbage til senderen. Pga. denne egenskab deles beskederne op i fire typer og navngives på følgende måde:

- Beskeder fra en applikation til CAPI kaldes requests (_REQ)
- Tilbagemeldinger fra CAPI til en applikation i forbindelse med et Request kaldes confirmation (_CONF)
- Beskeder fra CAPI til en applikation kaldes indications (_IND)
- Tilbagemeldinger fra applikationen i forbindelse med en Indication fra CAPI kaldes response (_RESP)

Alle beskeder, som sendes til CAPI har det samme format. Først i beskeden er der en header med en fast længde. Efter denne kommer de parametere, som skal være med i beskeden.
Headeren indeholder information om den totale længde af beskeden i bytes, hvilken applikation beskeden er til eller fra, hvilken type besked det er (kommando), type af besked (subcommando: request, confirmation, indication eller response) og et unikt besked nummer. Headerens struktur kan ses i figur B.3

Message	Type	Contents
Total length	word	Total length of the message including the complete message header
ApplID	word	Identification of the application. The application number is assigned to the application by CAPI in the CAPI_REGISTER operation
Command	byte	Command
Subcommand	byte	Command extension
Message number	word	Unique message number

Figur B.3: Beskedheaderens opbygning.

Parameterne, som hører til beskeden, som skal sendes skal være opbygget efter anvisningerne i CAPI dokumentationen [capi.org, 1999].
Headeren og parameterne sendes til CAPI med operationen CAPI_PUT_MESSAGE.

Appendiks C

Dekodning af DTMF-signaler

På terminalsiden af vores system er der tilsluttet PSTN-enheder vha. den i kravspecifikationen beskrevne grænseflade. DTMF-signaler (Dual Tone Multiple Frequency) bruges af PSTN-enheder til at sende information om brugerens tryk på enhedens taster. Vi har valgt at dekode disse signaler i SW bl.a. for at spare på HW-komponenter. Signalet fra en PSTN-enhed sendes ind i PC'en og samples via lydkortet (i samme kanal som tale). Dette afsnit beskriver udviklingen af en algoritme (som udnytter DSP) til dekodning af DTMF-signaler. Algoritmen udvikles efter kravene fra Q.24 anbefalingen "Multifrequency Push-Button Signal Reception"fra ITU-T. Kravene varierer lidt for forskellige lande og standarder. I anbefalingen følges kravene for "Danish Administration".

C.1 DTMF-specifikationer

C.1.1 Frekvenser

Høje frekvenser / Lave frekvenser	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Figur C.1: Tastaturets DTMF-signaler og de tilhørende frekvenser.

En PSTN-telefon har typisk et 12- eller 16-tasters tastatur som vist på fig. C.1. Tasterne ABCD benyttes dog ikke til at oprette samtaler og er sparet

væk på de fleste telefoner, hvorfor der kun udvikles 12-tasters dekodning. Når brugeren trykker på een af tasterne på tastaturet, skabes der et signal bestående af to frekvenser. De to frekvenser kommer fra hver sin frekvensgruppe. Den lave frekvensgruppe består af tonerne under 1kHz: 697, 770, 852 og 941 Hz. Den høje frekvensgruppe består af tonerne over 1kHz: 1209, 1336, 1477 og 1633 Hz. Hver tone i den lave frekvensgruppe svarer til en række på tastaturet og hver tone i den høje frekvensgruppe svarer til en kolonne på tastaturet. Når der trykkes på en tast, frembringes der en lyd bestående af de to frekvenser, som svarer til den række og kolonne, som tasten befinder sig i.
Kun een frekvens fra hver frekvensgruppe må være tilstede for en gyldig genkendelse af en tast.
Frekvensernes tolerance er ± (1,5 % + 2Hz). Under denne skal tonerne genkendes. Det er ikke specificeret, hvornår tonerne ikke må genkendes, dog fremhæves det, at brede grænser giver dårlig taleimmunitet.

C.1.2 Niveauer

Effektniveauerne er ikke specificeret præcist i Q.24 anbefalingen. For at en tone skal opfattes som gyldig, skal effektniveauet ligge mellem (A+25) og A dBm, hvor A ligger mellem -22 og -30 afhængigt af land. For at en tone ikke skal opfattes som gyldig, skal niveauet være på under (A-9) dBm. Hvis lydkortets optage-lydstyrke justeres korrekt, kan disse effektniveauer bruges relativt til lydkortets maksimale amplitude. Således skal en gyldig tone ligge mellem 0 og -25 dB, og en ugyldig tone skal ligge under -34 dB, i forhold til lydkortets maksimale amplitude.
Forskellen i effektniveauet mellem en frekvens fra den lave gruppe og en frekvens fra den høje gruppe må maks. være 6 dB for at frekvenserne må opfattes som gyldige.

C.1.3 Timing

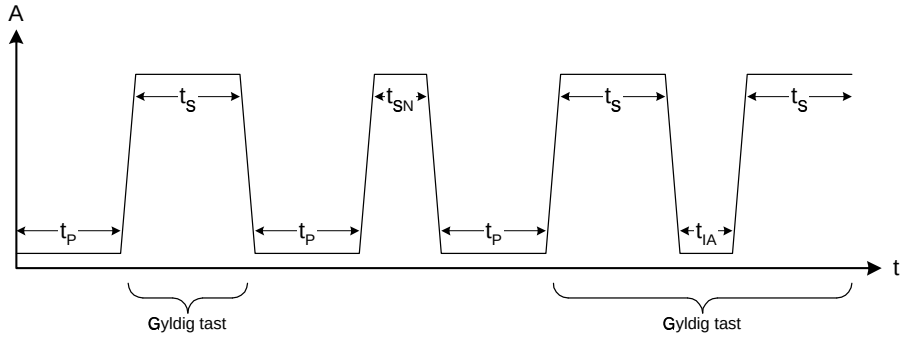
For at et DTMF signal bliver genkendt ved dekodning, skal signalet have en mindste varighed. DTMF signaler under en bestemt varighed må ikke genkendes. En pause mellem to ens DTMF signaler skal have en bestemt mindste varighed, for at signalerne bliver registreret som to efterfølgende tryk på en tast. Pga. støj/fejl kan der opstå afbrydelser i et DTMF signal. Afbrydelser under en bestemt varighed tolereres og registreres ikke som en pause. Værdierne for de beskrevne størrelser ses i fig C.2 og i figur C.3 ses et timingsdiagram med størrelserne.

C.1.4 Taleimmunitet

En DTMF-dekoder skal ikke registrere almindelig tale som DTMF-signaler. I Q.24 anbefalingen er det specificeret, at der højest må registreres 46 falske tegn under 100 timers tale med et gennemsnitligt niveau på -12 dBm.

Str.	Beskrivelse	Værdi
t_S	Signalets mindste varighed for genkendelse	Min. 40 ms
t_{SN}	Signalets største varighed for ikke-genkendelse	Max. 20 ms
t_P	Pausens mindste varighed for genkendelse	Min. 40 ms
t_{IA}	Afbrydelsens største varighed for ikke-genkendelse	Max. 20 ms
	Signaleringshastighed	Max. 10 taster/s

Figur C.2: Værdier for timing ved dekodning af DTMF-signaler.



Figur C.3: Timing af DTMF-signaler.

C.1.5 Ringetone

Når det første DTMF-tegn skal dekodes, vil der i PSTN-enhedens højttaler være en ringetone (dial-tone), som i E.180.S2 anbefalingen er specificeret til at være en tone med frekvensen 425 Hz (niveauet af denne er i Q.35 anbefalet til under -10 dBm). Denne tone vil i nogen grad også optræde på lydindgangen til vores system. Dekoderen skal kunne fungere med denne tone.

C.2 Realisering

Dekodningen af DTMF toner kan opdeles i følgende faser:

- Frekvensanalyse
- Lydniveauer
- Kombinatorisk logik
- Timing

C.2.1 Frekvensanalyse

Da DTMF toner skal genkendes udfra signalets frekvensindhold er en mulig angrebsvinkel at lave en form for fourieranalyse af signalet. FFT er en ofte anvendt algoritme til at lave fourieranalyse, men egner sig ikke i dette tilfælde,

da vi kun er interesserede i at analysere syv enkelte frekvenser. Derfor beregnes syv udvalgte koefficienter af en DFT (lign. C.1).

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn} \tag{C.1}$$

N er vindueslængden og skal bestemmes således at frekvensopløsningen bliver høj nok til at gøre det muligt at skelne mellem de forskellige frekvenser. En høj N giver en lav tidsopløsning. Vi har valgt N = 105 (ved 8kHz samplerate), da denne vindueslængde lader de aktuelle testfrekvenser ligge tæt på de nominelle samtidig med at frekvenserne ligger så tæt som muligt (se fig. C.4).

Nominel testfrekvens	**Aktuel testfrekvens**	**k**
697	686	9
770	762	10
852	838	11
941	914	12
1209	1219	16
1336	1371	18
1477	1447	19

Figur C.4: Nominelle og aktuelle DTMF testfrekvenser.

Ingen vinduesfunktion er anvendt, da dette giver den smalleste hovedsløjfe, hvilket er vigtigere end en lav sidesløjfe, da frekvenserne ligger lige op ad hinanden i det diskrete spektrum. Med så tætliggende frekvenser er lækagen ret stor, og vi kan ikke opfylde kravene til lydniveauerne. Et længere vindue ville dog gøre det sværere at opfylde kravene til timing.

C.2.2 Lydniveauer

Når signalet er blevet analyseret ved de syv frekvenser, haves syv koefficienter, der repræsenterer energien ved de syv frekvenser. Disse koefficienter skal nu sammenlignes med niveaukravene fra Q.24 standarden, og returneres som boolske koefficienter. Niveauer på under -34dB i forhold til lydkortets maksimale amplitude, skal opfattes som ugyldige, mens niveauer på over -25dB skal accepteres. Pga. lækage lader vi alle niveauer på under -18dB være ugyldige og alle niveauer på over -18dB være gyldige. Kun een koefficient fra den øvre frekvensgruppe og een fra den nedre frekvensgruppe må være opfattet som gyldig. Er dette ikke opfyldt, sættes alle syv boolske koefficienter til nul. Hvis det er opfyldt, sammenlignes de to koefficienters niveau. Forskellen må maksimalt være på 6dB. Er forskellen større, sættes alle boolske koefficienter til nul.

C.2.3 Kombinatorisk logik

I dette trin findes det DTMF-tegn, der svarer til de syv boolske koefficienters værdi. Passer værdierne ikke til noget gyldigt tegn, returneres tegnet "?".

C.2.4 Timing

DFT vinduet flyttes 105 samples for hver DFT, svarende til 13,125 ms. Med denne periode modtages altså enten gyldige eller ugyldige tegn. Disse tegn skal samles således at timingen i Q.24 overholdes. Timingen i afsnit C.1.3 viser at der skal times efter to forskellige tider: 20 ms og 40 ms. 13,125 ms passer ikke direkte til disse tider. En mulig fortolkning af indkomne tegn er:

- $13,125ms$ er for kort til opfattelse som et tegn eller en pause.
- $2 \cdot 13,125ms$ er nok til opfattelse som et tegn eller en pause.

Denne mekanisme kan implementeres vha. en tæller, som tæller antallet af efterfølgende tegn der er ens.

C.2.5 Test

Algoritmen er kun blevet testet på software-genererede DTMF-toner uden støj. Under denne test opførte algoritmen sig fuldt ud tilfredsstillende. En realistisk test kan kun udføres på DTMF-toner fra en ægte PSTN-telefon eller et testbånd.

Appendiks D

Generering af PSTN-toner

Afhængigt af en PSTN-enheds status, skal systemet på terminalsiden udsende forskellige toner til PSTN-enheden via lydkortet. Disse toner skal altså skabes digitalt i SW.

D.1 Specifikationer

D.1.1 Frekvenser og timing

I ITU-T anbefaling E.180.S2 findes en tabel over forskellige toner i forskellige lande. Værdierne for Danmanrk ses i fig. D.1.

Tone	**Frequency (Hz)**	**Cadence (Seconds)**
Dial tone	425	Continuous
Ringing tone	425	1,0 - 4,0
Busy tone	425	0,25 - 0,25

Figur D.1: Frekvenser og intervaller for forskellige toner i Danmark (fra E.180.S2).

D.1.2 Niveauer

I ITU-T anbefaling Q.35 anbefales det, at signalernes niveau er mellem -5 og -15 dBm. Dog anbefales det, at ringetonen er under -10 dBm, da den ellers kan forstyrre DTMF-dekodning.

D.2 Realisering

De angivne signalniveauer omsættes til dB relativt til lydkortets maksimale amplitude. Signalerne genereres vha. sinusfunktionen, og funktionerne indkapsles i en klasse, som kan arbejde på variabel lydbufferstørrelse.

Appendiks E

DirectSound

Lyden fra/til PSTN-enheder sendes via lydkortet. Vi har benyttet DirectSound 7.0 API'en for at gøre kommunikationen med lydkortet universel. DirectSound sørger selv for at snakke med lydkortets driver og for at lave en eventuel konvertering af antallet af bits til det ønskede.
For at indkapsle detalierne omkring API'en, har vi lavet en DirectSoundPlayback- og DirectSoundRecord-klasse.
DirectSoundPlayback-klassen stiller følgende metoder til rådighed:

- BOOL detected() - Returnerer om directsound er tilstede og om det er muligt at initialisere.
- BOOL setformat(int channels,int bitspersample,int samplerate,int buffersize) - Sætter formatet af lyddata og opsætter buffere. Skal kaldes inden lock/unlock/play/stop.
- BOOL setnotifications() - Opsætter tre signaleringshændelser: En i starten af lydbufferen, en i midten og en når man stopper afspilningen. Disse signaleringer bruges under afspilning til at finde ud af hvornår en bestemt halvdel af bufferen er klar til aflæsning.
- BOOL lock(int offset,int size,LPVOID *where) - Bruges til at låse en halvdel af bufferen, når der ønskes skrivning af lyddata til denne.
- BOOL unlock() - Frigiver en låst bufferhalvdel.
- BOOL play(BOOL loopsample) - Starter afspilning af bufferen.
- BOOL stop() - Stopper afspilning af en lydbuffer.

DirectSoundRecord-klassen stiller præcis samme metoder til rådighed som DirectSoundPlayback-klassen. Når der skal anvendes full-duplex, skal først afspilningsklassen initialiseres, hvorefter inspilningsklassen må initialiseres. Alle kald returnerer en boolsk værdi, der angiver om alt gik vel.