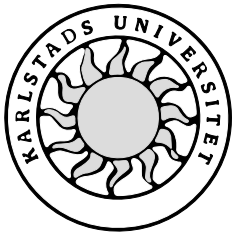


Datavetenskap

Berndt-Uno Nyström

Hasse Ellström

Utveckling och design av multimedialaboration med SIP-signalering för Datakommunikation II

Examensarbete, C-nivå

2002:03

Utveckling och design av multimedialaboration med SIP-signalering för Datakommunikation II

Berndt-Uno Nyström

Hasse Ellström

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Berndt-Uno Nyström

Hasse Ellström

Godkänd, 2002-06-12

Handledare: Annika Wennström

Examinator: Martin Blom

Sammanfattning

Vår del av det större projektet, att modernisera E-laboratoriet, har varit att producera en signalerings- och multimedialaboration för datakommunikation II på C-nivå som åskådliggör hur signalering i datakommunikation fungerar. Examensarbetet bygger på det nya SIP-protokollet vilket ligger till grund för en större multimediaarkitektur. En SIP-server har köpts in av Karlstads universitetet vilken vi installerat och konfigurerat. Till ändapplikationerna har vi använt oss av en SIP-stack med öppen källkod**,** där vi modifierat ett medföljande exempelprogram för att passa till laborationen. I den första delen av laborationen skall studenterna göra uppkopplingar, granska nätaktiviteten och beskriva signaleringsvägarna med sekvensdiagram. Även viss kodning och konfiguration skall utföras av studenterna. Den sista laborationsdelen består av en multimediasession där olika bildkodningar skall jämföras och beskrivas samt en uppkoppling över ett trådlöst nätverk till en bärbar dator, för att se hur yttre faktorer påverkar mediaöverföringen.

Development and design of a multimedialaboration including SIP-signaling for Datacommunication II

Abstract

Our part of the E-laboratory modernization project has been to design a laboration describing signaling and streaming media sessions for the C-level course Datacommunication II. The Bachelor's Project is built on the new SIP-signaling protocol which is a part of a wider multimedia architecture. A SIP-server, installed and configured by us, was bought by the University. For building User Agents was an open source SIP-stack including an example program used. We have modified the example program to fit the laboration. The laboration contains two parts. The first part includes making SIP-connections, studying the network activity and illustrating in a sequence diagram how the signaling acts. Some programming and configuration has to be done to fulfill the laboration. The last part of the laboration includes making a multimedia connection between two stationary computers and find out the result of several types of encoding for video and a connection between a stationary computer and a laptop over a wireless network, realizing how the signal is affected by the environment.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Uppsatsen är ett arbete som vi genomfört under våren 2002 och vårt uppdrag har varit att ta fram en ny laboration för kursen Datakommunikation II vid institutionen för informationsteknologi, avdelningen för datavetenskap, på Karlstads universitet. Laborationen och därmed vårt arbete består av två huvuddelar, en signaleringsdel och en multimediadel. De kommer att gå i varandra och visa hur signalering och multimediaströmmar kan gå olika vägar genom ett nätverk samt hur bildkodning och länkkaraktäristik påverkar mediaöverföring.

1.1 Plattform för datakommunikation

Utvecklingen inom datavetenskap och speciellt inom datakommunikation har under de senaste åren gått rasande fort. Universitetets målsättning är att ligga i främsta ledet och ge kurser som motsvarar förväntningarna från arbetsmarknaden.

För att kunna ge studenterna en bra utbildning i datakommunikation behövs en uppgraderad plattform där avancerade laborationer, framförallt på C-nivå, kan utföras.

Eftersom befintligt laboratorium blivit föråldrat har ett projekt för modernisering initierats. Projektet är ett samarbete mellan Karlstads universitet och Ericsson Infotech AB vilka gått in som delfinansiär med både kunskap och hårdvaror. Som en del i projektet ingår att producera en ny laboration för kurs i datakommunikation på C-nivå.

1.2 Mål - ny laboration

Vårt examensarbete innebär att producera en laboration vilken består av två delar. Första delen innebär att med Session Initiation Protocol (SIP) [1][5][7] skapa en applikation där signaleringens respektive datatransmissionens vägar genom ett nätverk åskådliggörs. Andra delen innehåller en multimediaöverföring över ett trådlöst nätverk där variationer i kvalitén beroende på avståndet påvisas samt hur kvalitén påverkas vid olika bildkodningar.

Syftet med laborationen är att ge studenterna:

- Förståelse för begreppet signalering i allmänhet, och skillnaderna mellan signalering i datakommunikationsvärlden kontra telekommunikationsvärlden i synnerhet.
- Förståelse för SIP och en introduktion till att läsa standardiseringsdokument.

- Vana vid att använda mediaprogramvara.
- Viss kunskap om hur länkkarakteristiken påverkar medieöverföringen.

De krav som finns på laborationen är:

- En total tidsåtgång på 20 – 25 timmar fördelat på:
  a) Med hjälp av givna byggklossar göra en SIP-klient, en så kallad User Agent (UA). Denna kan registrera sig hos en SIP-proxyserver, 10 timmar.
  b) Konfigurera en Media Gateway för att uttryckligen se skillnaden mellan signalering och mediaväg, 2 timmar.
  c) Genomföra och analysera ett antal uppkopplingsfall med lämplig programvara för protokollanalys, exempelvis Ethereal, 3 timmar.
  d) Skapa en förbindelse till en bärbar dator ansluten via ett trådlöst nätverk och undersöka hur mediakvalitén påverkas med ökat avstånd och olika kodning, 2 timmar.
  e) Redovisa resultatet i en skriftlig rapport som huvudsakligen redovisar punkterna c och d ovan, 5 timmar.
- Laborationen skall kunna utföras på standarddatorer med operativsystem Linux eller Windows NT.
- Laborationen skall kunna genomföras utan speciella privilegier såsom root eller administratör.
- Lösningar som innebär att handledare eller systemadministratör måste göra omkonfigurationer mellan laborationsgrupperna bör undvikas.
- Vid val mellan olika lösningar skall möjligheten till utvidgning och påbyggnad beaktas.

1.3 Uppsatsens upplägg

Vi börjar med att beskriva förutsättningarna för vårt arbete i detta kapitel, bl a projektet som den nya laborationen är en del av och innehållet i kravspecifikationen för laborationen som utarbetats av institutionen. Kapitel 2 innehåller en beskrivning av SIP-protokollet som är en central del i examensarbetet och signaleringsdelen av laborationen. De applikationer som vi använder för att bygga signaleringsdelen av laborationsplattformen presenteras i kapitel 3. Det är framförallt en inköpt SIP-server (SIPd) samt en SIP-stack (oSIP) med öppen källkod som beskrivs men också andra applikationer som är nödvändiga för att få full funktion.

Kapitel 4 inleds med en genomgång av övriga applikationer som används i laborationen bl.a. nätavlyssningsprogram och multimediaapplikationer. Därefter visar vi de ändringar vi

gjort i källkod för att anpassa programvara efter laborationen och fortsätter med hur en laboration genomförs i avsnitt 4.2 (signalering) och 4.3 (multimedia). Laborationsspecifikationen finns med som bilaga A. I slutkapitlen 5 och 6 skriver vi om våra erfarenheter, bl a svårigheter att installera programvara, och möjligheter för vidare påbyggnad av vårt arbete samt slutsatser för arbetet.

2 Session Initiation Protocol

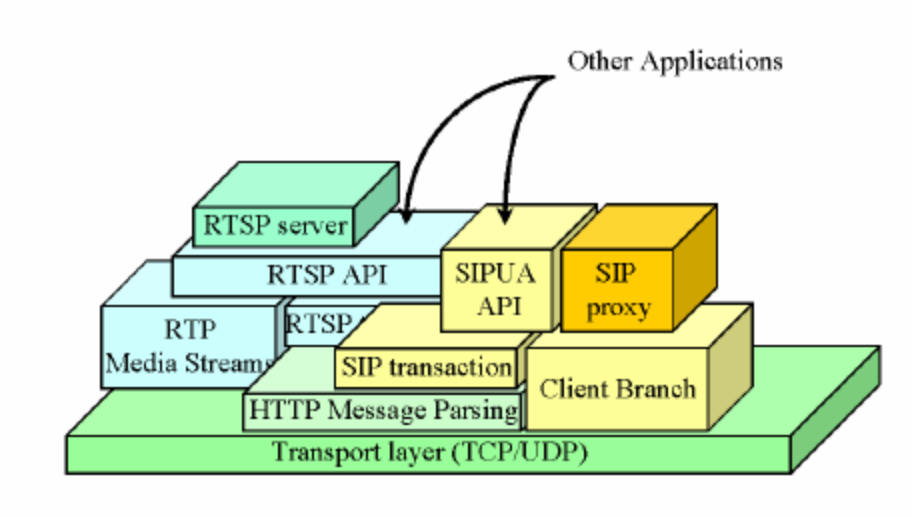
För att kunna förstå laborationens signaleringsdel är det viktigt med kunskap om SIP-protokollet och den omgivning det arbetar i. Vi har försökt att sammanfatta RFC 2543 [7] samt annat som publicerats i böcker och artiklar om SIP. Den http-liknande syntaxen är lätt att förstå och uppdelningen mellan header och data är välkänd för den som sett data-kommunikationsprotokoll förut. Tre olika servrar fungerar som nav i en SIP-arkitektur och vi beskriver dessa samt några sekvensdiagram som visar hur servrarna ingår i signaleringsvägen.

2.1 Arkitektur

SIP är från början ett signaleringsprotokoll vilket används för kontrollinformation mellan en eller flera enheter i syfte att initiera, kontrollera och bryta en nätverksession. SIP är en del av en större multimediaarkitektur som utvecklats under de senaste åren. I den ingår protokoll för överföring av realtidsmedia såsom Internet-telefonering, videokonferenser men även till idag befintliga nätverksapplikationer såsom vanlig telefon eller mobiltelefon. SIP-systemet fungerar liknande det etablerade e-mail-systemet med egna användaradresser, sk SIP-adresser, vilka kan lagras hierarkiskt på liknande sätt som Domain Name System (DNS) [10] gör.

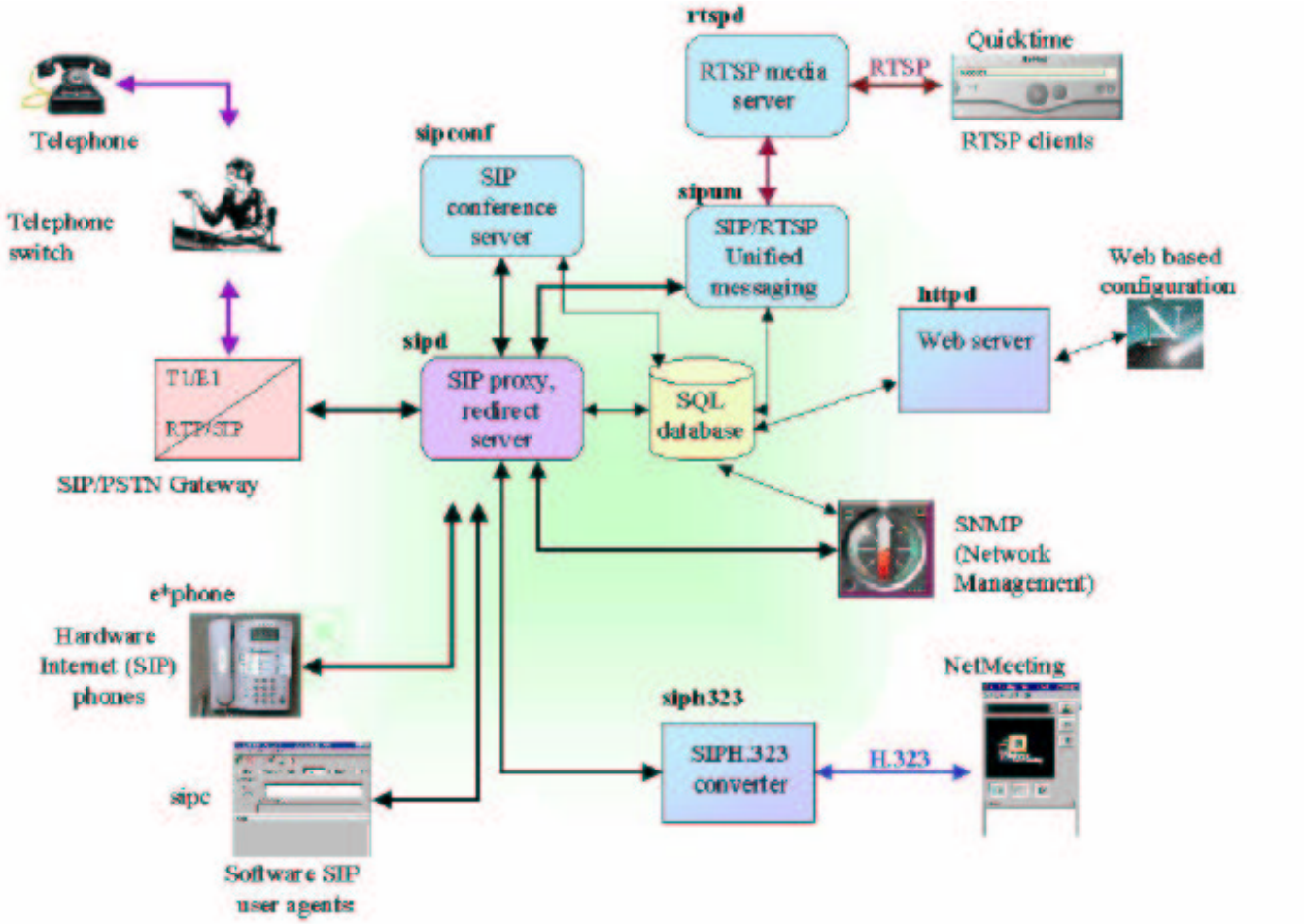
En av de stora skillnaderna mellan SIP och övriga signaleringsprotokoll är att SIP kan förmedla förfrågningar om mediauppkopplingar till enheter på flera geografiskt olika platser, endera samtidigt eller i en förutbestämd ordning om föregående förfrågan misslyckades. Samtal kan vidarekopplas till den eller de platser och enheter där man för tillfället befinner sig vilket innebär att adressen inte är knuten till en fast enhet utan en användare kan ta emot en uppkoppling där han för tillfället befinner sig.

De protokoll som ingår i multimediaarkitekturen och visas i Figur 1 är bl.a. SIP, Real-Time Transport Protocol (RTP) för transport av audio, video och andra tidskänsliga data, Real-Time Streaming Protocol (RTSP) för kontroll av strömmande media som är lagrade i filer. I SIP-systemet ingår även protokoll bl.a. för resursallokering, multicast-adressering och IP-telefonering.



Figur 1: Protokollstruktur över multimediaarkitekturen.

Figur 1 och Figur 2 är hämtade från Columbia University [14] och de med fet stil angivna texterna i Figur 2 är applikationer producerade av SIP Communications, Inc. (SIPCOMM), New York, USA. SIP-servern är kärnan i arkitekturen och samverkar med de olika komponenterna enligt Figur 2. De ur vår synvinkel intressanta komponenterna är, förutom servern (SIPd) som beskrivs ingående i ett eget kapitel, SQL databas, webserver och en User Agent där vi använder en applikation med öppen kod, oSIP, istället för SIPCOMMs sipc. Övriga komponenter, vilka inte kommer att ingå i vårt arbete, visar exempel på SIP-arkitekturens möjligheter.



Figur 2: Multimediaarkitekturen och interaktionen mellan de olika komponenterna.

2.1.1 SIP-protokollet

Användarapplikationerna belägna på ändstationerna i en session kan vara en telefon, stationär eller mobil, men också en mjukvaruapplikation för media- eller videokonferens på en dator. Det är användarapplikationerna som initierar en session genom att anropa en proxyserver med en INVITE-begäran och som efter uppkopplingen erhåller en direktförbindelse för dataöverföring med en eller flera andra användarapplikationer.

SIP är ett protokoll som i Internets protokollstack tillhör applikationslagret. Protokollet kan använda sig av vilket transportprotokoll som helst, men eftersom stöd för mekanismer liknande de i TCP finns, bl.a. flödeskontroll och omsändningar av meddelanden, kan man med fördel använda UDP.

Ett SIP-meddelande är skrivet i klartext, med en syntax liknande den http har, och innehåller ett antal fält beskrivna i Figur 3.

```
INVITE: sip:jacK@atosc.org SIP/2.0
Via: SIP/2.0/UDP home.sipworld.org
To: sip:jacK@atosc.org
From: sip:cha@sipworld.org
Call-ID: 35778645354@home.sipworld.org
CSeq: 1 INVITE
Contact: sip:cha@home.sipworld.org
Content-type: application/sdp
Content-length: 267

v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Figur 3: Exempel på SIP-meddelande.

Första raden består av vilken metod (method), SIP-adress samt vilken version av protokollet som meddelandet innehåller.

De olika metoderna är:

INVITE:	En uppkopplingsbegäran till en annan användare.
ACK:	Används för att bekräfta en INVITE-begäran.
REGISTER:	Används för att registrera i en SIP registrarserver vilken adress användaren för tillfället befinner sig på.
OPTION:	Används för att ta reda på en ändapplikations kapacitet.
CANCEL:	Ett meddelande för att terminera en sökning efter en användare.

BYE: Informerar en (eller flera) användare att användaren kommer att lämna sessionen.

Exempel på övriga fält i headern är:

Via: Är en eller flera rader som talar om vilken väg meddelandet gått, t.ex. om meddelandet har gått via en eller flera proxyservrar. Första via-raden anger avsändaren.

To: Anger SIP-adressen till mottagaren.

From: Anger SIP-adressen från den anropande användaren.

Call-ID: En unik identifierare för uppkopplingen.

Cseq: Ett sekvensnummer för anropet (request) samt typen av metod.

Contact: Den Unified Resource Locators (URL) där avsändaren befinner sig.

Content-type: Typen av innehåll i datafältet (payload).

Content-length: Storleken på datafältet i bytes.

Datafältet avgränsas från headern med en radmatning.

2.1.2 Servrar

Det finns flera olika typer av servrar, proxy, redirect och registrar. Dessa illustreras i Figur 2. De nedan beskrivna servrarna implementeras vanligtvis i samma applikation.

De vanligaste är:

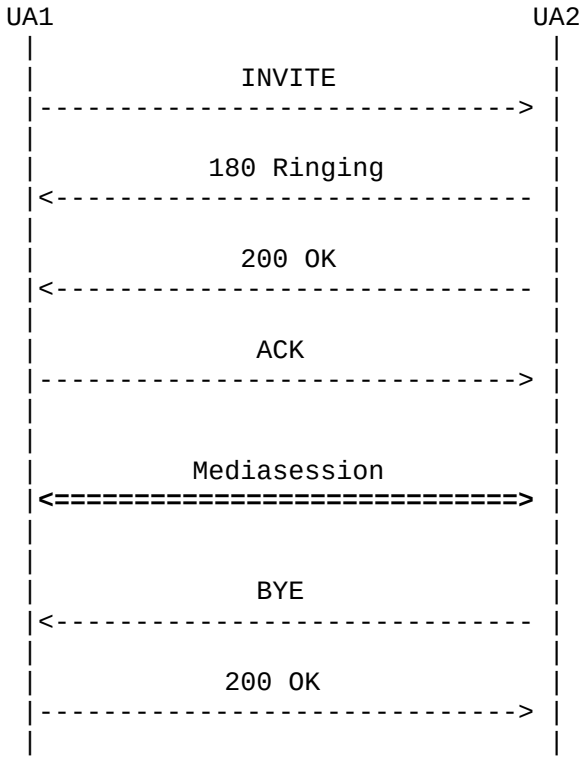
- **Proxy:** En mellanliggande enhet som lyssnar efter SIP-anrop, vanligtvis på port 5060. Den agerar både som klient och server av den anledningen att den förmedlar förfrågningar mellan klienter. Dess huvudsakliga uppgift är att routa meddelanden till en enhet, t.ex. en server, belägen närmare mottagaren. Den har även till uppgift att kontrollera användar-policies t.ex. om användaren skall tillåtas göra en uppkopplingsbegäran. En proxy lägger till sin egen adress i via-fältet när ett meddelande vidareförmedlas.

- **Registrar:** En användare måste finnas registrerad för att kunna ta emot ett SIP-anrop via en proxyserver. Registrarservern tar emot registreringarna från användarna. Den används av både proxyservern och redirectservern för att de skall få information om vart meddelanden skall vidareförmedlas.

- **Redirect:** En server som mappar en SIP-adress med en eller flera nya adresser samt returnerar dem till den klient som gör en uppkopplingsförfrågan. Avsändaren får sedan själv göra en ny uppkopplingsförfrågan med den nya adressen.

2.2 SIP-uppkopplingar

En uppkoppling kan göras direkt mellan två användare eller via en SIP-server. Figur 4 beskriver i ett sekvensdiagram hur en uppkoppling mellan två användare ser ut. För att initiera en uppkoppling skickas ett INVITE-meddelande. Den mottagande enheten sänder ett Ringing-meddelande tillbaka för att bekräfta att meddelandet kommit fram så att mottagaren inte behöver göra omsändningar samt påkallar den mottagande partens uppmärksamhet. Detta är analogt med ett vanligt telefonsamtal och flera ringsignaler kan göras innan ytterligare utbyten sker. När den mottagande parten svarar bekräftas detta genom att ett OK-meddelande sänds till den initierande parten vilken återsänder ett ACK-meddelande för att slutföra uppkopplingen. En förbindelse för en mediasession är nu etablerad. För att bryta ned sessionen kan båda parter sända ett BYE-meddelande vilken bekräftas av den andra parten med ett OK-meddelande.



Figur 4: SIP-uppkoppling mellan två användare.

I en arkitektur där servrar används och båda användarna är registrerade i samma domän kan en uppkopplingsförfrågan se ut som exemplet i Figur 5. För att kunna etablera en uppkoppling via proxyservrar måste åtminstone den mottagande parten vara registrerad. Detta görs med ett REGISTER-meddelande till en registrarserver, oftast via en proxyserver som vidareförmedlar registreringen.

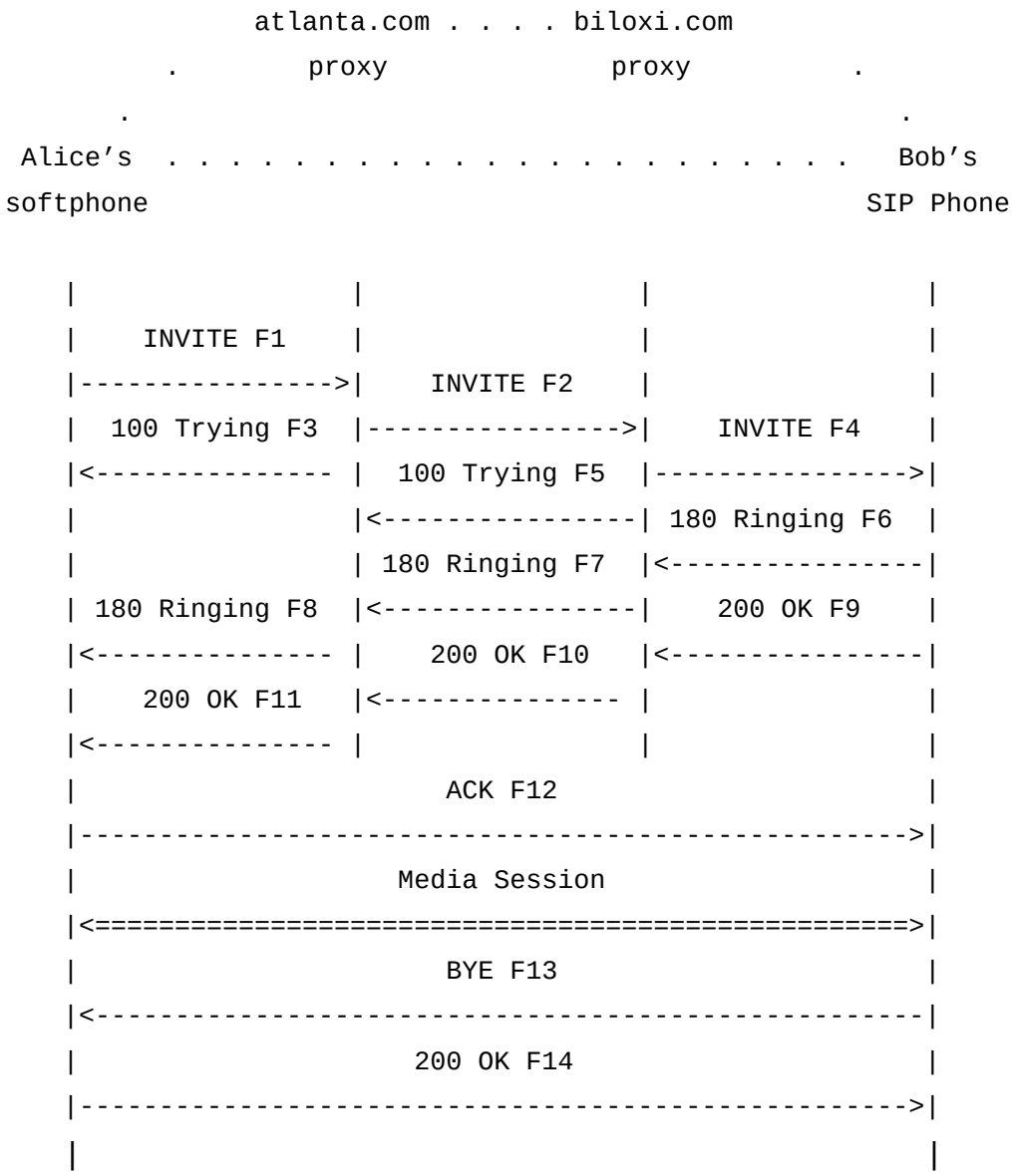
I exemplet i Figur 5 registrerar sig båda parter. När registreringen är gjord kan ett INVITE-meddelande från UA1, där användaren Bob finns, innehållande mottagaradressen, jack@home.net, sändas till proxyservern vilken agerar som redirectserver och returnerar den temporära adressen. Användarapplikationen får meddelandet att mottagaren är tillfälligt omdirigerad till en annan adress, jack@office.atosc.org.

UA1 sänder en bekräftelse på omdirigeringen, i form av ett ACK-meddelande till proxyservern och gör ett nytt INVITE-anrop, nu med den nya adressen till UA2 där Jack för tillfället befinner sig. UA2 påkallar användarens uppmärksamhet och skickar Ringing-meddelanden till UA1 till dess Jack svarar. Detta bekräftas med ett OK-meddelande till UA1 varvid UA1 skickar en ACK-signal som verifiering på att mediakanalen är etablerad.

SIP kan användas för ändra en mediakanal. I exemplet görs detta av UA2 som vill byta från audio till audio + video. Sessionen modifieras med ett INVITE-meddelande från UA2 innehållande SIP-adressen till Bob, bob@bob.home.net och en förhandlingförfrågan med parametrar för bildöverföring. Denna bekräftas med ett OK-meddelande av UA1 och media-överföringen kan fortsätta. När sessionen skall avbrytas skickar UA1 ett BYE-meddelande som bekräftas av UA2 med ett OK-meddelande.

Figur 5: SIP-uppkoppling med proxyserver.

I Figur 6 åskådliggörs en uppkoppling med servrar som befinner sig i olika domäner. Här ser man hur de olika meddelandena som går mellan ändenheterna passerar och vidarebefordras av proxyservrarna under uppkopplingen. Varje proxyserver som vidareförmedlar en INVITE, F1,F2 och F4, skickar samtidigt tillbaka ett Trying-meddelande till den nod varifrån meddelandet kom, t. ex. avsändaren (Alice's softphone), F3, eller föregående proxyserver, F5. Trying-meddelandet indikerar aktivitet så att inte omsändningar görs. Ringing- och OK-meddelandena, F6-F11, fungerar som beskrivits i exemplet i Figur 5. Från det att ACK-meddelandet, F12, sänds har ändenheterna direktkontakt med de framförhandlade sessionsparametrarna. I figuren går ACK-meddelandet direkt till den uppringda parten men kan även konfigureras så att den går via kedjan av proxyservrarna. BYE-meddelandet, F13, och det följande OK-meddelandet, F14, vilka går direkt mellan avsändare och mottagare, avslutar uppkopplingen.



Figur 6: Exempel på uppkoppling med flera proxyservrar.

3 Implementation av SIP-applikationer

I signaleringsdelen av laborationen används framförallt två SIP-produkter, SIPd, en server som köpts in av institutionen och oSIP, en SIP-stack med öppen källkod som vi använder till UA-applikationerna. Vi räknade med att servern enkelt kunde installeras och startas medan ett program med öppen källkod skulle vara betydligt svårare. Det var mot förmodan tvärtom. Veckor fylldes av installationsarbete med servern medan oSIP fungerade direkt.

I detta kapitlet går vi igenom hur ovanstående applikationer, samt de applikationer som SIPd är beroende av, fungerar och används.

3.1 oSIP

Den delen av SIP-arkitekturen som laborationen kretsar kring är användarapplikationerna i ändpunkterna av en kommunikation. Laborationen kommer att bygga på en SIP-stack som heter oSIP. Källkoden finns att ladda hem från http://osip.atosc.org, och är licencierat enligt GNU Public Licence, (GPL). Med den version av oSIP (0.7.9) vi använder medföljer ett exempelprogram att använda för enkla SIP-sessioner. I senare versioner finns inte exempelprogrammet med. Vi har inte utfört några tester för att utröna om senare versioner är kompatibla med exempelprogrammet.

oSIP erbjuder en bas för att bygga proxy-, registrar- och redirectservrar, men framför allt applikationer i ändstationerna. En transaktionshanterare, bestående av tillståndsmaskiner, ser till att rätt aktivitet sker i olika situationer. SIP-adresser och meddelanden kontrolleras av en parser. oSIP består av funktioner som används när en SIP-applikation eller ett multimediaprogram med stöd för SIP byggs. Programmet hanterar inte sockets och portar utan det är upp till den som utvecklar applikationen att hantera. Detta innebär att valet av transportprotokoll är fritt, men UDP är det naturligaste eftersom SIP erbjuder tillförlitlighet.

De meddelanden som kan köras med exempelprogrammet är INVITE- och BYE-anrop mellan två oSIP-installationer eller via proxyservrar. Registrering görs genom REGISTER-anrop till en registrar-server. Sockethanteringen är inbyggd i exempelprogrammet och användaruppgifter lagras i en konfigurationsfil som anges när exempelprogrammet startas.

```
# this file is used to store user config values
#
# empty lines are not accepted
# comments starts with #
#
# !!!!!!! extra SPACES may lead to unknown behavior!!!!!!
#
# duplicates fields are discarded...
# no other restrictions
#
#
#
displayname=Aymeric
username=jack
localip=127.0.0.1
contact=<sip:jack@127.0.0.1>
from=<sip:jack@127.0.0.1>;tag=12gtr38erf7-1dfw
#
localport=5060
#
# MUST BE COMMENTED IF YOU DON’t WANT TO USE THE PROXY
#
#sipproxy=sip:127.0.0.1:5080
#
sipregistrar=sip:127.0.0.1:5080
#
#SDP config
networktype=IN
addr_type=IP4
#
```

Figur 7: Exempel på konfigurationsfil, siprc i conf-biblioteket.

Figur 7 visar ett exempel på en konfigurationsfil som fungerar för en uppkoppling mellan två UA på samma enhet. De rader som inleds med ”#” är bortkommenterade och används inte av programmet. Övriga rader är användaruppgifter.

- `displayname` kan ingå i To-, From- och Contactfälten i ett meddelande men är inte en del av SIP-adressen.

- `username` används där hela SIP-adressen inte behövs t.ex. i SDP-protokollet.
- `localip` är användarens egen IP-adress.
- `contact` anger SIP-adressen till den enhet där användaren vill ta emot uppkopplingen. SIP-adressen består av användarnamn och IP-adress avskiljt med ett @-tecken, t.ex. `carl@193.10.221.187`.
- `from` innehåller normalt en SIP-adress bestående av användarnamn och ett domännamn där användaren har sin tillhörighet tex. `carl@cse.kau.se`.
- `localport` är porten användaren lyssnar av för att upptäcka SIP-anrop.
- `sipproxy` är IP-adressen till den proxyserver som uppkopplingsbegäran skickas till. Om raden är bortkommenterad måste startkommandot innehålla mottagarens IP-adress.
- `sipregistrar` är domänen där registrarservern finns.
- `SDP config` listar parametrar för en multimediasession.

Figur 8 visar ett uppstartskommando i exempelprogrammet där konfigurationsfilen (inklusive sökvägen) anges efter en f-flagga. Övriga argument i startkommandot anger vilket beteende, mod (-m) och debug-nivå (-d) programmet skall använda samt SIP-adressen (-t) till mottagaren under sessionen. En makefile finns med och vid kompilering skapas en fil med namnet ua. Mod 0 ger möjligheter att skicka INVITE-, REGISTER- och BYE-anrop. Debugnivå 5 ger skärmutskrifter på alla requests och responses.

```
$> ./example_mt/ua -m 0 -f conf/siprc -d 5 -t "<sip:jack@127.0.0.1>"
```

Figur 8: Ett giltigt startkommando för exempelprogrammet i oSIP 0.7.9.

För att få en session till stånd måste två program vara igång och program som skall skicka requests, måste vara startade i mod 0. Båda sidor kan var för sig skicka requests. Om ett program endast skall ta emot och skicka responser kan mod 3 användas istället.

Anges en proxyserver i konfigurationsfilen kommer alla meddelanden att skickas dit. Då måste mottagarsidan vara registrerad hos en registrarserver.

3.2 SIPd

Den proxyserver vi har tillgång till heter SIPd och ingår i multimediaarkitekturen CINEMA, som produceras av SIPCOMM. Den fungerar också som en registrar- och redirectserver vilket innebär att de tjänster som behövs för att bygga en infrastruktur för SIP-signalering finns.

3.2.1 Applikationen

Servern, vilken fungerar som kärnan i systemet, är en komplett implementation enligt RFC 2543 och handhar alla meddelanden om användare och anrop. SIPd använder DNS-systemet för att ta reda på mottagaradresser utanför den egna domänen. Det finns stöd för både IPv4 och IPv6 samt för konvertering mellan dem, liksom för transportprotokollen TCP, UDP och Transport Layer Security (TLS) [10]. Möjligheter att autentisera sig ingår samt förmedling av uppkopplingar till vanliga telefonnummer. Servern lyssnar på port 5060 och beroende på vilket meddelande som kommer vidareförmedlas dessa till rätt enhet eller adress.

Vid REGISTER-anrop förmedlas dessa till registrarservern och lagras i en databas upp till 24 timmar. Ett meddelande återsänds till användaren med information om registreringen lyckats eller inte. När ett INVITE-anrop kommer till servern, kontrolleras om mottagaren finns registrerad i en egen databas. Finns mottagaren registrerad sänds anropet vidare till rätt enhet. Om mottagaren inte finns registrerad, sänds INVITE-anropet vidare till en nod närmare mottagaren. I SIP-headerns Via-fält läggs den egna adressen till och ett Trying-meddelande sänds till avsändaren. Proxyservern vidareförmedlar också Ringing-meddelanden och OK-meddelanden från mottagaren till den som initierar sessionen.

SIPd, vilken inhandlats av avdelningen för datavetenskap, hämtades hem via Internet från SIPCOMMs hemsida. Den version, Cinema 1.20-1, vi använder i vår installation är kompletterad med en specialkompilerad uppstartsfil för den Linuxkärna som universitetet använder i sin konfiguration. I programpaketet finns en fil, install.html, som beskriver hur installationen går till. Installationen görs med hjälp av ett tcl-script där applikationen konfigureras med domännamn, administratör, användarnamn, lösenord, proxyserverns IP-adress mm.

Före installationen måste MySQL, Apache och tcl vara installerade. Dessa presenteras mer ingående i efterföljande avsnitt. SIPd startas genom att vid prompten ange startkommando enligt Figur 9. `sipd` är namnet på det exekverbara programmet, `–X`-flaggan anger att programmet startas i non-daemon mode , `-d`-flaggan anger att debugging-information skrivs ut på skärmen och `–f`-flaggan anger att en sökväg till konfigurationsfilen, i vilken en URL till SQL- databasen skall anges.

```
./sipd –X –d –f /opt/cinema/web/cinema_db.conf
```

Figur 9: Exempel på ett startkommando för SIPd.

3.2.2 MySQL

MySQL [9] är ett databassystem utvecklat av det svenska företaget MySQL AB. Applikationen är ett sk. freeware med öppen källkod vilket innebär att det är tillgängligt för alla enligt GPL. Programmet är en relationsdatabas vilket lagrar data som ihoplänkade tabeller. Tabellerna hanteras med hjälp av databasspråket Structured Query Language (SQL). SQL är ett standardspråk för databaser.

Den version av MySQL vi använder är 3.23.43. Vår implementation av SIPd innehåller 33 tabeller, se Figur 10, med för proxyservern, registrarservern och redirectservern nödvändiga data.

Figur 10: MySQL- tabellerna som används av SIPd.

De mest använda tabellerna är *person* och *contacts* vilka innehåller data om användare som är registrerade och var de kan nås.

```
mysql> select * from contacts;
+-----------------+-----------------------------+---------------------+------+--------+---------------+--------------+-------------+---------
+
| user            | contact                     | expires             | q    | action | last_modified  | display_name | sip_methods | deleted
|
+-----------------+-----------------------------+---------------------+------+--------+---------------+--------------+-------------+---------
+
| hane@cse.kau.se | sip:hane@carl-ws3.cse.kau.se | 2002-04-16 13:53:50 | 1.00 | Proxy  | 20020416145408 | NULL         | any         | false
|
| bun@cse.kau.se  | sip:bun@carl-ws3.cse.kau.se  | 2002-04-15 13:59:46 | 1.00 | Proxy  | 20020415150411 | NULL         | any         | true
|
| hane@cse.kau.se | sip:hane@tdspdc01.cse.kau.se | 2002-04-15 13:40:43 | 1.00 | Proxy  | 20020415144923 | NULL         | any         | true
|
| an@cse.kau.se   | sip:an@carl-ws3.cse.kau.se   | 2002-04-10 09:51:17 | 1.00 | Proxy  | 20020410105215 | NULL         | any         | false
|
| bun@cse.kau.se  | sip:bun@tdspdc01.cse.kau.se  | 2002-04-16 13:55:34 | 1.00 | Proxy  | 20020416145608 | NULL         | any         | false
|
+-----------------+-----------------------------+---------------------+------+--------+---------------+--------------+-------------+---------
+
5 rows in set (0.50 sec)
```

Figur 11: Tabellen contacts i databasen sip.

Tabellen contacts, se Figur 11, innehåller data som lagras när ett REGISTER-meddelande görs. Fälten i tabellen innehåller:

- `user` innehåller SIP-adressen.
- `contact` är adressen till den fysiska enhet personen vill att SIP-anrop skall vidareförmedlas till.
- `expires` är datum och tid som registreringen är giltig.
- `q` anger prioriteringsordningen av adresser i contact-kolumnen om fler registreringar finns för samma SIP-adress.
- `action` anger vilken tjänst som skall tillhandahållas, proxy eller redirect.
- `last_modified` är när användaren registrerades senaste gången.
- `display_name` kan visas istället för SIP-adressen.
- `sip_methods` anger vilka SIP-metoder som stöds.
- `deleted` anger om tabellraden är raderad.

3.2.3 Apache

Apache HTTP Server [1] är en webbserver som lyssnar efter anrop över Internet eller ett lokalt nätverk. När en uppkopplingsbegäran kommer förmedlar servern den begäran som finns i http-meddelandet till rätt katalog där ett skript anger vad som skall göras. Servern förmedlar sedan detta vidare till klientens browser.

Även Apache är en applikation med öppen källkod. Den version vi använder i vår implementation är Apache-1.3.14-1.

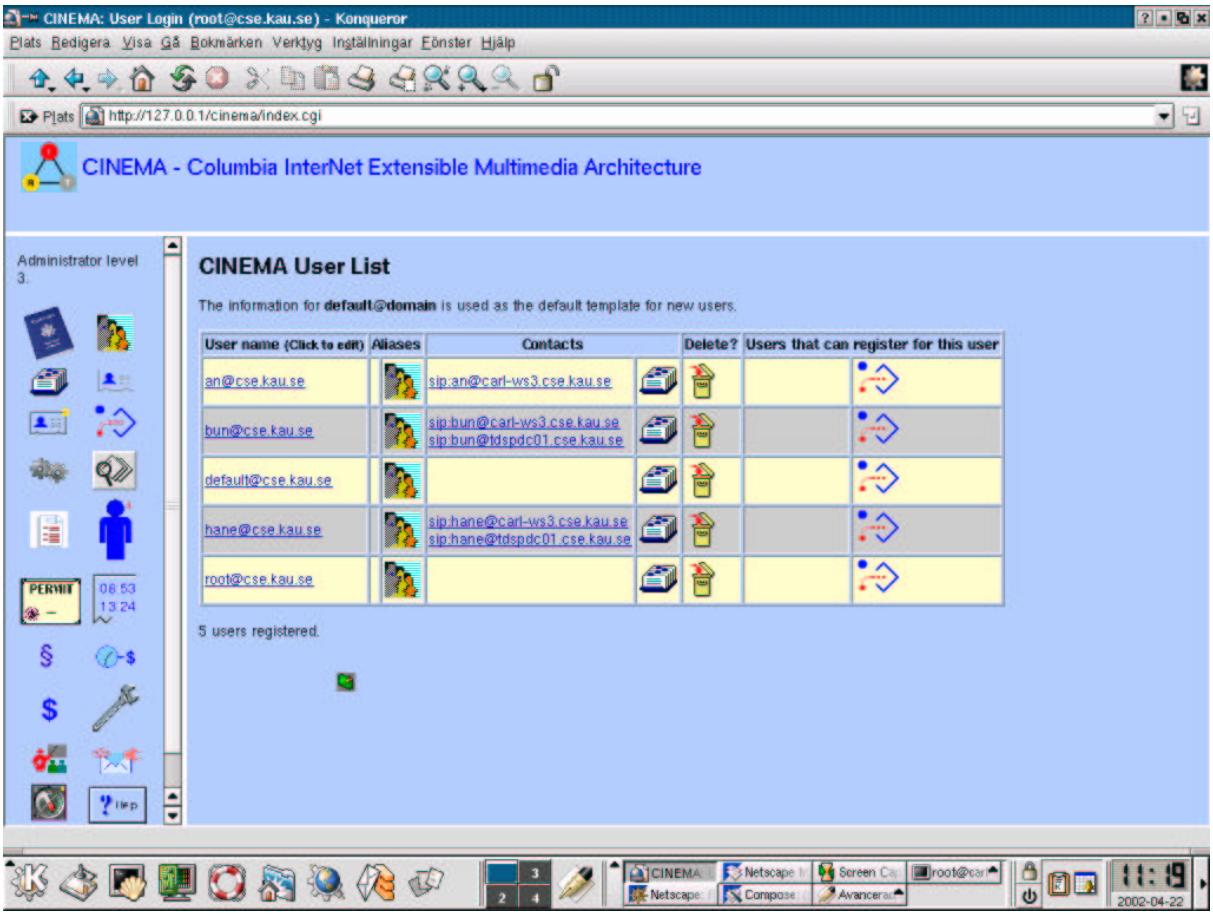
3.2.4 Tool Command Language

Tool Command Language (tcl) [11] är ett tolkande skriptspråk som de flesta av Cinemas webbdokument är skrivna i. För att kunna installera SIPd och använda oss av webbgränssnittet var vi tvungna att installera en tolk för detta språk. Versionen vi använder är tcl 8.3.

3.2.5 Cinema

Cinema är samlingsnamnet på de av SIPCOMM designade applikationerna och innehåller ett webbgränssnitt bestående av en mängd skript för administrering av SIP-databasen över ett nätverk.

Cinema som vid en uppkopplingsbegäran anropas av Apacheservern visar hos klienten en inloggningssida där man kan logga in som administratör eller användare. Efter inloggnings-fasen visas ett nytt fönster med en mängd grafiska verktyg för administrering av systemet. Härifrån kan man konfigurera SIPd, skapa nya och redigera användare. Figur 12 visar gränsnittet för Cinema och den aktuella bilden avser en lista över de användare som lagrats i databasen. I Contacts-kolumnen syns information om var dessa kan vara registrerade. Om man klickar sig vidare i fältet visas den information som beskrivs i Figur 11.



Figur 12: Cinemas gränssnitt, användarlista.

4 Laboration

För att bestämma hur laborationerna slutgiltigt skulle utformas visade vi vårt arbete för institutionens anställda samt för speciellt inbjudna från Ericsson Infotech AB. Vi gick igenom arkitekturen, protokollets uppbyggnad samt visade signaleringens vägar vid en uppkoppling mellan två UA via en proxyserver. Nätet avlyssnades med Ethereal och möjligheterna med programmet förevisades. En mediaöverföring från en videokamera till en bärbar dator via det trådlösa nätet visades också.

Efter demonstrationen hölls en diskussion om hur laborationerna bör utformas. Tillsammans med ansvariga lärare för kursen Datakommunikation II togs beslut om hur denna skall utformas. Laborationen skall bestå av två delar, en signaleringsdel och en multimedia-del.

Media Gateway-delen avgränsas från den ursprungliga kravspecifikationen p.g.a. att lämplig applikation saknas. Funktionen med differentierade signalerings- och datavägar kommer i ställer att visas med hjälp av proxyservern.

Kapitlet fortsätter med en presentation av de applikationer som kommer att samverka med de tidigare beskrivna SIP-applikationerna. Avsnitt 4.2 och 4.3 beskriver själva laborationen.

4.1 Applikationer som används i laborationen

För att genomföra laborationen krävs ytterligare program och produkter förutom de SIP-specifika applikationerna som beskrevs i kapitel 3. Ethereal [3] är det program som skall lyssna av nätverket och visa vägarna genom nätet. PC-kameran och mediaspelaren vi använder i multimediadelen presenteras samt den anpassning av kod som gjorts i oSIPs exempelprogram visas i kapitlet.

4.1.1 Ethereal

Ethereal är ett program som avlyssnar den trafik som passerar i nätet. Det är ett så kallat sniffer-program. I Ethereal kan man se från vilken avsändare och till vilken mottagare paket går samt paketets innehåll ända ner på bitnivå. Detta innebär bland annat att man kan se vilket transportprotokoll och vilka IP-adresser som används. Det finns många möjligheter att välja vilken trafik som skall fångas upp.

I laborationen skall all trafik vilken inte innehåller ett SIP-meddelande filtreras bort. Granskning skall göras av avsändarens och mottagarens IP-adresser, vilket transportprotokoll som används samt vilka olika SIP-meddelanden som sänds.

I multimediadelen tillkommer dessutom att granska vilken väg de olika datapaketen går. Filtret ställs in så att det släpper igenom trafik från användarens och mottagarens IP-adresser.

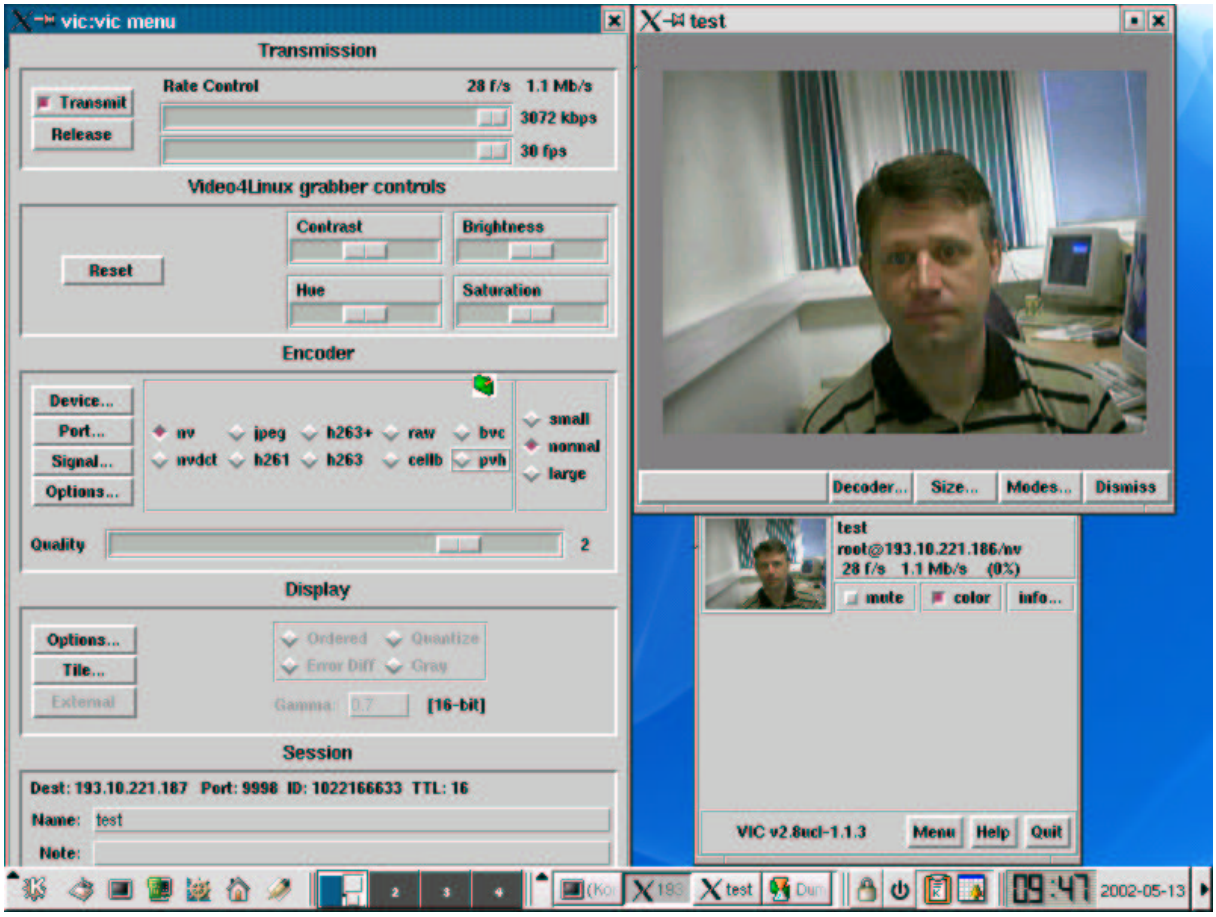
4.1.2 PC-kamera

Den kamera som används är Philips ToUcam PRO PCVC 740K vilken används via USB-porten. Kameran har en högkänslig sensor och ger en flimmerfri bild med upp till 60 bildrutor per sekund. En inbyggd mikrofon finns som ej kommer att användas i laborationen.

4.1.3 Video Conferencing Tool

Video Conferencing Tool (VIC) [13] är en mediaspelare för både unicast- och multicast-sessioner via en punkt-till-punkt-förbindelse respektive en multicastadress. Programmet ger många möjligheter att påverka överföringskvalitén med hjälp av olika bild- och ljud-

kodningar. Det går även att avläsa överföringshastighet, bps, och antal överförda bilder per sekund, fps, vilka kommer att användas i laborationen.



Figur 13: VIC användargränssnitt.

VIC startas från ett terminalfönster genom att vid prompten ange kommandot: `vic [option] adress/port.` Vid unicast skall IP-adressen samt ett portnummer till den mottagande enheten anges hos båda deltagande parterna. När programmet startas visas ett huvudfönster, nederst till höger i Figur 13, där miniatyrbilder på multimediasessionerna samt relaterad data finns. Via huvudfönstret kan ett grafiskt menyfönster, till vänster i Figur 13, startas och genom att klicka på miniatyrerna öppnas ett fönster, överst till höger i Figur 13, för vald session där bilden visas i inställd storlek och kvalité.

Vid videoöverföring går det att använda ett antal olika kodningar. Dessa ställs in genom att klicka på respektive knapp i rutan Encoder i menyfönstret. Lämpligast används kodningen network video tool (nv) [6] och båda parametrarna för hastighetskontroll (Rate Control) ställs in på maximalt värde. Den reella överföringshastigheten visas med digitala parametrar. För att

starta överföringen klickar man på Transmit-knappen. Menyn avser inställningar för den sändande enheten.

4.1.4 User Agent för laboration

För att göra en UA till laborationsmiljön har vi byggt ovanpå oSIP-stacken och använder till stora delar det exempelprogram som ingick i oSIP 0.7.9. Vi har byggt in stöd för att starta en VIC-applikation, ändrat tiden en användare är registrerad efter ett REGISTER-anrop till en registrar-server och ändrat signaleringsvägen för ACK-meddelandet.

För att starta VIC behövs tillgång till en IP-adress och ett portnummer. Vi har modifierat initieringen så att det är möjligt att ange ett portnummer för VIC-applikationen i startkommandot efter en `–v`-flagga.

Vi har lagt till ett menyval, ”`v`” i den meny som visas när en UA startas i mod 0. Den utför samma procedur som ett INVITE-anrop med tillägget att VIC blir tilldelad ett fast portnummer om inte en port angivits i startkommandot.

Om en port har angivits, antingen genom startkommandot eller om ”v”-kommandot ges, kommer VIC att startas då ett ACK-meddelande skickas. ACK-meddelandet skickas till mottagaren som bekräftelse på att OK-meddelandet i en INVITE-session erhållits. OK-meddelandet är mottagarens svar på avsändarens INVITE-anrop. VIC startas när den kod vi lagt till i filen `example_agentlogic.c`, se bilaga B.2, exekveras. VIC-applikationen startas och tilldelas loopback-adressen 127.0.0.1 samt den port som tilldelats applikationen.

Då ett portnummer är angivet i startkommandot för en uppringd UA startas en VIC-applikation när ett INVITE-anrop anländer, även den med loopback-adressen och den angivna porten.

Under laborationen är det bra om en registrering inte ligger kvar för länge eftersom det kan störa funktionalitet under laborationen. Det finns en risk att registreringar från föregående laborationsgrupper ligger kvar eftersom SIP-adresserna inte är unika för laborationsgruppen utan återanvänds. Tiden för hur länge en registrering är giltig är satt till 20 minuter. När en registrering hos en registrar-server görs med ett REGISTER-meddelande skickas tiden för hur länge registreringen skall gälla med i `Contact`-fältet som en parameter efter ett semikolon, t ex `;expires=3600`, eller som ett eget fält i headern, `Expires:3600`. Tiden mäts i sekunder och defaultvärde i exempelprogrammet är 3600 sekunder. Defaultvärdet initieras i `main`-funktionen som finns i filen `app.c`, se bilaga B.1. Den UA som studenterna skall kommunicera med i första delen av laborationen måste vara ständigt registrerad. Proxyservern

vi använder, SIPd, godkänner högst 24 timmars registrering. Det innebär att för varje laborationsdag måste laborationshandledaren göra en ny registrering.

Signaleringsvägen för ACK-meddelandet kan gå samma väg, via en eller flera proxyservrar, som tidigare meddelanden i en INVITE-session eller direkt till den UA som blivit uppringd. Exempelprogrammet sänder ACK-meddelandet via proxyservrar. För att åskådliggöra att ACK-meddelandet och mediasessionen kan ta andra vägar genom ett nätverk än de intierande meddelandena har vi ändrat koden så att ACK-meddelandet går direkt till den mottagande UAn. Det är bara ACK-meddelandet som ändras, ytterligare meddelande i sessionen, t ex en ny INVITE eller en BYE, kommer att gå via proxy.

Ändringen av programmet är gjord så att adressen i OK-meddelandets `contact`-fält läses av och används som mottagaradress i ACK-meddelandet. Justeringen är gjord i funktionen `makeack()` som ingår i filen `msg_ack.c`, där meddelandets startfält, d.v.s. första raden i SIP-headern modifierats. Även funktionen `udp_send_request()` i filen `udp_send.c` har justerats så att när SIP-metoden är en ACK, väljer sockethanteringen mottagaradressen istället för adressen till proxyservern. Ändringarna i koden framgår av bilagorna B.3 och B.4.

4.2 Signalering

En förutsättning för laborationens genomförande är ingående studier av RFC 2543 [7]. Inledningsvis görs en uppkoppling mellan två UA utan samverkan med mellanliggande SIP-servrar. I det här skedet kopplas ingen multimediasession upp utan uppgiften är att studera signaleringsvägarna och vilka meddelanden som skickas. En given mottagare finns uppstartad i laborationsmiljön och studenterna använder oSIP med det bifogade exempelprogrammet vilket innefattar en konfigurationsfil som är modifierad bl.a. med enhetens IP-adress. I startkommandot anges efter `-t`-flaggan en från laborationshandledaren erhållen SIP-adress innehållande den exakta IP-adressen. De meddelanden som skall skickas är INVITE och därefter BYE.

Granskningen av signaleringsvägarna, meddelanden och IP-adresser görs med hjälp av Ethereal och uppgiften redovisas med ett sekvensdiagram liknande Figur 4 samt svar på följande frågor:

- Vilka IP-adresser har du respektive mottagaren?
- Vilka olika typer av meddelanden (messages) utbyts?
- Vilka olika typer av metoder (methods) används vid uppkopplingen?
- Vilket transportprotokoll använder sig SIP av i uppkopplingen och varför?

Nästa del av laborationen innebär att ta reda på en mottagares IP-adress med hjälp av en SIP-adress. Först måste en registrering hos en registrarserver göras. Detta görs genom ett SIP-meddelande innehållande ett REGISTER-anrop. För att en registrering skall vara möjlig måste studenterna ändra konfigurationsfilens innehåll. De fält i filen som skall ändras framgår av Tabell 1 där andra kolumnen visar fälten i ursprungligt skick och den tredje visar ett exempel på konfiguration för uppkoppling via proxyserver. IP-adressen i `contact`-fältet måste vara på numerisk form enligt vårt lösningsförslag då stöd för DNS-sökning av adresser i namnform saknas i för laborationen relevanta oSIP-funktioner.

Fältnamn	**Ursprungligt innehåll**	**Lösningsförslag**
`username=`	`jack`	`Sven`
`contact=`	`<sip:jack@127.0.0.1>`	`<sip:sven@193.10.221.186>`
`from=`	`<sip:jack@127.0.0.1>; tag=12gtr38erf7-1dfw`	`<sip:sven@cse.kau.se>; tag=12gtr38erf7-1dfw`
`#sipproxy=`	`sip:193.10.221.187:5060`	*`endast borttagning av kommentartecken (#) i fältets början`*

Tabell 1: Lösningsförslag till konfigurationsfil.

Konfigurationsfilen måste ändras innan oSIP startas. Detta beskrevs i avsnitt 3.1. Därefter väljs ”`r`” i oSIPs meny för att göra en registrering vilken bekräftas med ett OK-meddelande. Den användare som fungerar som mottagare i en session via proxyserver måste vara registrerad och dess UA måste vara startad och lyssna efter anrop.

En INVITE görs från menyn genom att välja ”`i`”, då används mottagarens SIP-adress som angivits efter `-t`-flaggan i startkommandot. Redovisningen skall göras med ett sekvensdiagram liknande det i Figur 5 med samtliga noders IP-adresser angivna samt redovisa den ändrade konfigurationsfilen.

Den sista signaleringsdelen av laborationen består av att starta en multimediasession med hjälp av SIP-signalering. Om ”`v`” väljs i oSIP-menyn kommer en INVITE-begäran att skickas och när OK-meddelandet anländer kommer en lokal VIC-applikation att startas. Studenternas uppgift är att ändra koden så att mediaströmmen från VIC sänds till mottagaren angiven i startkommandot.

Ändringarna skall göras i funktionen `respondto200forinvite()` vilken finns i filen `example_agentlogic.c`. Kodblocket som skall ändras lyder:

```
if(vic_port != NULL)
{
  if(fork() == 0)
    execlp("vic","vic","127.0.0.1/6668",NULL);
  free(vic_port);
  vic_port=NULL;
}
```

Structen `sip_t`, se bilaga A, innehåller uppgifter som ingår i headern i ett SIP-meddelande, bl a vad som står i `contact`-fältet. I ett response-meddelande, i det här fallet OK, finns information om den uppringda partens IP-adress samt vilken port som skall användas vid fortsatt signalering. Det är den uppringda partens IP-adress som skall användas i VIC-applikationens startkommando. Ett sätt att ta reda på adressen är att använda oSIP-funktionen `msg_getcontact()` som lagrar alla `contact`-fält i en array. Dess innehåll är structar av typen `contact_t` vilken i sin tur innehåller en annan struct, `url_t`, där IP-adressen finns i form av en sträng. Alla inblandade structar finns beskrivna i laborationsspecifikationen, se bilaga A, liksom hela filen `example_agentlogic.c` i bilaga B.2.

Då tillgång till källkod finns kan man lösa uppgiften på olika sätt. Vårt förslag till lösning använder funktionen `msg_getcontact()`.

```
if(vic_port != NULL)
{
  char* arg = (char*)malloc(25);
  contact_t **contact;
  contact = (contact_t**)malloc(sizeof(contact));
  contact_init(&contact[0]);
  msg_getcontact(response,0,contact);
  strcpy(arg,contact[0]->url->host);
  strcat(arg,"/");
  strcat(arg,vic_port);
  if(fork() == 0)
    execlp("vic","vic",arg,NULL);
  free(vic_port);
  vic_port=NULL;
}
```

Studenterna har genomfört uppgiften när de lyckats göra en multimediaöverföring mellan två datorer över ett nätverk. Deluppgiften redovisas genom att för laborationshandledare demonstrera uppkopplingen samt i dokumentationen bifoga modifierad kod.

4.3 Multimedia

Multimediadelen av laborationen visar hur överföringskvalitén påverkas med olika kodningar och överföringsmediets karaktäristik. För att se hur olika bildkodning påverkar resultatet görs en uppkoppling mellan två stationära enheter med hjälp av föregående laborationsdel.

När sessionen är uppkopplad varieras kodningen mellan nv, h263+ [4] och cellbytestream (cellb) [8]. Enligt våra tester av de olika kodningarna ger nv den bästa bildkvalitén och den snabbaste överföringshastigheten, h263+ ger en relativt bra bildkvalité men den överförda bildhastigheten blev endast 3-4 bilder per sekund. Cellb ger en godkänd bildkvalité men klart synbara s.k. dropouts (bildförändringar i form av saknade eller förändrade bildpunkter) förekommer. Dessa egenskaper kan förändras beroende av VIC-applikationens implementation och konfiguration vilket kan ge en missvisande bild av kodningarnas effektivitet.

Den sista delen av laborationen skall utföras mellan en stationär dator med en PC-kamera installerad och en uppkoppling via ett trådlöst nätverk till en bärbar PC. På båda enheterna skall VIC användas. Syftet med laborationsdelen är att visa hur en överföring förändras beroende av yttre faktorer som påverkar radiosignalen i det trådlösa nätet. Då flera kodningar varierar överföringshastighet efter bildinnehållets förändringar måste kameran vara riktad på ett föremål som rör sig kontinuerligt, helst så regelbundet som möjligt. Detta innebär att en student i laborationsgruppen bör vara placerad framför kameran och röra sig.

Kodningen som skall användas vid överföringen bör vara den bästa tillgängliga. Bildkvalité, överföringshastighet och antalet överförda bilder per sekund skall läsas av på fyra geografiskt olika platser. För att få likartade värden mellan de olika laborationsgrupperna måste mätningarna göras på av handledare noggrant angivna platser. Vi föreslår nedanstående platser, där vi erhållit mätvärden enligt Tabell 2:

1. Framför handikappskylten norr om 5-huset.
2. I 12-huset på trappavsats mellan plan 2 och 3.
3. I korridoren framför dörr 12C314.
4. Inne i sal 12A326.

Redovisningen av laborationsdelen sker genom att beskriva bildkvalitéer, sammanställa mätresultat samt kommentera resultaten.

Plats	**Överförings-hastighet i kbps**	**Överförda bilder per sekund**	**Bildkvalité**
1	2 400	23-24	Godkänd
2	3 000	28-30	Mycket bra
3	800	4	Föremålen i bilden rör sig ryckigt
4	0	0	Bilden fryst

Tabell 2: Testresultat vid multimediaöverföring med bärbar dator.

VICs funktion är sådan att när bildkvalitén går under en viss nivå fryses bilden på en godtagbar nivå. Detta innebär att det är svårt att jämföra olika bildkvalitéer på det sätt vi antog när utvecklingsarbetet började. Även Universitetets trådlösa nätverk är känsligt vilket gör att man på samma mätplats vid olika tidpunkter kan få skiftande mätresultat. En förutsättning för att på ett realistiskt sätt kunna genomföra laborationen måste VICs funktion stabiliseras samt det trådlösa nätets egenskaper förbättras.

5 Erfarenheter och utvecklingsmöjligheter

SIP är under kontinuerlig utveckling och under den tid vi arbetat med examensarbetet har ett antal nya Internet-drafts på RFC 2543 publicerats. Den dokumentation om SIP som vi började vårt examensarbete med är draft RFC 2543 bis-05 och i februari 2002 kom draft RFC 2543 bis-09. Den version vi har använt mest och refererar till i dokumentet är draft RFC 2543 bis-06. Detta betyder att SIPs användningsområde kan växa och nya möjligheter tillkommer. För oss har detta inneburit att flera av de produkter vi använt oss av varit på utvecklingsstadiet och därför skapat problem vid installationer. Efter installation av SIPd, som är en färdigkompilerad produkt, kunde den inte samverka med den Linuxkärna som vi installerat i vår laborationsmiljö. Installationsförsök med två olika versioner av SIPd gjordes, den första både manuellt och med färdiga skripts, utan lyckade resultat. Installations- och konfigurationsarbetet pågick i ett antal veckor med systemadministratörer Anders Hedberg och Stefan Alfredsson involverade innan en specialversion av den exekverbara filen med stöd för vår Linuxkärna, kompilerad av SIPCOMM, löste vårt problem.

5.1 Installation och vidare design av laborationen

Ett krav på laborationen är att den skall genomföras utan speciella privilegier såsom root eller administratör. För att uppfylla kravet måste installationen av oSIP på laboratoriets datorer göras så att exempelprogrammet kan kompileras från studenternas hemkataloger. För att åstadkomma detta måste en modifiering av befintlig makefile göras.

En UA skall installeras och konfigureras så att den är aktiv och registrerad i SIPd-servern under samtliga laborationstillfällen. Den används som mottagare i laborationsdel a) och b).

oSIP i dess nuvarande form är inte helt stabil efter omstyrning av ACK-signalen från signalväg via proxyserver till att få den att gå direkt till den uppringda enheten. Det fel som uppkommer består av att vid upprepade signaleringsmeddelanden avbryts programmet och meddelande om segmenteringfel erhålls. Vi antar att felet uppkommer p.g.a. att programmet återlämnar resurser som behövs i senare faser.

Multimedialaborationen, vilken idag inte går att genomföra tillfredsställande p.g.a de problem som finns inbyggda i VIC, kan utvecklas på olika sätt. Ett arbete med modifiering av VIC pågår, om det lyckas innebär det att laborationen går att genomföra på av oss föreslaget

sätt. Misslyckas modifieringen måste överväganden om byte av mediaapplikation göras, alternativt ändra laborationen så att den blir anpassad efter VICs möjligheter.

5.2 Framtida möjligheter

oSIP-plattformen har också lanserats i nyare versioner efter den version vi valt att bygga plattformen på, vilket kan medföra att funktionalitet tillkommit som kan vara intressant i vidare utvecklingsarbete av ett kommunikationslaboratorium. Ett utvecklingsprojekt för att ta fram en SIP-server baserad på oSIP har initierats av samma utvecklingsgrupp som utvecklar oSIP, atosc.org, också den med öppen källkod vilket öppnar möjligheter för egna server-lösningar.

Fortsatt utvecklingsarbete av en UA byggd på oSIP kan innebära

- ändring av debuggnings-utskrifterna i exempelprogrammet så att dubbel presentation av meddelande ej görs.
- en utökning av interaktiviteten, t ex att en mottagare kan anges under programmets exekvering i stället för i kommandoraden.
- att stöd för IP-adresser på namnform byggs in i sockethanteringen.
- att loggningen, som för varje exekvering sparas i en ny fil, effektiviseras.

6 Slutsatser

Vi har under våren utarbetat ett förslag till laboration enligt den kravspecifikation som universitet angivit. Vår slutsats är att den väl speglar det angivna syftet och följdaktligen är genomförbar.

I laborationens signaleringsdel kan den User Agent, som bygger på oSIP-plattformen, förfinas efter de förslag vi presenterat i avsnitt 5.2. Multimediaapplikationen, VIC, vilken vi varit hänvisade att använda, uppfyller i nuläget inte de på förhand angivna kraven fullt ut. Möjligen kan den modifieras så att den blir användbar, om inte bör en annan produkt väljas eller en anpassning av laborationen efter VICs prestanda göras.

Referenser

[1] Apache Digital. http://www.apache.com, 2002-02-14.

[2] Daniel Collins. *Carrier Grade Voice Over IP*. McGraw-Hill, 2001.

[3] Ethereal. http://www.ethereal.com/, april 2002.

[4] Fred Halsall. *Multimedia Communications*. Addison-Wesley, 2001.

[5] Henning Schulzrinne och Jonathan Rosenberg. *The Session Initiation Protocol: Internet-Centric Signaling.* IEEE Communication Magazine, oktober 2000.

[6] Jay Schuster. *nv, version 3.3*, The Physician's Computer Company, april 1997, http://www.pcc.com/~jay/src/networking/nv/ORIGINALS/.

[7] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg. *SIP: Session Initiation Protocol*, RFC 2543, mars 1999.

[8] M.Speer, D. Hoffman. *RTP Payload Format of Sun's CellB Video Encoding*, RFC 2029, oktober 1996.

[9] MYSQL AB. http://www.mysql.com/products/what_is_mysql.html, 2002-02-03.

[10] P. Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION, RFC 1035, november 1987.

[11] Tcl Developer X-change. http://www.tcl.tk/software/tcltk, 2002-02-14.

[12] T. Dierks. *The TLS Protocol*, RFC 2246, januari 1999.

[13] *User Guide for VIC v2.8, Version 1*, Computer Science Department, University College London, september 1998.

[14] Wenyu Jiang, Jonathan Lennox, Henning Schulzrinne and Kundan Singh. *Toward Junking the PBX: Deploying IP Thelephony*, Department of Computer Sience, Columbia University, 2001.

A Specifikation för multimedialaboration

Laborationen består av två delar. Den första delen innebär att ta reda på och att förstå signaleringsvägarna vid uppkopplingar med SIP. Den andra består av en videosession mellan två stationära datorer och en session mellan en stationär och en bärbar dator. Information om hur SIP-arkitekturen fungerar finns i RFC 2543 på http://www.ietf.org/rfc/rfc2543. För att få en överblick över signaleringen skall programmet Ethereal användas. Ethereal avlyssnar och analyserar trafik på nätet. I Ethereal filtreras alla paket utom de som innehåller SIP-meddelanden bort. oSIP implementeras i hemkatalogen på de datorer som skall användas efter handledarens instruktioner.

Signaleringsdelen

Uppgift 1

En direktuppkoppling mellan två User Agent (UA) skall göras med hjälp av oSIP och det exempelprogram som finns. En UA dit uppkopplingen skall göras finns aktiv. Adressen erhålls av handledaren.

- Starta Ethereal och filtrera bort alla paket som inte innehåller SIP-meddelanden.
- oSIP startas genom att stå i katalogen `/…/example_mt` och vid promten ange:
  `./ua -m 0 -f sip_conf -d 5 -t sip:sven@111.111.111.111`
- Skicka ett INVITE-anrop till den givna mottagaren.
- Avsluta med ett BYE-anrop.
- Granska med Ethereal vilka meddelanden som sänts och från vem dessa gått. Öppna paketen och granska hur dessa ser ut.

I redovisningen skall nedanstående ingå:

- Beskrivning av upp- och nedkopplingen i ett sekvensdiagram.
- Ange de IP-adresser som du respektive mottagaren har.

- Beskriv vilka olika typer av meddelanden (messages) som utbyts och vilka olika typer av metoder (methods) som används vid uppkopplingen.
- Ange vilket transportprotokoll SIP använder sig av i uppkopplingen och varför.

Uppgift 2

Gör ett REGISTER-anrop till servern och därefter ett INVITE-anrop till mottagaren i uppgift 1, nu med en SIP-adress som delas ut av handledaren. För att göra detta måste en konfigureringsfil (`sip_myconf`) skrivas. Använd filen `sip_conf` som mall som finns i katalogen example_mt. Adressen till proxyservern erhålls av handledaren.

- Starta Ethereal.
- Starta oSIP med SIP-adressen till en given mottagare.
  `./ua -m 0 -f sip_conf -d 5 -t sip:sven@111.111.111.111`
- Skicka en registrering till servern.
- Gör en INVITE-begäran till en given mottagare.
- Kontrollera vilken väg de olika meddelandena går.

I redovisningen skall ett sekvensdiagram ingå för uppkopplingen. IP-adresserna för samtliga noder skall anges och konfigurationsfilen skall bifogas.

Uppgift 3

Tillgång till 2 datorer behövs och samarbete kan ske mellan laborationsgrupper.

- Starta VIC från oSIPs meny genom att välja ”`v`”. Du får nu en mediasession med PC-kameran som sitter på skärmen.
- Genom att klicka på meny så öppnas ett konfigurationsfönster där du skall ställa in överföringshastigheterna på maximum och kodningen nv.
- Tryck på transmit och en miniatyrbild visas. För större bild klickar du på miniatyren.

Sessionen är lokal på datorn.

- Skriv om funktionen `respondto200forinvite()` i filen `example_agentlogic.c` så att VIC sänder till en annan dator i laborationssalen.

Funktionen används när ett ACK-meddelande skall skickas. Inparametern `response` innehåller föregående OK-meddelande. De structar och funktioner du kan behöva för att lösa uppgiften finns tillgängliga på kursens hemsida.
På den mottagande datorn måste en oSIP-applikation vara startad med ett portnummer (>5002) angivet i startkommandot efter en `–v`-flagga:

```
./ua -m 0 -f sip_myconf -d 5 -t sip:sven@cse.kau.se –v 8888
```

För att en överföring mellan två VIC-applikationer skall fungera skall båda startas med samma portnummer och den sändande parten skall ange mottagarens IP-nummer.

Uppgiften redovisas genom:

- Demonstration av uppkopplingen för laborationshandledaren.
- Beskriv utförligt i dokumentationen vad som ändrats.
- Bifoga den kod som modifierats.

Multimediadelen

Uppgift 4

Använd det program du modifierat för att starta en videoöverföring till en annan dator i laborationssalen. Använd kodningarna:

- nv
- h263+
- cellb.

I redovisningen skall ingå:

- En beskrivning av skillnaden i bildkvalité.
- Ange varför olika bildkodningar ger olika resultat.

Uppgift 5

Gör en uppkoppling till en bärbar dator, som du bokar och erhåller från laborationshandledaren.

- Ställ bildkodningen på nv. Maximera överföringshastighet och antalet bilder/sekund.
- Gå till fyra olika platser och kontrollera överföringshastigheten och antalet överförda bilder/sekund. Platserna där kontrollen skall ske är:
  1. Framför handikappskylt norr om 5-huset.
  2. I trappavsatsen mellan plan 2 och 3 i 12-huset.
  3. Korridoren framför dörr 12C314.
  4. Inne i sal 12A326.

Redovisa om, hur och varför mottagningskvalitén förändras samt ange de olika värdena, bilder/sekund och bitar/sekund i tabellform.

Structar i oSIP

```
typedef struct _startline_t {
  char *sipmethod;
  char *sipversion;
  url_t *rquri;
  char *statuscode;
  char *reasonphrase;
} startline_t ;

typedef struct _from_t {
  char *displayname;
  url_t *url;
  list_t *gen_params;
} from_t;

typedef from_t to_t;
typedef from_t contact_t;
typedef from_t record_route_t;
typedef from_t route_t;

typedef struct _sip_t {
  startline_t  *strtline;
  list_t    *accepts;
  list_t    *accept_encodings;
  list_t    *accept_languages;
  list_t    *alert_infos;
  list_t    *allows;
  authorization_t *authorization;
  call_id_t *call_id;
  list_t    *call_infos;
  list_t    *contacts;
  list_t    *content_dispositions;
  list_t    *content_encodings;
```

```
  content_length_t *contentlength;
  content_type_t   *content_type;
  cseq_t    *cseq;
  list_t    *error_infos;
  from_t    *from;
  mime_version_t   *mime_version;
  proxy_authenticate_t *proxy_authenticate;
  list_t    *proxy_authorizations;
  list_t    *record_routes;
  list_t    *routes;
  to_t      *to;
  list_t    *vias;
  www_authenticate_t *www_authenticate;
  list_t    *headers;
  list_t    *bodies;
  int   message_property;
  char *message;
} sip_t;

typedef struct _sipurl_t {
  char *scheme;
  char *username;
  char *password;
  char *host;
  char *port;
  list_t *url_params;
  list_t *url_headers;
} url_t ;

typedef struct _node_t {
  void *next;
  void *element;
} node_t;
```

```
typedef struct _list_t {
  int  nb_elt;
  node_t *node;
} list_t;
```

oSIP-funktioner

msg_setcontact

Name

msg_setcontact -- Parse *hvalue*, allocate and add a new contact attribute in the *sip* element.

Description

```
int msg_setcontact(sip_t *sip, char *hvalue);
```

on success, return 0.

msg_getcontact

Name

msg_getcontact -- set in *dest* the contact at index *pos* from the *sip* element.

Description

```
int msg_getcontact(sip_t *sip, int pos, contact_t **dest);
```

on success, return 0.

contact_init

Name

contact_init – Allocate and initialize the structure *contact*.

Description

```
int contact_init(contact_t **contact);
```

on success, return 0.

msg_setfrom

Name

msg_setfrom -- Parse *hvalue*, allocate and set the from attribute of the *sip* element.

Description

```
int msg_setfrom(sip_t *sip, char *hvalue);
```

on success, return 0.

msg_getfrom

Name

msg_getfrom -- return the from attribute from the *sip* element.

Description

```
from_t *msg_getfrom(sip_t *sip);
```

on success, return 0.

from_init

Name

from_init – Allocate and initialize the structure *from*.

Description

```
int from_init(from_t **from);
```

on success, return 0.

msg_setto

Name

msg_setto -- Parse *hvalue*, allocate and set the to attribute of the *sip* element.

Description

```
int msg_setto(sip_t *sip, char *hvalue);
```

on success, return 0.

msg_getto

Name

msg_getto -- return the *to* attribute from the *sip* element.

Description

```
to_t *msg_getto(sip_t *sip);
```

on success, return 0.

to_init

Name

to_init – Allocate and initialize the structure *to*.

Description

```
int to_init(to_t **to);
```

on success, return 0.

msg_setvia

Name

msg_setvia -- Parse *hvalue*, allocate and add a new via in *sip* element.

Description

```
int msg_setvia(sip_t *sip, char *hvalue);
```

on success, return 0.

msg_getvia

Name

msg_getvia -- set in *dest* the via at index *pos* from the *sip* element.

Description

```
int msg_getvia(sip_t *sip, int pos, via_t **via);
```

on success, return 0.

B Kod

B.1 app.c

```
/************************************************/
/* app.c i example_mt, oSIP 0.7.9               */
/* modifierad av Hasse Ellström och Berndt-Uno  */
/* Nyström för laboration i Datakommunikation II */
/* I filen är kod som stöder andra operativsystem */
/* än Linux borttagen.                          */
/* Ändringarna är markerade med fet stil        */
/************************************************/

#include "example_rcfile.h"
#include "app.h"
#include <osip/timers.h>
#include <sys/signal.h>

session_t *sessiontest;
int global_static_code=200; /* default return code. Can be changed by user
*/

#include <unistd.h>

#include <sys/types.h>
#include <sys/time.h>
#include <libgen.h>

osip_t *myconfig;

/*####* sträng för lagring av VICs portnummer *####*/

char *vic_port=NULL;

FILE *logfile;

void app_invite(char *tostring);
void app_bye();

void onexit();
void onsignal(int s);
void onalarm(int sig);

void set_debug_level(char *tmp)
{
  if (0==strncmp(tmp,"0",1)) {   TRACE_ENABLE_LEVEL(0); }
  if (0==strncmp(tmp,"1",1)) {   TRACE_ENABLE_LEVEL(1); }
  if (0==strncmp(tmp,"2",1)) {   TRACE_ENABLE_LEVEL(2); }
  if (0==strncmp(tmp,"3",1)) {   TRACE_ENABLE_LEVEL(3); }
  if (0==strncmp(tmp,"4",1)) {   TRACE_ENABLE_LEVEL(4); }
  if (0==strncmp(tmp,"5",1)) {   TRACE_ENABLE_LEVEL(5); }
  if (0==strncmp(tmp,"d",1)) {
  TRACE_DISABLE_LEVEL(0);
  TRACE_DISABLE_LEVEL(1);
```

```
  TRACE_DISABLE_LEVEL(2);
  TRACE_DISABLE_LEVEL(3);
  TRACE_DISABLE_LEVEL(4);
  TRACE_DISABLE_LEVEL(5);
  }
}
void set_sipreturn_code(char *tmp)
{
  if (0==strncmp(tmp,"c",1))
    {
    int i;
    char *newcode;
    newcode = (char *)smalloc(11);
    printf("<app.c> Give the new return code to use:\n");
    fgets(newcode,10,stdin);
    i = atoi(newcode);
    sfree(newcode);
    if (100<=i&&i<700)
      {
      global_static_code = i;
      printf("<app.c> Status code changed!\n");
      }
    else
      printf("<app.c> error: Status code NOT changed!\n");

    }
}

char* simple_readline(int descr, int forever)
{
  int ret;
  struct timeval tv;
  fd_set fset;
  tv.tv_sec = 0;
  tv.tv_usec = 0;
  FD_ZERO(&fset);
  FD_SET(descr, &fset);

  if (forever)
    ret = select(descr+1, &fset ,NULL ,
                      NULL, NULL );
  else
    ret = select(descr+1, &fset ,NULL ,
                      NULL, &tv );
  if (FD_ISSET(descr, &fset))
    {
    char *tmp;
    tmp = (char *)smalloc(201);
    read(descr,tmp,200);
    set_debug_level(tmp);
    set_sipreturn_code(tmp);
    return tmp;
    }
  return NULL;
}

void usage()
{
  printf("usage: ua -m mode -f config_file -d trace_level -t to_field\n");
  printf("mode is mandatory:\n");
  printf("\t0 for command mode\n");
```

```
  printf("\t1 for slow test mode (2 transactions/sec)\n");
  printf("\t2 for high load test (200 transactions/20sec)\n");
  printf("\t3 for UAS behavior (only answer to transactions)\n");

  printf("config_file is mandatory:\n");
  printf("\tfile contains the whole configuration of SIP agent\n");
  printf("trace_level is the level of trace\n");
  printf("\ttrace_level is an integer between 0 and 5\n");

  printf("to_field is the url of the correspondant\n");
  printf("\turl looks like <sip:jack@127.0.0.1>\n");
}

int main(int argc, char **argv)
    {
    mt_ua_t *mt_ua;
    char *tostring = NULL;
    char *configfile = NULL;
    int trace_level = 2; /* should be between 0 and 5 */
    int mode = 0;        /* should be between 0 and 2 */
    int arg_num;

    /* signal(SIGSEGV, &onsignal); */
    printf("This is a facility to test the oSIP library!\n");
    if ((argc<2))
      {
      usage();
      exit(0);
      }

    arg_num = 1;
    while (arg_num<argc)
      {
      if (strlen(argv[arg_num])!=2)
           {
           usage();
           exit(0);
           }
      if (strncmp("-f",argv[arg_num],2)==0)
           {
           /* the next arg is the config file */
           arg_num++;
           configfile = sgetcopy(argv[arg_num]);
           }
      if (strncmp("-t",argv[arg_num],2)==0)
           {
           /* the next arg is the address in the to field */
           arg_num++;
           tostring =  sgetcopy(argv[arg_num]);
           }
      if (strncmp("-m",argv[arg_num],2)==0)
           {
           /* the next arg is the mode to start... */
           arg_num++;
           mode = atoi(argv[arg_num]);
           }
      if (strncmp("-d",argv[arg_num],2)==0)
           {
           /* the next arg is the trace level... */
           arg_num++;
           trace_level = atoi(argv[arg_num]);
```

```
          }

/*####* Om portnummer till VIC anges i kommandoraden *####*/

      if (strncmp("-v",argv[arg_num],2)==0)
           {
           arg_num++;
           vic_port = sgetcopy(argv[arg_num]);
           }
      arg_num++;
      }

/*####* Slut portnummer till VIC *####*/

    {
    char *tmp;
    tmp = smalloc(strlen(argv[0])+10);
    sprintf(tmp,"%s_%i",basename(argv[0]),(int)getpid());
    logfile = fopen(tmp, "w");
    /* init logger facility */
    sfree(tmp);
    }
    if (configfile==NULL)
      {
      usage();
      exit(0);
      }

    TRACE_INITIALIZE(trace_level,logfile);

    if ((mode!=3)&&(tostring==NULL))
      {
      usage();
      exit(0);
      }


    {
      int i;
      /* Load the configuration */
      i = loadsipconf(configfile);
      if (i!=0)
           {
             usage();
             exit(0);
           }
      sfree(configfile);
    }

    { /* set the proxy if you want... */
      char *proxy = getsipconf("sipproxy");
      url_t *prox;
      osip_init(&myconfig);
      if (proxy==NULL)
             fprintf(stdout,"<app.c> You will not use a proxy in this
configuration\n");
      else
           {
             int i;
             i = url_init(&prox);
             i = url_parse(prox,proxy);
```

```
            osip_init_proxy(myconfig, prox);
            if (i==-1)
              {
                fprintf(stdout,"<app.c> the URL of the proxy is not valid!
check the value in the rc file\n");
                return -1;
              }
          }
    }

    mt_ua_init(&mt_ua, atoi(getsipconf("localport")), myconfig);

    mt_ua_start_design1(mt_ua);

    if (mode==2) /* high load test */
      {
      int active = 1;
      int i=10;
      printf("Entering high load test (launch 200 transactions each
20s!\n");
      printf("keystrokes:\n");
      printf("0 1 2 3 4 5: enable trace levels   d: disable trace\n");
      printf("c: new static return code wanted\n");
      printf("s: stop test\n");
      printf("r: restart test\n");
      while (i!=0)
           {
           int j=50;
           char *tmp;

           tmp = simple_readline(0,0);
           if (tmp!=NULL)
             {
             if (strncmp(tmp,"s",1)==0)
               active = 0;
             if (strncmp(tmp,"r",1)==0)
               active = 1;
             sfree(tmp);
             }
           if (active==1)
             {
               int pending_transactions = list_size(myconfig-
>uac_transactions)
                 + list_size(myconfig->uas_transactions);
               /* limit the number of transactions */
             while (j!=0)
               {
                 if (pending_transactions<600)
                      app_invite(tostring);
                 else
                      susleep(200000); /* the application is overloeaded...
*/
                 j--;
               }
             /*  i--; */ /* loop for ever... */
             }
           /* susleep(1000000); */
           }
      }
    if (mode==1) /* slower test */
      {
```

```
      int active = 1;
      char *tmp;
      printf("keystrokes:\n");
      printf("0 1 2 3 4 5: enable trace levels   d: disable trace\n");
      printf("c: new static return code wanted\n");
      printf("s: stop test\n");
      printf("r: restart test\n");
      while (1)
           {
           tmp = simple_readline(0,0);
           if (tmp!=NULL)
             {
             if (strncmp(tmp,"s",1)==0)
               active = 0;
             if (strncmp(tmp,"r",1)==0)
               active = 1;
             sfree(tmp);
             }
           susleep(500000); /* ok for 1s */
           if (active==1)
             {
             app_invite(tostring);
             /* i++; */
             }
           }
      }

    if (mode==0) /* Command mode */
      {
      char *tmp;
      int i=1;

      while (i!=0)
           {
           printf("keystrokes:\n");
           printf("0 1 2 3 4 5: enable trace levels   d: disable trace\n");
           printf("c: new static return code wanted\n");
           printf("i: start an INVITE transaction    t: start N INVITE
trans\n");
           printf("r: start a REGISTER transaction   b: start a BYE
trans\n");
           printf("v: start a VIC-session\n");
           printf("osip command> ");
           tmp = simple_readline(0,1);
           /*fgets(tmp, 100, stdin); */

/*####* Nytt menyval för start av VIC-applikation *####*/

           if (0==strncmp(tmp,"v",1))
             {
             if (!vic_port)
               vic_port=sgetcopy("6664");
             fprintf(stdout,"INVITE\n");
             app_invite(tostring);
             }

/*####* Slut menyval för start av VIC-applikation *####*/

           if (0==strncmp(tmp,"i",1))
             {
             fprintf(stdout,"INVITE\n");
```

```
            app_invite(tostring);
            }
          if (0==strncmp(tmp,"r",1))
            {
            char *expires;
            expires = (char *) smalloc(5);

/*####* Här ändras tiden som begärs i REGISTER-meddelandet  *####*/

            sprintf(expires,"%i",1200);

            fprintf(stdout,"REGISTER expires after: %s\n",expires);
            sip_register(expires);
            sfree(expires);
            }
          if (0==strncmp(tmp,"b",1))
            {
            fprintf(stdout,"BYE transaction: \n");
            app_bye();
            }
          if (0==strncmp(tmp,"t",1))
            {
            int j;
            sfree(tmp);
            tmp = (char *)smalloc(100);
            fprintf(stdout,"Give the number of transactions to start\n");
            fgets(tmp, 100, stdin);
            j = atoi(tmp);
            fprintf(stdout,"INVITE * %i\n",j);
            while (j!=0)
              {
              app_invite(tostring);
              j--;
              }
            }
          if (tmp!=NULL) sfree(tmp);
          }
      }

    while (mode==3) /* only a UAC behavior */
      {
      char *tmp;
      printf("keystrokes:\n");
      printf("0 1 2 3 4 5: enable trace levels   d: disable trace\n");
      printf("c: new static return code wanted\n");
      tmp = simple_readline(0,1);
      if (tmp!=NULL) sfree(tmp);
      }
    return 1; /* ok */

    }


  void
    uaapp_annouceoutinvite(sip_t *invite,transaction_t *tr)
    {
    }

  void
    uaapp_annouceoutrequest(sip_t *request,transaction_t *tr)
    {
```

```
  }

void
  uaapp_annouceincresponse(sip_t *response,transaction_t *tr)
  {
  /* Here we SHOULD give order to send ack.         */
  /* (only for INVITE's response and for code>=200) */

  if (MSG_IS_RESPONSEFOR(response,"INVITE")
        && !MSG_TEST_CODE(response,100)
        && !MSG_TEST_CODE(response,180))
    respondto200forinvite(tr,response);
  }

void
  uaapp_annouceincack(sip_t *request,transaction_t *tr)
  {
  /* do nothing */
  }

void
  uaapp_annouceincrequest(sip_t *request,transaction_t *tr)
  {
  /* autoresponse 200 to all request except ACK */
  if (!MSG_IS_ACK(request))
    respondtorequest(tr,global_static_code);
  }

void
  uaapp_annouceincinvite(sip_t *invite,transaction_t *tr)
  {
  respondtoinvite(tr,180);
  /* usleep(10000000); timeout to allow cancel*/
  respondtoinvite(tr,global_static_code);
  }
void
  uaapp_annouceoutresponse(sip_t *response,transaction_t *tr)
  {

  }

void
  app_invite(char *tostring)
  {
  static int only_once = 0;

  if (only_once == 0)
    {
    only_once = 1;
    sessiontest   = (session_t *)smalloc(sizeof(session_t));
    to_init(&(sessiontest->to));
    to_parse(sessiontest->to, tostring);

    {
    header_t *header;
    header_init(&header);

    header->hname = (char *) smalloc(strlen("subject")+1);
    sstrncpy(header->hname,"subject",strlen("subject"));
    header->hvalue = (char *) smalloc(strlen("Hello.")+1);
    sstrncpy(header->hvalue,"Hello.",strlen("Hello."));
```

```
      sessiontest->headers = (list_t *) smalloc(sizeof(list_t));
      list_init(sessiontest->headers);
      list_add(sessiontest->headers,header,-1);
      }
      {
      header_t *header;
      header_init(&header);

      header->hname = (char *) smalloc(strlen("User-Agent")+1);
      sstrncpy(header->hname,"User-Agent",strlen("User-Agent"));
      header->hvalue = (char *) smalloc(strlen("oSIP-UA/0.6.0")+1);
      sstrncpy(header->hvalue,"oSIP-UA/0.6.0",strlen("oSIP-UA/0.6.0"));

      sessiontest->headers = (list_t *) smalloc(sizeof(list_t));
      list_init(sessiontest->headers);
      list_add(sessiontest->headers,header,-1);
      }

      sessiontest->from = NULL;
      sessiontest->contact = NULL;
      sessiontest->proxy = NULL;
      }
    sessiontest->transactionid = sip_invite(&sessiontest);

    }


  void app_bye()
    {
    sip_bye(&sessiontest);
    }

  /* for use of int atexit(void (*func)(void) */
  void
    onexit(){
  fclose(logfile);
  /* close(currentsipconfig->udpout_sock); to be done in stack? */
  TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,stdout,"program end normally
:)\n"));
  exit(1);
  }
```

B.2 example_agentlogic.c

```
/************************************************/
/* example_agentlogic i example_mt, oSIP 0.7.9     */
/* modifierat av Hasse Ellström och Berndt-Uno     */
/* Nyström för laboration i Datakommunikation II   */
/* I filen är kod som stöder andra operativsystem  */
/* än Linux borttagen. Filen modifieras under      */
/* laborationen.                                   */
/* Ändringarna är markerade med fet stil           */
/************************************************/

#include <sys/types.h>
#include <unistd.h>

#include "app.h"
#include "example_rcfile.h"
#include <osip/sdp.h>

/* #include <unistd.h> */

extern osip_t *myconfig;
extern char *vic_port;

char* getbodyfor200(sip_t *request)
{
    sdp_t *remote_sdp;
    sdp_t *local_sdp;
    int i;
    char *local_body;
    local_body = NULL;
    if (MSG_IS_INVITE(request))
      {
           body_t *body;
           body = (body_t *)list_get(request->bodies,0);

           local_sdp = (sdp_t *) smalloc(sizeof(sdp_t));
        remote_sdp = (sdp_t *) smalloc(sizeof(sdp_t));

           /* WE ASSUME IT IS A SDP BODY AND THAT    */
           /* IT IS THE ONLY ONE, OF COURSE, THIS IS */
           /* NOT TRUE */
           sdp_parse(remote_sdp,body->body);

           i =
sdp_replyto(remote_sdp,local_sdp,getsipconf("username"),getsipconf("network
type"),getsipconf("addr_type"),getsipconf("localip"));
           local_body = sdp_2char(local_sdp);


           sdp_free(remote_sdp);
           sfree(remote_sdp);

           sdp_free(local_sdp);
           sfree(local_sdp);

      }

    return local_body;
```

```
}

/* fill default feilds for reply.                    */
/* OUTPUT: sip_t *dest | structure to store response.*/
/* INPUT : sip_t *request | previous request.        */
/* INPUT : int status | status for response.         */
/* INPUT : char *reason | reasonphrase for response. */
/* INPUT : char *body | body for response.           */
/* INPUT : int options | add a tag in to header.     */
/* return -1 on error */
/* TODO: add built-in support for record-route and route */
int msg_makereply(int status,char *reason,sip_t *request,
                  sip_t *response,char *body,char *content_type
                  ,int options)
{
  int pos;
  int i;

  /* initialise sip_t structure */
  /* yet done... */

  response->strtline->sipversion = (char *)smalloc(8*sizeof(char));
  sprintf(response->strtline->sipversion,"SIP/2.0");
  response->strtline->statuscode = (char *)smalloc(5*sizeof(char));
  sprintf(response->strtline->statuscode,"%i",status);
  response->strtline->reasonphrase = (char *)smalloc(strlen(reason)+1);
  sprintf(response->strtline->reasonphrase,"%s",reason);

  response->strtline->rquri     = NULL;
  response->strtline->sipmethod = NULL;

  /* should add a tag in to feild (for useragent only) */
  /* add a boolean to reqest for it */
  {
    char *tmp;
    i = to_2char(request->to, &tmp);
    if (i==-1) return -1;
    i = msg_setto(response, tmp);
    sfree(tmp);
    if (i==-1) return -1;
    /*
      if (options==ADD_TAG)
      {
      response->to->tag = (char *)smalloc(12);
      sstrncpy(response->to->tag,"tag=faketag",11);
      }
    */
  }
  {
    char *tmp;
    i = from_2char(request->from, &tmp);
    if (i==-1) return -1;
    i = msg_setfrom(response, tmp);
    sfree(tmp);
    if (i==-1) return -1;
  }

  /* via headers */
  {
    char *tmp;
    pos = 0;
```

```
    while (!list_eol(request->vias,pos))
      {
           via_t *via;
           via = (via_t *)list_get(request->vias,pos);
           i = via_2char(via, &tmp);
           if (i==-1) return -1;
           i = msg_setvia(response, tmp);
           sfree(tmp);
           if (i==-1) return -1;
           pos++;
      }
  }

  {
    char *tmp;
    i = call_id_2char(request->call_id, &tmp);
    if (i==-1) return -1;
    i = msg_setcall_id(response, tmp);
    sfree(tmp);
    if (i==-1) return -1;
  }
  {
    char *tmp;
    i = cseq_2char(request->cseq, &tmp);
    if (i==-1) return -1;
    i = msg_setcseq(response, tmp);
    sfree(tmp);
    if (i==-1) return -1;
  }

   /* add personnal contact information (to be used for next request */
   /* (only for useragent) */
   if (body!=NULL)
     {
       char *size;
       i = msg_setbody(response, body);
       if (i==-1) return -1;

       size = (char *) smalloc(6*sizeof(char));
       sprintf(size,"%i",strlen(body));
       i = msg_setcontent_length(response, size);
       sfree(size);
       if (i==-1) return -1;

       {
            char *ct = sgetcopy("content-type");
            i = msg_setheader(response, ct, content_type);
            sfree(ct);
            if (i==-1) return -1;
       }
     }
   else
     {
       char *cl = sgetcopy("0");
       i = msg_setcontent_length(response, "0");
       sfree(cl);
       if (i==-1) return -1;
     }

   return 1;
}
```

```
int
osip_ul_sendmsg(transaction_t *transaction,sip_t *msg)
{
    sipevent_t *sipevent;

    sipevent = osip_new_outgoing_sipmessage(msg);
    sipevent->transactionid =  transaction->transactionid;

    uaapp_annouceoutrequest(sipevent->sip,transaction);
    fifo_add(transaction->transactionff,sipevent);

    return 0;
}

void respondto200forinvite(transaction_t *tr, sip_t *response)
{
  sip_t *ack;
  msg_init(&ack);


  if (1==makeack(ack,response))
  {

/*####* Ny kod för laboration som startar VIC-applikation *####*/

/*####* Koden ersätts med lösningsförslaget nedan *####*/

    if(vic_port!=NULL)
    {
      if(fork() == 0)
        execlp("vic","vic","127.0.0.1/6668",NULL);
      free(vic_port);
      vic_port=NULL;
    }

/*####* Lösningsförslag till laborationsuppgift *####*/

/*
    if(vic_port!=NULL)
    {
      char* arg = (char*)malloc(25);
      contact_t **contact;
      contact = (contact_t**)malloc(sizeof(contact));
      contact_init(&contact[0]);
      msg_getcontact(response,0,contact);
      strcpy(arg,contact[0]->url->host);
      strcat(arg,"/");
      strcat(arg,vic_port);
      if(fork() == 0)
        execlp("vic","vic",arg,NULL);
      free(vic_port);
      vic_port=NULL;
    }
*/

/*####* Slut på kod för start av VIC-applikation *####*/

      osip_ul_sendmsg(tr,ack);
    }
  else
```

```
    {
    TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,NULL,"Error while creating
ACK request\n"));
    msg_free(ack);
    sfree(ack);
    }
}

void respondtoinvite(transaction_t *tr, int code )
{

/*####* Ny kod för laboration som startar VIC-applikation *####*/

  if(vic_port!=NULL)
  {
    char* arg = (char*)malloc(25);
    arg=getsipconf("localip");
    strcat(arg,"/");
    strcat(arg,vic_port);
    if(fork() == 0)
      execlp("vic","vic",arg,NULL);
    free(vic_port);
    vic_port=NULL;
  }

/*####* Slut på kod för start av VIC-applikation *####*/

  respondtorequest(tr,code);
}

void respondtorequest(transaction_t *tr, int code )
{
  char *reason;
  char *apptype;
  sip_t *response;
  char *local_body;

  msg_init(&response);
  reason = msg_getreason(code);

  if (code<200&&code!=183)
    local_body = NULL;
  else
    {
      if ( !MSG_IS_BYE(tr->lastrequest)
              && !MSG_IS_REGISTER(tr->lastrequest)
              && !MSG_IS_CANCEL(tr->lastrequest) )
           {
             local_body =  getbodyfor200(tr->lastrequest);
             if (local_body==NULL)
               code = 415;
             /* WE MUST ADD THE CORRESPONDING WARNING ERROR */
             /* getWarningphrase(i); */
           }
      else
           {
             local_body=NULL;
           }
    }

  if (local_body!=NULL) /* only sdp is allowed in the test app.... */
```

```
    apptype = sgetcopy("application/sdp");
  else
    apptype = NULL;

  if (1==msg_makereply(code,reason,
                               tr->lastrequest,
                               response,
                               local_body,
                               apptype,0))
    {
      msg_setcontact(response, getsipconf("contact"));
      osip_ul_sendmsg(tr,response);
    }
  else
    {
      if (local_body!=NULL)
           sfree(local_body);

TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,NULL,"<example_agentlogic.c>
Could not create response for current status\n"));
      sfree(response);
      sfree(reason);
      if (apptype!=NULL)
           sfree(apptype);
      return ;
    }
  if (local_body!=NULL)
    sfree(local_body);
  sfree(reason);
  if (apptype!=NULL)
    sfree(apptype);
}


int sip_invite(session_t **session)
{
  static int only_once = 0;
  transaction_mt_t *transaction_mt;
  sip_t         *invite;
  transaction_t *transaction;
  char *tmp;

  msg_init (&invite);
printf("\nFÖRE\n");


  {/* mandatory */
    to_2char((*session)->to, &tmp);
printf("\nEFTER\n");
    msg_setto(invite,tmp);
    sfree(tmp);
  }

  /* this is used for subject and User-Agent header */
  if ((*session)->headers!=NULL)
    {
    int pos = 0;
    while (!list_eol((*session)->headers,pos))
      {
      header_t *header;
      header = (header_t *)list_get((*session)->headers,pos);
```

```
      msg_setheader(invite,header->hname,header->hvalue);
      pos++;
      }
    }
  /* do something for proxy ?!*/

  makeinvite(invite);

  if (only_once==0) {
    /* this is not used and not free() by any layer during test! */
    (*session)->callid = (call_id_t *)smalloc(sizeof(call_id_t));
    (*session)->callid->number = sgetcopy(invite->call_id->number);
    (*session)->callid->host   = sgetcopy(invite->call_id->host);

    (*session)->cseq = (cseq_t *)smalloc(sizeof(cseq_t));
    (*session)->cseq->number = sgetcopy(invite->cseq->number);
    (*session)->cseq->method = sgetcopy(invite->cseq->method);

    only_once = 1;
  }

  transaction = (transaction_t *) smalloc(sizeof(transaction_t));
  transaction_init(myconfig,
                        transaction,
                        invite->to,
                        invite->from,
                        invite->call_id,
                        invite->cseq);
  transaction_mt_init(&transaction_mt);
  transaction_mt_set_transaction(transaction_mt,transaction);
  transaction_mt_start_transaction(transaction_mt);

  osip_ul_sendmsg(transaction,invite);

  return transaction->transactionid;
}

int sip_bye(session_t **session)
{
  transaction_mt_t *transaction_mt;
  sip_t         *bye;
  transaction_t *transaction;

  msg_init (&bye);

  makebye(bye,NULL,(*session)->to,NULL
            ,(*session)->callid,(*session)->cseq);
  fprintf(stdout,"BYE DONE: \n");


  transaction = (transaction_t *) smalloc(sizeof(transaction_t));
  transaction_init(myconfig,
                        transaction,
                        bye->to,
                        bye->from,
                        bye->call_id,
                        bye->cseq);
  transaction_mt_init(&transaction_mt);
  transaction_mt_set_transaction(transaction_mt,transaction);
  transaction_mt_start_transaction(transaction_mt);
```

```
  osip_ul_sendmsg(transaction,bye);

  return transaction->transactionid;
}

void sip_register(char *expires)
{
  transaction_mt_t *transaction_mt;
  sip_t         *sregister;
  transaction_t *transaction;
  char *registrationnumber = NULL;
  static call_id_t *registrationcallid = NULL;
  static int regnumber = 1;
  if (registrationcallid==NULL)
    {
      registrationcallid = (call_id_t *) smalloc(sizeof(call_id_t));
      call_id_setnumber(registrationcallid,sgetcopy("1239874324324"));
      call_id_sethost(registrationcallid,sgetcopy(getsipconf("localip")));
    }
  regnumber++;
  registrationnumber = (char *)smalloc(7*sizeof(char));
  sprintf(registrationnumber,"%i",regnumber);

  msg_init (&sregister);
  makeregister(sregister,registrationcallid,registrationnumber,expires);

  sfree(registrationnumber);

  transaction = (transaction_t *) smalloc(sizeof(transaction_t));
  transaction_init(myconfig,
                        transaction,
                        sregister->to,
                        sregister->from,
                        sregister->call_id,
                        sregister->cseq);
  transaction_mt_init(&transaction_mt);
  transaction_mt_set_transaction(transaction_mt,transaction);
  transaction_mt_start_transaction(transaction_mt);
  osip_ul_sendmsg(transaction,sregister);

  return ;
}
```

B.3 msg_ack.c

```
/************************************************/
/* msg_ack.c i example_mt, oSIP 0.7.9           */
/* modifierad av Hasse Ellström och Berndt-Uno  */
/* Nyström för laboration i Datakommunikation II */
/* Ändringarna är markerade med fet stil        */
/************************************************/

/*
 FILE: message.c
 AUTHOR: AYMERIC MOIZARD
 features: declaration of the SIP message format
*/

#include <osip/smsg.h>
#include "example_rcfile.h"

int
makeack(sip_t *dest,sip_t *response)
{
  char *tmp;

  from_2char(response->from, &tmp);
  msg_setfrom(dest, tmp);
  sfree(tmp);


  to_2char(response->to, &tmp);
  msg_setto(dest, tmp);
  sfree(tmp);


  /* create Request-URI like for invite */
  dest->strtline->sipmethod = (char *)smalloc(14);
  sprintf(dest->strtline->sipmethod,"ACK");
  dest->strtline->sipversion = (char *) smalloc(8*sizeof(char));
  sprintf(dest->strtline->sipversion,"SIP/2.0");

  dest->strtline->statuscode   = NULL;
  dest->strtline->reasonphrase = NULL;

  url_init(&(dest->strtline->rquri));

/*####* Koden innan ändring för laborationsplattform *####*/

/*
  url_2char(dest->to->url, &tmp);
  url_parse(dest->strtline->rquri, tmp);
*/

/*####* Ny kod för att styra om ACK-meddelandet *####*/

{
  contact_t **contact;
  tmp=(char *)smalloc(30);

  contact = (contact_t**)malloc(sizeof(contact));
  contact_init(&contact[0]);
```

```
  /* Adressen ligger i contactfältet i response */
  msg_getcontact(response,0,contact);

  /* Adressen byggs om för att passa parsering */
  sprintf(tmp,"sip:");
  strcat(tmp,contact[0]->url->host);
  strcat(tmp,":");
  strcat(tmp,contact[0]->url->port);

  url_parse(dest->strtline->rquri, tmp);

  dest->strtline->rquri->username=contact[0]->url->username;
}

/*####* Slut på kod för att styra om ACK-meddelandet *####*/

  sfree(tmp);

  /* if some headers and params exist remove unwanted information if
     They do not appear on request URI */
  /* TODO...  */

  tmp = (char *)smalloc(60*sizeof(char));
  sprintf(tmp,"SIP/2.0/UDP
%s:%s",getsipconf("localip"),getsipconf("localport"));
  msg_setvia(dest, tmp);
  sfree(tmp);

  call_id_2char(response->call_id, &tmp);
  msg_setcall_id(dest, tmp);
  sfree(tmp);


  /* MODIFICATION IS REQUIRED, IT MUST BE "number ACK" */
  cseq_init(&(dest->cseq));
  cseq_setnumber(dest->cseq,sgetcopy(response->cseq->number));
  cseq_setmethod(dest->cseq,sgetcopy("ACK"));

  msg_setcontact(dest, getsipconf("contact"));

  msg_setcontent_length(dest, "0");

  return 1;
}
```

B.4 udp_send.c

```
/************************************************/
/* udp_send.c i example_mt, oSIP 0.7.9          */
/* modifierad av Hasse Ellström och Berndt-Uno  */
/* Nyström för laboration i Datakommunikation II */
/* Ändringarna är markerade med fet stil        */
/************************************************/

/*
  The oSIP library implements the Session Initiation Protocol (SIP -
rfc2543-)
  Copyright (C) 2001  Aymeric MOIZARD jack@atosc.org

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

#include <stdio.h>

#ifndef __VXWORKS_OS__
#include <errno.h>
#include <ctype.h>
#include <time.h>
#include <sys/types.h>
#include <stdlib.h>
#ifdef WIN32
#include <winsock.h>
#else
#include <sys/time.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif
#endif

#ifdef __VXWORKS_OS__
#include "vxWorks.h"
#include "sockLib.h"
#include "inetLib.h"
#include "stdioLib.h"
#include "strLib.h"
#include "ioLib.h"
#include "fioLib.h"
#endif
```

```
#include <osip/port.h>
#include <osip/const.h>
#include <osip/smsg.h>


/* create one outgoing socket for the whole session.  */
/* return the socket. */
/* This is only temporary. a new transport layer will */
/* be added in the future. */
int
getsock()
{
  static int udpout_sock = 0;
  if (udpout_sock==0)
    {
      udpout_sock = (int)socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
      if (udpout_sock==-1)
           {
             perror("failed to open socket!");
             return -1;
           }
    }
  return udpout_sock;
}

/* send a message to host:port on UDP.          */
/* INPUT : char *host | host to be used.        */
/* INPUT : int port   | port to be use.         */
/* INPUT : char *message | message to be sent.  */
/* returns null on error. */
int
udp_send(char *message,char *host,int port)
{
  struct hostent     *hp;
#ifdef __linux /* ok for GNU/linux */
  struct hostent     **hp_ptr;
#else
#  ifdef __sun__ /* ok for solaris (Not yet tested MACRO??) */
  struct hostent     *hp_2;
#  else
#    ifdef __unix
  struct hostent     *hp_2;
#    endif
#  endif
#endif
  struct sockaddr_in addr;
  unsigned long int  one_inet_addr;
  int sock;

  hp = NULL;

#ifdef __linux /* avoid warning before replacing the gethostbyname_r */
  hp_ptr = NULL;
#endif
  /* TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,stdout
      ,"<udp_send.c> sending message to %s on port %i\n"
      ,host,port)); */

  if ((int)(one_inet_addr = inet_addr(host)) == -1)
    {
```

```
      DEBUG(fprintf(stdout,"<udp_send.c> TODO: There is no support for name
resolution!!!\n"));
      /* this has to be moved to the sutil module */
      /* Not implemented yet
#ifndef __VXWORKS_OS__
      int i;
      int *herrno = (int *) smalloc(sizeof(int ));
      char *pbuf = (char *) smalloc(513*sizeof(char));
      hp = (struct hostent *)smalloc(sizeof(struct hostent));
      TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,stdout
             ,"<udp_send.c>  resolving address! !?\n"));
#ifdef __linux
      hp_ptr = (struct hostent **)smalloc(sizeof(struct hostent*));
      i = gethostbyname_r(host,hp,pbuf,512,hp_ptr,herrno);
#else
#  ifdef __sun__
      hp_2 = gethostbyname_r(host,hp,pbuf,512,herrno);
#  else
#    ifdef __unix
      hp_2 = gethostbyname_r(host,hp,pbuf,512,herrno);
#    endif
#  endif
#endif

      if (*herrno<0)
           {
             TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL1,stdout
                 ,"<udp_send.c> Could not resolve address
(error:%i)!!?\n",herrno));
             return -1;
           }
      addr.sin_addr.s_addr = *((long *)(hp->h_addr));
      sfree(pbuf);
      sfree(hp);
#ifdef  __linux
      sfree(hp_ptr);
#else
#  ifdef __sun__
      sfree(herrno);
#  else
#    ifdef __unix
      sfree(herrno);
#    endif
#  endif
#endif

#endif
 */
    }
  else
    {
      addr.sin_addr.s_addr = one_inet_addr;
    }

  addr.sin_port        = htons((short)port);
  addr.sin_family      = AF_INET;

  sock = getsock();

  /* If we want to detect ICMP message and ECONNREFUSED
     errors, this line must be commented. */
```

```
#ifndef WIN32
  connect(sock,(struct sockaddr *) &addr,sizeof(addr));
#endif
  if (0  > sendto (sock, (const void*) message, strlen (message), 0,
                           (struct sockaddr *) &addr, sizeof(addr)))
    /* if (sendto (sock, (caddr_t) message, strlen (message), 0,
       (struct sockaddr *) &addr, sizeof(addr)) < 0) */
    {
#ifdef WIN32
      if (WSAECONNREFUSED==WSAGetLastError())
#else
      if (ECONNREFUSED==errno)
#endif
           {
             TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL3,stdout,
                      "SIP_ECONNREFUSED - No remote server.\n"));
             /* This can be considered as an error, but for the moment,
                I prefer that the application continue to try sending
                message again and again... so we are not in a error case.
                Nevertheless, this error should be announced!
                ALSO, UAS may not have any other options than retry always
                on the same port.
             */
             return 1;
           }
      else
           {
             TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL3,stdout,
                      "SIP_NETWORK_ERROR - Network error.\n"));
             /* SIP_NETWORK_ERROR; */
             return -1;
           }
    }

  /* #endif */
  TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2, NULL,
                "SND UDP MESSAGE.\n"));

  return 0;
}


int
udp_send_request(sip_t *request, url_t *proxy)
/* day3: support for proxy usage
   udp_send_request(sip_t *request) */
{
  char *mess;
  url_t *url;
  int port;
  int i;

/*####* Ny kod för laborationen, skickar ej via proxy vid ACK *####*/
/*
  if (proxy==NULL)
    url = request->strtline->rquri;
*/
  if (MSG_IS_ACK(request) || proxy==NULL)
    url = request->strtline->rquri;

/*####* Ny kod för laborationen, skickar ej via proxy vid ACK *####*/
```

```
  else
    url = proxy;

  if (url->port!=NULL)
    port = atoi(url->port);
  else
    port = 5060;

  i = msg_2char(request, &mess);

  if (i==-1)
    {
    TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2,stdout,"<udp_send.c> Error
while creating REQUEST\n"));
    return -1; /* error */
    }

  i = udp_send(mess, url->host, port);
  sfree(mess);
  if (i==-1)
    {
      TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2,stdout,"<udp_send.c> Error
while sending REQUEST\n"));
      return -1; /* error */
    }
  DEBUG(msg_logrequest(request,"SND %s f:%s t:%s cseq:%s callid:%s\n"));
  return 0;
}

int
udp_send_response(sip_t *response)
{
  char *mess;
  via_t *via;
  int port;
  int i;

  /*  via = (via_t *)smalloc(sizeof(via_t)); */
  if (-1==msg_getvia(response,0,&via))
    {
      TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2,stdout,"<udp_send.c> Error
in via header\n"));
      return -1;
    }
  i = msg_2char(response, &mess);
  if (i==-1)
    {
      TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2,stdout,"<udp_send.c> Error
could not create message\n"));
      return -1;
    }
  if (via->port!=NULL)
    port = atoi(via->port);
  else
    port = 5060;

  i = udp_send(mess,via->host, port);
  sfree(mess);
  if (i==-1)
    {
```

```
      TRACE(trace(__FILE__,__LINE__,TRACE_LEVEL2,stdout,"<udp_send.c> Error
while sending REQUEST\n"));
      return -1; /* error */
    }
  DEBUG(msg_logresponse(response,"SND %s %s (%s) f:%s t:%s cseq:%s
callid:%s\n"));
  return 0; /* return 1 on success and i on error */
}
```