Micro-controller based Internet Phone

Shaun Kaplan

Supervisor: Dr Jevon Davies

Centre for Instrumentation Research

Submitted in fulfilment of the
requirements for the
degree of Magister Technologiae
in the Department of Electrical Engineering

Cape Technikon

March 2004

Declaration

I, Shaun Kaplan, declare that the work contained within this thesis is that of my own and has not previously been submitted for academic examination towards any qualification. The opinions expressed in this thesis are not necessarily those of the Cape Technikon.

Acknowledgements

I would like to acknowledge the following people and groups who in some way or another contributed towards the completion of this thesis:

- Firstly I would like to thank Dr Bruce Mortimer, my original supervisor, who thought it would be a good idea for me implement my idea as an MTech and for helping out even from across the Atlantic.
- Dr Jevon Davies, Jedi master, who took over from Bruce as my supervisor.
- David and Estelle Gerber for inspiring the concept.
- All at the CIR for their support.
- Gary Jacobson for proof reading the near complete thesis and listening to my many thesis related rants.
- My friends and family.
- Everyone who took an interest in my project.

This thesis is dedicated to all who heard the words 'still working on my thesis'.

Synopsis

This work describes research towards the development of a micro-controller based, stand-alone Internet telephone to be used as an alternative to conventional line telephones. Our definition of 'stand-alone' refers to the unit's capability to perform its function wholly without the need for an attached computer. The unit should be low cost and capable of allowing two users to communicate using the units. Bandwidth usage should be kept low to allow the unit to be used over dial up connections which are prevalent in South Africa. The units should be easy to use as the anticipated users may be unskilled.

A module containing a 16-bit micro-controller, an Ethernet controller, flash memory and RAM was chosen as the controller. The module came with a real-time operating system and a TCP/IP stack.

The session initiation protocol (SIP) was selected to perform the signalling. SIP uses the session description protocol (SDP) to negotiate the attributes of the media session to be established.

The real-time transport protocol (RTP) was implemented to transport encoded audio between the end points. The RTP control protocol (RTCP) was implemented to provide basic quality of service parameters.

The ITU-T recommendation G.729 annex A was the voice codec selected. Codec ICs were used to encode and decode the audio.

The implementations were designed specifically for a two user, direct communication environment. That is two phone units were developed that communicated directly with each other and not through intermediary servers.

Contents

List of Figures

List of Tables

Glossary

A

ABNF — Augmented Backus-Naur Form.

B

bps — Bits per second.

byte — A string of bits, usually eight. In this text a byte will always refer to eight bits.

C

callee — User to whom the call is placed.

caller — User placing the call.

CNAME — Canonical name — an end-point identifier SDES item.

codec — Coder / decoder or compressor / decompressor.

E

Ethernet — A local area network technology.

G

G.729 — ITU-T Recommendation G.729 : Coding of speech at 8kbit/s using Conjugate Structure Algebraic-Code-Excited-Linear-Prediction (CS-ACELP).

H

H.323 ITU-T Recommendation H.323 : Packet-based multimedia communications systems.

I

IETF Internet Engineering Task Force.

ISP Internet Service Provider.

IP Internet Protocol.

ITU International Telecommunication Union.

ITU-T Telecommunication standardisation sector of the ITU.

L

LAN Local Area Network.

M

multicast Communication between a sender and multiple receivers.

P

PC Personal Computer.

PTT Post, Telegraph and Telephone.

PPP Point-to-Point Protocol.

PSTN Public Switched Telephone Network, sometimes referred to as GSTN — General Switched Telephone Network.

R

RR Receiver report — an RTCP packet.

RTCP RTP Control Protocol.

RTP Real-time Transport Protocol.

S

SDES Source description — an RTCP packet.

SDP Session Description Protocol.

SIP Session Initiation Protocol.

SLIP Serial Line Internet Protocol.

SR Sender report — an RTCP packet.

T

TCP Transmission Control Protocol.

TCP/IP The Internet model / stack is often referred to by its two main protocols: TCP and IP.

U

UDP User Datagram Protocol.

unicast Communication between a single sender and receiver.

URI Universal Resource Indicator.

V

VoIP Voice over Internet Protocol also referred to as Internet telephony.

Chapter 1

Introduction

1.1 Background to research problem

1844 was a landmark year with Samuel Morse inventing and patenting the telegraph (Frenzel 1989). The invention of the electric telegraph allowed messages to be transmitted over long distances almost instantaneously. This was neatly illustrated in 1866 with the first successful use of the newly laid transatlantic telegraph cable (Frenzel 1989). A number of inventors began working on improvements to the telegraph. One of these inventors, Alexander Graham Bell, shifted his work from the musical telegraph, a device that may someday 'send as many messages simultaneously over one wire as there are notes on that piano', to a wild idea of sending voice over an electric wire (Casson 1997). In 1876 he succeeded and patented the telephone (Frenzel 1989). Initially the telephone was not well received though it soon caught on. Edison helped the cause, though not from Bell's perspective as Edison was working for Western Union — their competitors, by developing an improved transmitter using a carbon coated disk as one of it's components. The details can be seen in the United States patent number 474230 (Edison 1892). The implications of the patent are described in Dyer (1997). At the start call switching was done manually by switchboard operators (Buchanan 1999). Manual switching services soon became mechanical switches which were better able to cope with the growth of telephone usage. Later digital switching technology would replace the mechanical switch. The telephone system is essentially ubiquitous. It was able to reach this state through government intervention and standardization bodies. For instance in the Unites States, American Telephone and Telegraph (AT&T) was created to provide affordable telephony services throughout the country (Comer 1995). The International Telecommunication Union (ITU) provides standards for the telecommunication industry (Tanenbaum 1996).

In 1958 the Advanced Research Projects Agency (ARPA), now known as DARPA

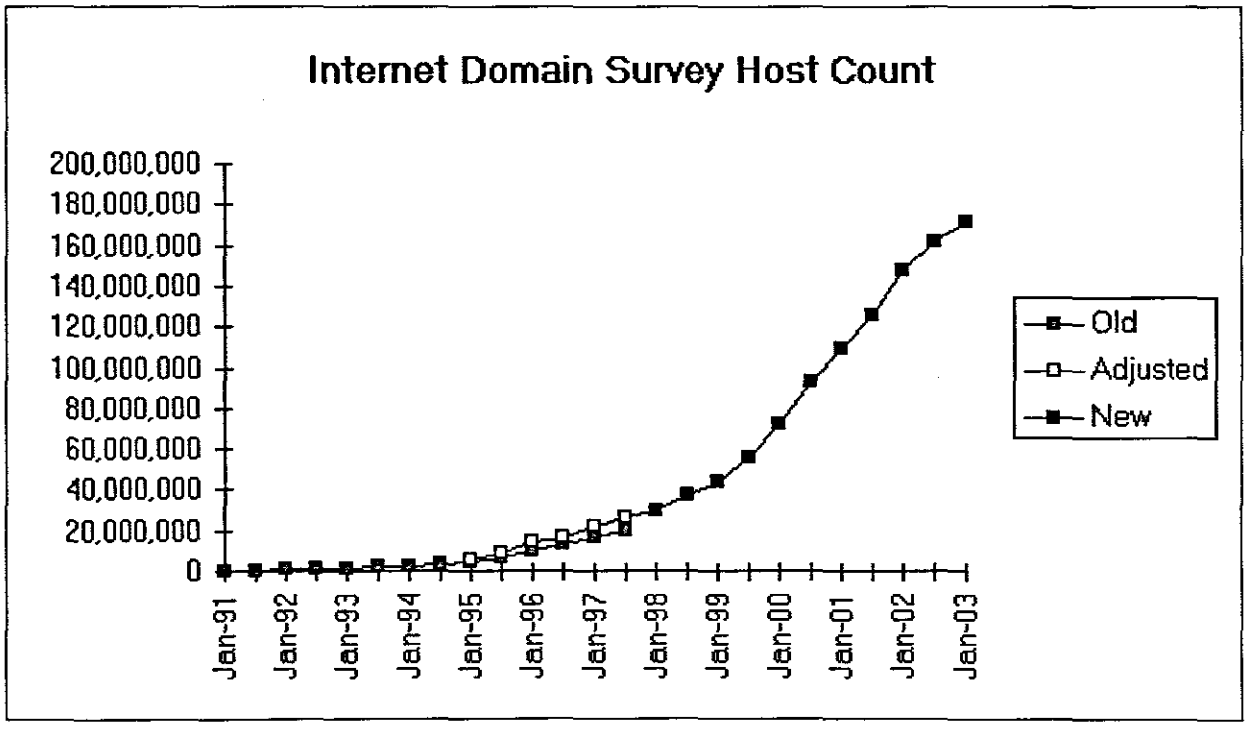


Figure 1.1: The number of Internet hosts counted in surveys done by the Internet Software Consortium (ISC). More information on the methods used and what classifies a host can be found at the ISC website. Source: Internet Software Consortium (www.isc.org) (ISC 2003).

(DARPA n.d.) (Defence Advanced Research Projects agency), was formed (Tanenbaum 1996). This was in response to the launch of Sputnik by the Soviet Union (Tanenbaum 1996). ARPA would initiate a project which would be the start of what we know today as the Internet. In 1967 a plan was published to network computers at ARPA funded research centres. This network, using packet switching technology, would be known as ARPANET (Roberts 1984). The ARPANET grew rapidly and was connected to other networks. Along with this growth was the development of the TCP/IP model and protocols (Tanenbaum 1996, Cerf & Kahn 1984, Postel, Sunshine & Cohen 1984). In 1990 ARPANET was shut down (Tanenbaum 1996) though the Internet continued to grow at an incredible rate (ISC 2003). Figure 1.1 gives an indication of the growth of the Internet. More information on the surveys can be found at (ISC 2003). Section 3.5 in RFC 1035 gives a good explanation of the IN-ADDR.ARPA domain mentioned in explanation of the new survey (Mockapetris 1987). With the expansion of the Internet, services such as e-mail have become an integral part of many people's lives.

Recently there has been much interest in transmitting voice over the Internet and other data networks. This is not a new concept and has been implemented since the early 1970s (Cohen 1976). This interest is most likely due to the vast number of people with access to the Internet and the increase in user access speed. Ethernet LANs have become commonplace and Internet dial-up modems typically offer 56Kbps access. The introduction and permeation of VoIP related standards is another factor influencing the growth of

the industry. Standards are typically of no real concern to the average end user although users may find that they require a collection of software in order to communicate with others. Standardization becomes an issue with network implementers, those concerned with interoperability, multi-vendor support, large scale deployment and expansion.

Transmitting voice over the Internet and other networks using Internet protocols is usually referred to as Voice over IP, VoIP for short, or Internet telephony. Voice over Internet Protocol became a reality for many Internet users in 1995 when Vocaltec released it's Internet Phone software (IEC n.d.). This software allowed telephone-like communication between users provided they both ran the Internet Phone program. The main advantage of VoIP at that time — and still today — is cost. Probably the most obvious example of this is dial up Internet users being able to speak to friends and family anywhere in the world, provided they have a computer and an Internet connection, for the cost of a call to their local ISP. PC to telephone calls can be accomplished with gateways, which translate between the Internet protocols and the public switched telephone network (PSTN) signals. Businesses can save money with VoIP as they only need one set of cables and equipment for both their voice and data networks. Transmitting voice over data lines between premises can significantly reduce costs. VoIP has the ability to add a number of services in addition to those offered by the PSTN.

- Media quality is adjustable. End points can switch between codecs during calls based on information gathered about the network or to allow for differing quality media to be sent.
- VoIP systems need not be limited to transporting voice alone. Video, high quality audio and other media can be sent between the end points (Schulzrinne & Rosenberg 1998).
- Internet telephony can provide for secure communications. Signalling can be encrypted and authenticated. Data transferred between end points can also be encrypted (Schulzrinne & Rosenberg 1998).
- Internet telephony can be combined with a presence system. 'Presence is a means for finding, retrieving, and subscribing to changes in the presence information (e.g. "online" or "offline") of other users' (Day, Aggarwal, Mohr & Vincent 2000). Therefore users will be able to determine whether the person they are trying to contact is available before placing a call.
- Convergence is a main selling point of VoIP. As mentioned in previous points, audio can easily be combined with other technologies and services, like presence, instant messaging, video, whiteboards, file sharing, web accessibility, etc. to provide an improved and flexible communication experience for the user.

- When there is no audio to be transmitted, no packets need be sent. This is known as silence suppression. Silence suppression is handled by the end points and makes more efficient use of the available bandwidth (Schulzrinne & Rosenberg 1998).

These are some of the numerous advantages of switching to VoIP but for now we'll concentrate on cost as this was a significant factor in deciding on this project.

In South Africa, telephone services are provided by a single company: Telkom. Local, long distance and international calls all go through Telkom's infrastructure. There is no competition and therefore no driving force to reduce costs and improve service unlike the situation in the United States where there are 1500 privately owned telephone companies (Tanenbaum 1996). Long distance calls within the United States have dropped to 5c (US) a minute (Rosenberg 2000). Domestic long distance calls in South Africa cost 1,65c (ZA) per second including VAT during standard time (Telkom 2003*a*). This works out to 99c (ZA) a minute. A call to the United Kingdom from South Africa will cost R3,35 a minute during standard time (Telkom 2003*b*) while MCI (MCI 2003) and AT&T (AT&T 2003) offer 7c (US)[1] and 8c (US)[2] a minute respectively. The cost per minute of a call to the United Kingdom described above is illustrated by table 1.1 with units in ZA Rands. Rates have been converted to Rands using XE.com's Universal Currency Converter® at 11:20:44 GMT on the 14/01/2004 (XE.com 2004). $1 (US) is equal to R7,0897 (ZA).

Company	Cost per minute of call to UK in ZAR
Telkom (South Africa)	R 3,35
MCI (USA)	R 0.50
AT&T (USA)	R 0.57

Table 1.1: Comparison of international telephone rates from three telecommunications operators for calls to the UK.

Many people make use of PC-based Internet phones to communicate with their family and friends overseas to avoid paying the long distance toll. There are a number of drawbacks to the systems being used. The system requires a computer, which is perhaps an unnecessary expense if the user wishes to only use a single application (i.e. an Internet phone). With the computer comes the requirement of at least limited computer literacy. In many cases the software must be downloaded, installed and configured before use — a daunting task to someone new to computers. The user communicates by talking into a microphone and listens to speakers attached to the computer's sound card. A headset could replace the microphone and speakers, though this would probably entail unplugging speakers each time the user chose to use the Internet phone software. An alternative

[1]MCI Anytime Worldwide (with MCI nationwide)
[2]AT&T AnyHour Advantage Plan with City Savings

to this is a micro-controller based standalone Internet phone with a keypad and display interface similar to many PSTN / cellular telephones. This unit would not require a computer to access the Internet or perform any functions required to function as an Internet phone. This development would have a number of advantages over the PC based Internet phone. The telephone-like interface would be familiar to most users. The cost would be significantly less than that of a computer and the required software. There is no need to locate, download or install software before use. The main disadvantage of the proposed system would be the limitations imposed by the hardware, e.g. limited memory. It was envisioned the unit would have a built-in voice modem allowing the user to replace their PSTN phone with the Internet phone and use it for both PSTN and VoIP calls.

1.2 Objectives

The objectives of this project are to:

1. Research the protocols required to send voice over an IP network.
2. Research the hardware and software required to transmit voice over an IP network.
3. Develop a micro-controller based unit capable of sending voice over an IP network.
4. Build two units to illustrate the units' usage and perform analysis on the designed system.

> 'In short, packet switching seems ideally suited to both voice and data transmissions. The transition to packet switching for the public data network has taken a decade, and is still not complete; many PTT's and carriers have not accepted its viability. Given the huge fixed investment in voice equipment in place today, the transition to voice switching may be considerably slower and more difficult. There is no way, however, to stop it from happening.' (Roberts 1984)

Chapter 2

Voice over IP

2.1 Components of VoIP

Voice over IP, or Internet telephony, is the term used to describe the use of packet switched networks — in particular IP networks — for transporting voice; a job traditionally performed by circuit switched networks. VoIP is not limited to sending voice alone but can be used for sending video, as in video conferencing, various different audio encodings and qualities, etc.

In order for a voice over IP system to work it requires at least two end points, e.g. IP phones. Typically there will be a number of intermediary nodes and any number of end points. In order for these end points to operate they require at least the following three components:

- A method of signalling — between the end points (and users).
- A means for transferring real-time data.
- An audio component including an audio codec.

The intermediary nodes may require one or more of the above-mentioned components depending on their role in the system.

Functionality can be expanded by including other components in the end points and intermediary nodes. Below is a short list of services that can be added by including more components and specialised nodes:

- Authentication and location services — to assist in the location of end points —

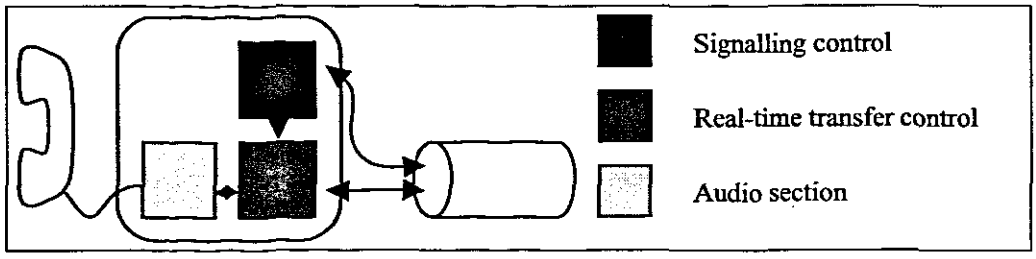


Figure 2.1: The core components of a VoIP end point.

can be provided by proxy servers (Rosenberg, Schulzrinne, Camarillo, Johnston, Peterson, Sparks, Handley & Schooler 2002) or gatekeepers (Jones 2000).

- The interconnecting of networks using gateways. Gateways are devices that allow the interconnection of networks with different specifications, for example connecting a VoIP system to the PSTN to allow PC to telephone communication. Two major standards have arisen in Internet Telephony. Gateways allow interoperability between the two systems.
- Users can be notified of other users' online status, etc. by using presence servers.
- Real-time data can be combined from multiple sources into a single stream by using real-time transport protocol (RTP) mixers (Schulzrinne, Casner, Frederick & Jacobson 2003).
- Accounting services required for billing and administrative purposes can be implemented by including technology such as remote authentication dial in user service (RADIUS) accounting (Rigney 2000).
- Real-time video capabilities and shared whiteboards can be implemented to aid in communication and the transfer of information.

2.2 The operation of VoIP

In order for a VoIP system to work, the end points and nodes need to understand each other either by using the same specifications or via intermediary nodes like gateways. Most implementations will use a common set of protocols and media encodings for the VoIP system to work.

Two open, standardised approaches have emerged for the implementation of Internet telephony. These two approaches are known by their core documents: the session initiation protocol (SIP) (Rosenberg et al. 2002) and H.323 (Jones 2000). They were developed by the Internet Engineering Task Force (IETF) (IETF 2004) and the International Telecommunication Union (ITU) respectively (ITU 2004).

H.323 was developed first; it was approved in November 1996 (Databeam 1998), and is widely implemented. The session initiation protocol (SIP) was first standardized in March 1999 and has since been replaced by RFC 3261 and accompanying documents in June 2002. The most recent version of H.323 is version 5 approved in May of last year (Packetizer 2003*a*).

H.323 is an umbrella recommendation that covers a selection of specifications required for VoIP and multimedia conferencing. SIP is just a signalling protocol and relies on other protocols chosen by the implementor. SIP's only requirement at the application layer is that all user agents at least support the session description protocol (SDP) for describing sessions. H.323 makes use of the Real-time Transport Protocol (RTP) for real-time data transfers. A SIP implementation would typically also use RTP. All H.323 end points must support the G.711 audio codec. Other codecs are optional. SDP allows SIP users to negotiate for the use of whichever codecs they may support.

2.2.1 Session initiation protocol

SIP can be used to initialise, modify and terminate multimedia sessions. SIP can be used with various IETF protocols to provide the required services. For a two user basic Internet telephony application, the session description protocol (SDP) would be required to negotiate the media attributes of the session. The real-time transport protocol (RTP) would be used to transfer the voice between the end points. The session initiation protocol specifies a number of elements, these are:

- user agent clients and servers;
- proxy servers, both stateful and stateless;
- and registrars.

Internet telephones will contain both a user agent client and a user agent server. The type used will depend on whether the user agent is sending requests (client) or responding to requests (server). User agents can switch between being a client or server on a transaction by transaction basis. For example a called user's Internet phone will operate as a user agent server to process the INVITE request and send responses to initialise the call. The same user may decide to terminate the call at some point, in which case the user's Internet phone will operate as a user agent client to send the BYE message which will end the call.

SIP proxy servers work on behalf of user agents and assist in routing messages towards their destination. Proxies can be used to enforce policies and for security reasons. Proxies

will act as both a client and server to receive messages from a SIP entity and forward messages to other SIP entities. Registrars are specialised user agent servers which update location service data received from user agents in REGISTER requests.

Usually SIP proxy servers will route messages between SIP end points. Proxy servers will provide location services to enable correct routing. Typically a SIP session will take place in what is known as a 'SIP trapezoid'. A caller from one domain will attempt to contact a callee in another domain. The caller will send an INVITE for the callee to a proxy in the caller's domain. The proxy will attempt to locate and forward the message to the callee or a proxy server closer to the callee. In the case of a classic SIP trapezoid this will be the proxy in the callee's domain — possibly through other proxies — which will forward the INVITE to the callee. As an invite can take some time to complete, the servers will send back provisional responses to indicate that the transaction is still active. Typically the callee will send a RINGING provisional response to inform the caller that the phone is ringing on the callee's side. If the callee answers the call an OK response is sent to the caller via the proxies. This ends the INVITE transaction. As the location of the callee is made known during the INVITE transaction the acknowledge request can be sent directly to the callee unless a proxy wishes to remain part of the route. During the INVITE transaction the session description protocol (SDP) is used to determine the attributes of the multimedia session to be established. The session can be modified during the call by either party by sending a reINVITE with an SDP body. Whichever party sends a reINVITE will act as a user agent client regardless of its previous role. A call is terminated when one of the user agents sends a BYE request. The other user agent will respond with a final response ending the session.

2.2.2 H.323

H.323 covers multimedia communication over packet based networks that don't necessarily provide a guaranteed quality of service. The H.323 recommendation itself describes the components of a multimedia system, e.g. terminals, gatekeepers and multipoint control units. Terminals would be somewhat equivalent to SIP user agents and gatekeepers to SIP proxies. Gatekeepers are not mandatory components of H.323. Endpoints (terminals, gateways, and multipoint control units) use registration, admission, and status (RAS) messages to communicate with the gatekeeper in their zone. There may only be one gatekeeper per zone. H.323 uses three message types:

- H.245 for control signalling and capabilities exchange.
- Q.931 as described in H.225.0 for call control signalling.
- RAS.

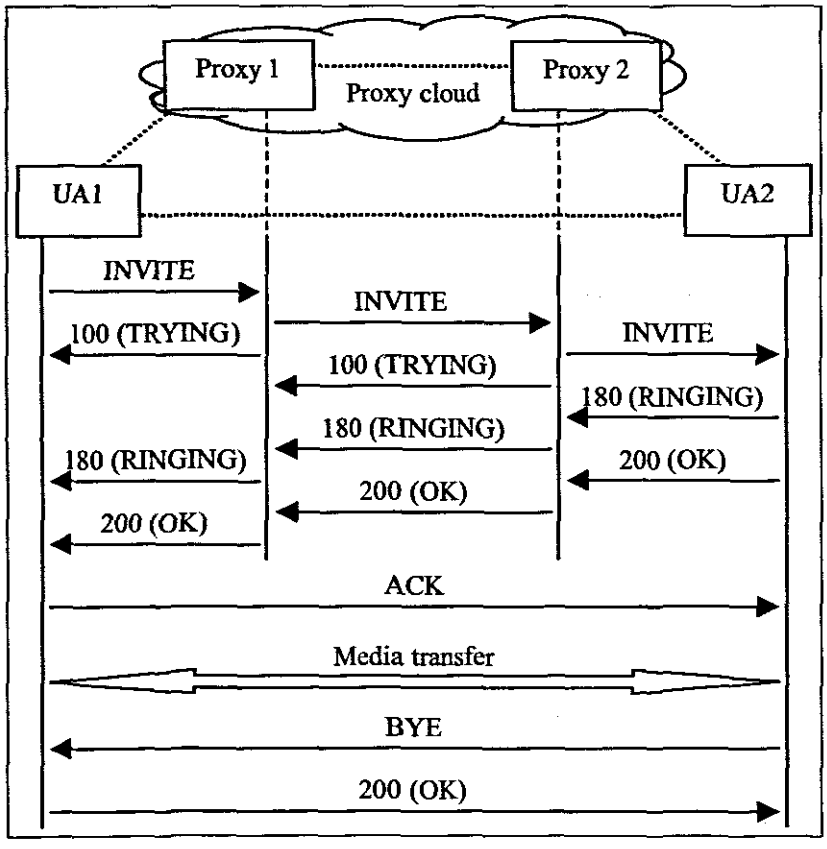


Figure 2.2: SIP trapezoid example with UA1 contacting UA2 and UA2 terminating the call after media transfer has taken place. Sourced from (Rosenberg et al. 2002).

The relationship between the protocols can be seen in figure 2.3. Figure 2.4 illustrates the establishment of an H.323 call between endpoints (EP) that are in different zones. In the figure both gatekeepers (GK) allow for direct call signalling, that is the call signalling is sent directly between the endpoints and not routed via the gatekeepers. RAS messages are represented by three letter acronyms. Message names tend to be self explanatory. First endpoint one (EP1) will send an AdmissionRequest (ARQ) message to the gatekeeper controlling it's zone (GK_A). The gatekeeper will forward a LocationRequest (LRQ) to the gatekeeper of zone B (GK_B), the zone in which endpoint two (EP2) resides. GK_B responds with a LocationConfirm (ACF) and GK_A sends an AdmissionConfirm (ACF) to EP1. As the gatekeepers have allowed for direct call signalling, EP1 will attempt to set up the call (using H.225.0 — Q.931) with EP2 directly. EP2 requests permission sending an AdmissionRequest (ARQ) to GK_B. GK_B replies with an AdmissionConfirm (ACF). EP2 will indicate to EP1 that the connection has been established. H.245 capabilities exchange and master and slave determination takes place. The H.245 OpenLogicalChannel procedures are followed to open channels for transferring the media streams.

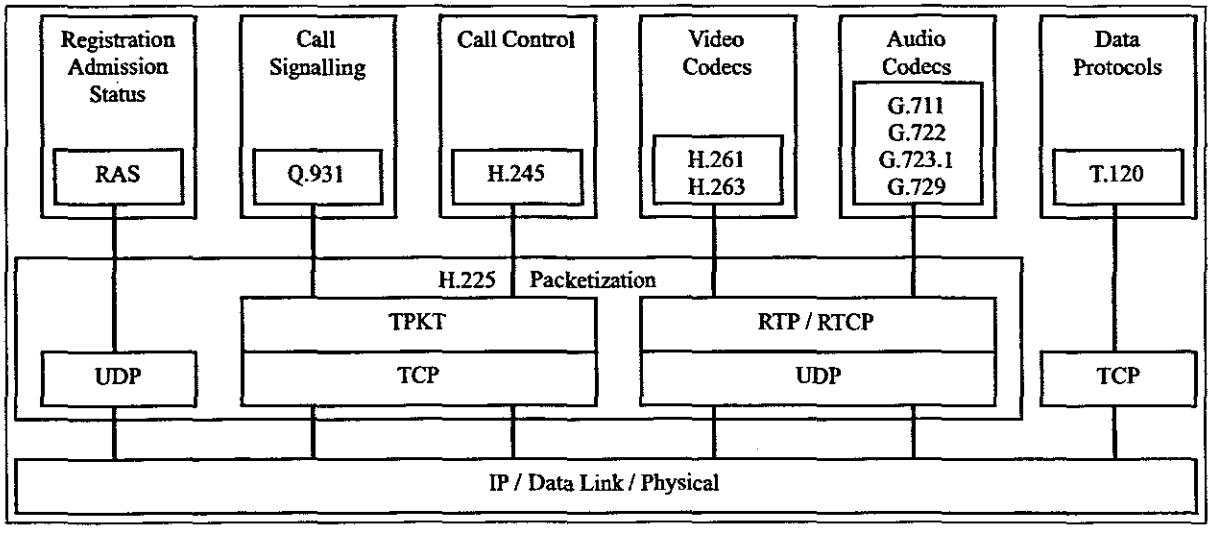


Figure 2.3: H.323 protocol stack. Sourced from (Schlatter 2003)

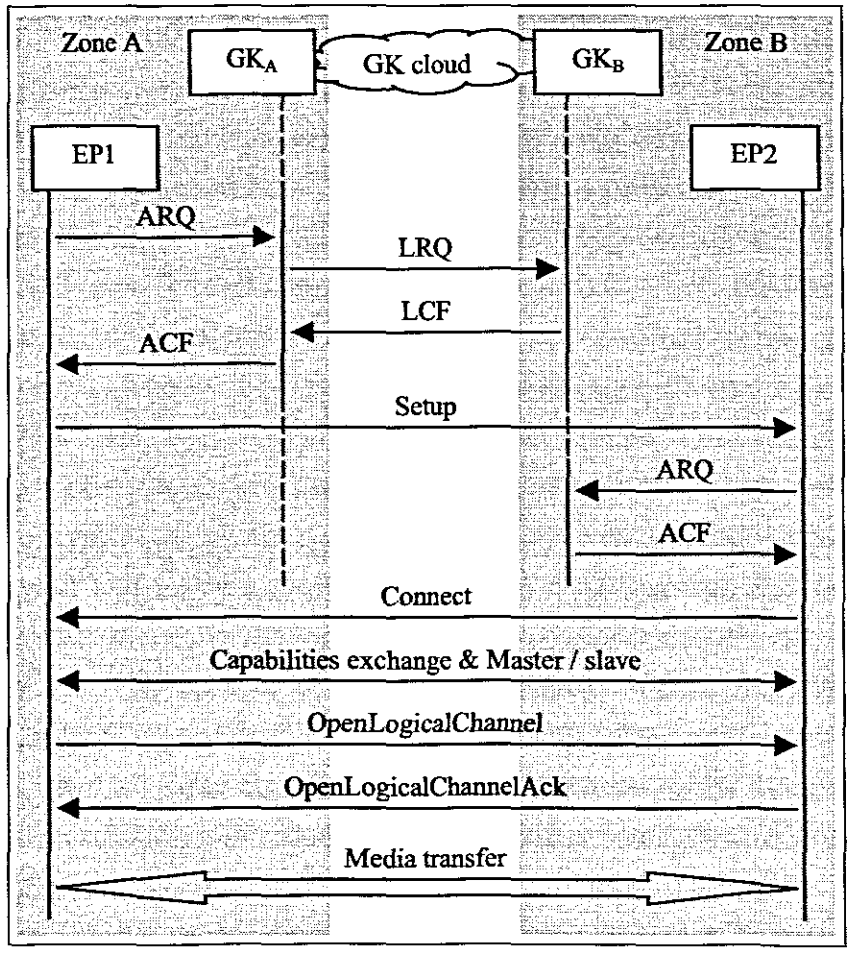


Figure 2.4: An example of the procedures required to establish media transfer using H.323. Sourced from (Schlatter 2003).

2.2.3 Real-time transport protocol

The real-time transport protocol (RTP) is an IETF specification for transferring real-time data over a network. RTP has an accompanying control protocol called the RTP control protocol (RTCP). RTP has been designed to handle user groups of any size. RTP does not provide resource reservation or guarantee any quality of service. Other protocols such as the resource reservation protocol (RSVP) (Braden, Zhang, Berson, Herzog & Jamin 1997) can be used with RTP to enhance quality of service. RTP has been designed to carry current as well as as-yet-undeveloped payloads. RTP is altered somewhat depending on the payload it is carrying, therefore the protocol is split into multiple documents: The core protocol RFC 3550 (Schulzrinne et al. 2003) and profile and payload format documents. The most well known of these is RFC 3551 (Schulzrinne & Casner 2003).

Each RTP packet contains a timestamp and sequence number to assist in reconstructing the stream at the receiver. The participants in each media stream will be distinguished by an RTP level address known as a synchronisation source (SSRC). This SSRC must be unique within the RTP session. If the participants are communicating with more than one stream within the RTP session then the sources of each stream will have different SSRCs. The RTCP provides a canonical name that can be used to link the SSRCs from a particular implementation. RTCP sender reports provide a wallclock time and the equivalent RTP timestamp for each source from the sender to enable synchronisation of the source streams at the receiver. RTCP provides data that is useful for determining the quality of the RTP session.

2.2.4 Codecs

Real-time media such as audio and video typically require a fair amount of bandwidth. For instance a full duplex uncompressed telephone quality (narrowband) call, that is 16-bit resolution sampled at 8KHz, will require that each side of the conversation be capable of sending and receiving 128Kbps (VoiceAge 2000) excluding the overhead of the packets in which the data will be placed. In order to conserve bandwidth, compression and encoding techniques are used. This is especially important for dial-up Internet users. Typically telephone quality voice codecs are in the range 64Kbps (ITU-T 1993) to around 5,3Kbps (ITU-T 1996*b*). More heavily compressed voice codecs are available but as the codecs are lossy the quality deteriorates with the decrease in bandwidth requirements. Another useful tool for decreasing bandwidth usage is silence suppression. If enabled it allows media transmission to stop until audio is detected again. Often comfort noise will be generated at the receiver during periods of silence to simulate background noise at the sender. Table 2.1 shows the most prominent of the voice codecs researched and their

Voice codec	Developer	Bandwidth	Reference
PureVoice	Qualcomm	6,8Kbps on average	(Qualcomm 1997)
G.723.1	ITU-T	5,3Kbps or 6,3Kbps	(ITU-T 1996*b*)
G.729	ITU-T	8Kbps	(ITU-T 1996*a*)

Table 2.1: Selection of voice codecs researched and their bandwidth requirements.

required bandwidths.

Chapter 3

Design of a VoIP system

3.1 Overview of proposed system

The system is designed to mimic the operation (from the user's perspective) of the PSTN. The user will be able to place or receive calls in a similar manner to that used in the PSTN. Two mostly identical units (end points) will be created between which data will be transferred. Each unit will consist of a micro-controller, which will form the core of the unit. A voice codec will be required to do the audio encoding and decoding. A means for connecting the unit to an IP network is required. Various pieces of software are required to control the unit and allow communication to take place. Two user interfaces are required, one digital and the other analogue. The digital interface, taking the form of a keypad and display, will allow the user to control and configure the system. The analogue interface fulfils the role of the handset in a PSTN phone.

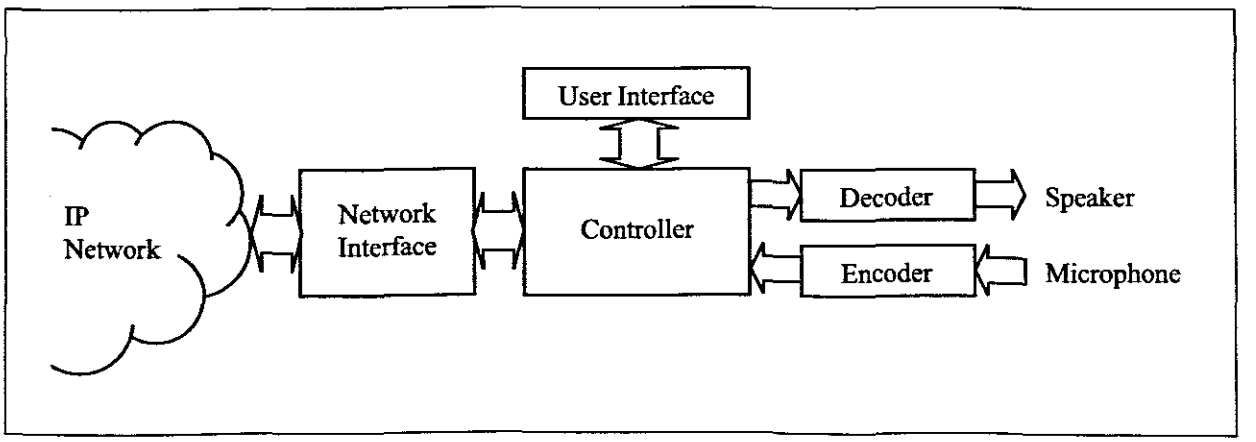


Figure 3.1: Block diagram of proposed unit. Two units will be used to allow communication to take place.

3.2 Components Required

The main components required to produce a VoIP unit are:

- A micro-controller, which will control the system.
- A voice codec to do the encoding and decoding of the audio signal.
- A real-time operating system (RTOS) for the micro-controller.
- A TCP/IP stack for the micro-controller.
- Development tools for the micro-controller.
- A protocol or suite of protocols to perform the signalling and media transportation.

Whilst the real-time operating system is not strictly necessary, it was felt that having one would be beneficial.

3.3 Evaluation of technology

There were numerous components that could be used in this development. The initial research evaluated which would be used in the project. Various factors had to be considered before making a decision as to which components to use. Cost was typically the main factor with all components. Other factors included; availability, support, documentation, issues regarding development tools, and complexity.

The micro-controller is the main hardware component and its choice was directly related to other components required, these are (1) the real-time operating system, (2) the TCP/IP stack, and (3) the development tools.

Individual components as well as modules, e.g. development boards, were considered. In the end two modules remained on the list for the controller section: the μCsimm from Lineo (Lineo 2001) and the AWC86 Slim-Link® Web Server-Controller (AWC86) made by Xecom (Xecom 2003).

Both modules contained a micro-controller, RAM, ROM, an Ethernet controller and came bundled with a free operating system and TCP/IP stack. Development tools for both were readily available. Below is a comparison of the two modules.

Criteria	μCsimm	AWC86
Micro-controller	Motorola Dragonball	AMD AM186ES
RAM	8MB	512KB
ROM	2MB	512KB
Operating system	μCLinux (free, open source)	MicroRTOS (free)
Internet stack	Yes	Yes
Network interfaces	10base-T Serial ports available for a modem	10base-T Serial ports available for a modem
Package	Components mounted on PCB which can be plugged into a 30 pin SIMM socket	52 pin module containing all components.
Development tools	GCC, GNU tools Tools are supplied with the development kit.	Borland Turbo C++ versions 3.0, 3.1, 4.5 and 4.52 Microsoft Visual C++ Versions 1.0 to 1.52 Target specific tools supplied Compiler must be purchased separately.
Cost	\$250 — academic kit \$300 — regular kit	\$159 — AWC86 \$199 — AWC86A
Notes	While μClinux is not a real-time operating system, its multi-tasking system would suffice. RTLinux, a hard real-time, implementation of Linux is available.	The AWC86A includes a real-time clock and replaces the AWC86's 8 digital inputs with 8 analogue inputs. It also features 2 analogue outputs. Development boards are available from Xecom. Xecom can supply Borland Turbo C++ 4.52 to users who have purchased Borland C++ Builder 5 (AWC 2001).

Table 3.1: Comparison of the μCsimm and AWC86.

Note that Lineo and the commercial release of the μCsimm don't seem to exist anymore. Arcturus Networks is now the supplier of μClinux based development kits (Networks 2001). A common problem in intellectual property development is that components have very short development life-cycles and most commercial development will have access to pre-production component technologies for inclusion in new designs. This makes it difficult for individuals or small organisations to compete in cutting edge imple-

mentation. Product and company life-cycles are often very short due to the risks inherent in investing in new technology. Due to rapid development and the dynamic nature of technology development bounds must be set for the research of components.

The Cape Technikon already had Borland Turbo C++ 4.52 and the AWC86 can be programmed using a terminal emulator such as Hyperterminal[1] and a serial cable. Programs can be loaded via the file transfer protocol (FTP) once the operating system has been loaded and the MicroRTOS FTP server started. Therefore the cost of development tools for the AWC86 was negligible. It was felt that the μCsimm was overkill and that the project could easily be implemented on the more limited AWC86. It was also perceived to be more difficult to learn to program for Linux than to use the simpler MicroRTOS.

3.4 System selected

The core of the initially selected system is the AWC86 Slim-Link® Web Server-Controller from Xecom. The AWC86 appeared to meet the requirements for implementing the Internet phone controller. The AWC86 contains a 40MHz 16-bit micro-controller, 512KB Flash memory and 512KB RAM. A large number of I/O pins allow for the connection of codec ICs and other external circuitry. Serial port A, the default console connection, can be used to connect the unit to a computer for programming and user interaction. The computer could later be exchanged for a keypad, display, and a small micro-controller. Two network connection points are provided: (1) An onboard 10base-T Ethernet controller can be used to connect the unit to a local area network, while (2) a second serial port, port B, can be used to connect to a modem for dial up access.

The AWC86's bigger brother, the AWC86A Slim-Link® Web Server-Controller comes with a real-time clock, eight analogue inputs — which replace eight digit inputs on the AWC86 — and two analogue outputs.

A major point in the AWC86's favour is that it comes with a free multi-user, multi-tasking, pre-emptive real-time operating system. The operating system includes a TCP/IP stack. Xecom's offering therefore met the three main requirements for the project; a suitable micro-controller, a real-time operating system, and a TCP/IP stack. The module is available at a price lower than many of the alternatives. Documentation is available in Adobe Portable Document Format (PDF) and sample code is provided with the operating system. Two AWC86 modules and an AWC86A were purchased.

Prior to the controller being chosen a decision to use IETF designed protocols was

[1]Hyperterminal comes packaged with Microsoft Windows

taken. There were a number of reasons for this decision. Initially there was a debate as to whether to implement standardised protocols or simply to implement a 'homebrew' software solution. An example of a proprietary VoIP implementation is Skype (Skype 2004). Skype is developed by the people who made KaZaA and uses peer-to-peer technology. The homebrew concept was scrapped in favour of using a standards based solution. It was felt that learning widely used standardised protocols would be beneficial as:

- It would allow the units to be integrated with other systems using the same standards with minimal effort.
- Standards are typically developed by groups of experts over long periods of time and as such should be thoroughly considered and tested.
- It would provide a good grounding for future work in the field both in implementation and further research and development.

There are two major players in the development of protocol suites for VoIP: the ITU and the IETF. The ITU effort is known as H.323 and was introduced before the IETF's session initiation protocol. As mentioned earlier an all IETF based solution was chosen. Two IETF VoIP protocols were developed to run on the Xecom provided TCP/IP stack: the real-time transport protocol (RTP) and the session initiation protocol (SIP). The session description protocol (SDP) would also be developed to describe sessions to be created by the session initiation protocol. They were picked over the ITU suite for the following main reasons:

- Unlike the ITU's documentation, all IETF documentation can be downloaded for free. Draft ITU documents can be downloaded for no charge. Picturetel maintained an archive of H.323 draft documentation that, like the company, doesn't seem to exist any more. Packetizer (Packetizer 2003*b*) has links to draft ITU documentation.
- It was felt that it would be easier to implement and fault find code developed using SIP, the IETF's signalling protocol, as the headers are text based (Schulzrinne & Rosenberg n.d.). H.323 uses ASN.1 to encode information (Schulzrinne & Rosenberg n.d.).
- Whilst both suites essentially perform the same functions they are designed from different standpoints employing different philosophies to solve problems.
- A number of organizations have chosen to support SIP despite the fact that the H.323 suite of protocols is more widely used having been introduced first. The 3rd Generation Partnership Project (3GPP) have chosen to incorporate SIP into their plans (3GPP 2004).

3.5 Discussion

There were a number of components available at the time of selecting the hardware and operating system. The initial cost was a significant factor and the controller chosen met all the requirements including cost. Since then other technologies have become available such as the ETRAX 100LX, a 32-bit processor with 2MB flash, 8MB RAM and an Ethernet controller, among it's features, on a single chip (Axis n.d.). The downside to the product is that it comes in a 256 pin PBGA package. Developing a PCB for it would be a challenge. The amount of open source VoIP related projects has increased. If starting the project now, selecting hardware that can run an open source operating system such as Linux or perhaps NetBSD for which relevant software can be ported would be a priority. This would greatly reduce development time.

Chapter 4

Prototype designs

4.1 Controller board hardware

The first piece of hardware to be designed was the printed circuit board (PCB) for the AWC86. It was developed and in use for some time before additional hardware was required. As with most decisions throughout the course of the project, the decision to design a board as opposed to buying a development board was based on cost.

There were few hardware changes during the development of this project. The majority of code written for this project was written for the AWC86 using the board made by the author.

It was decided to develop the controller board as a general-purpose development board. This would allow for flexibility in developing the prototype. The board could be reused later for other applications. The AWC86 development kit (AWC86DK) schematic was the predominant document consulted while developing the controller board. The AWC86DK schematic can be seen in appendix A.1. The controller board design is logically almost identical to the Xecom design. Only a few minor changes were made to the design.

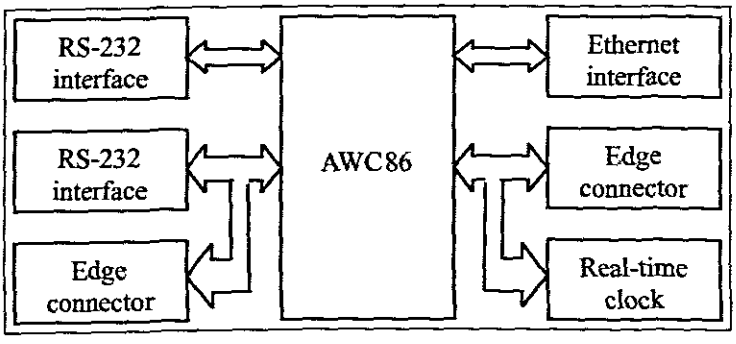


Figure 4.1: Block diagram of controller board.

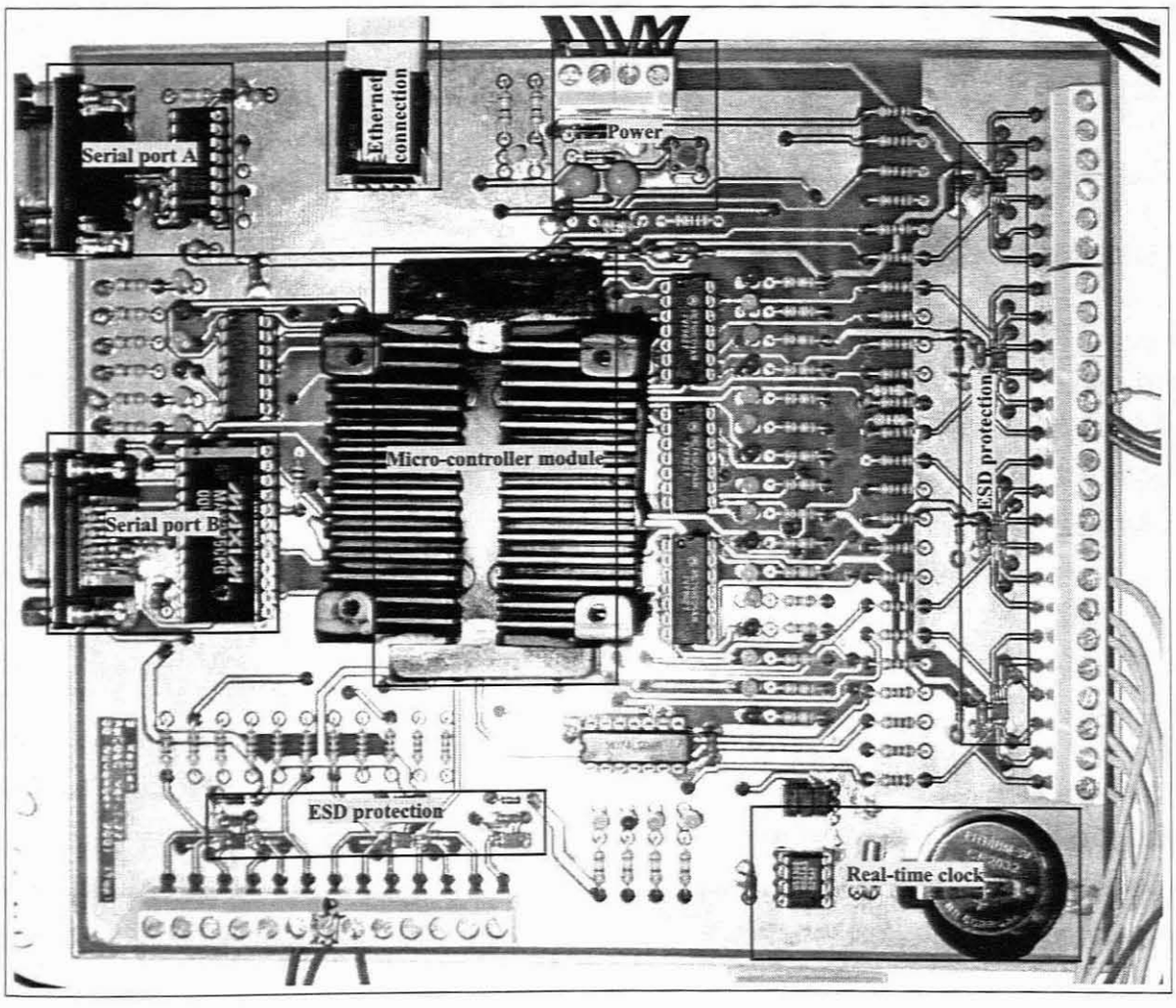


Figure 4.2: Photograph of controller board

The design requirements were:

- 5V operation.
- A socket for the AWC86.
- RS232 level conversion on the two serial ports.
- 10Base-T network connection.
- Current limiting/ESD protection for I/O pins.
- LEDs on I/O ports to indicate activity and to aid in fault finding.
- Design for reusability.
- Optional real-time clock.

The schematic can be seen in appendix A.2.

4.1.1 Design decisions / points

4.1.1.1 Power

The board is designed to operate from a single 5V supply. A two-way PCB mounted screw connector is used in order to connect the board to a power supply. A second two-way connector next to the power input can be used to power external circuits. A 5.1V zener diode is used for over-voltage protection. A schottky diode has been included for additional protection. 100nF (high frequency) and 22μF (low frequency) decoupling capacitors have been placed near the power supply connector. A 100nF decoupling capacitor has been placed close to the positive supply and ground pins of each IC on the board. Undervoltage detectors were considered though not implemented, as a detector would require additional board real estate. The board is to be used in the lab where suitable power supplies are available.

4.1.1.2 Micro-controller

The board is designed around Xecom's AWC86 micro-controller, which is positioned in the centre of the board. The design could be adapted to accommodate the AWC86A's analogue capabilities if they are required for future projects.

4.1.1.3 Serial ports

The primary serial port, port A, is connected via an RS232 level converter, a MAX232E (Maxim 1996), to a DB9 connector. Port A has only transmit and receive pins, i.e. no hardware flow control. Port A is used to connect the micro-controller to a computer for programming or user interaction via the PC. LEDs are used to indicate activity on the port.

Each pin of the secondary serial port, port B, can be connected to either an RS232 level converter IC pin or directly to connectors at the edge of the board by using a combination of jumper settings. Port B includes hardware flow control using the RS232 lines: request to send (RTS) and clear to send (CTS). The AWC86 pins DIO6 – DIO9 can be used for the remainder of the serial interface's control lines. A MAX235 is used as the RS232 interface IC (Maxim 2000). It has five RS232 inputs and five RS232 outputs. The RS232 level pins of the IC are connected to a DB9 connector. It was intended that serial port B be used to attach a modem to enable Internet connectivity over a telephone line. Note that both serial interfaces have been routed for male DB9 connectors.

4.1.1.4 Ethernet connection

The Ethernet controller and all required components are contained within the AWC86 module. Only an RJ45 connector is needed to connect the controller board to a 10Base-T network. LEDs on pins NSTAT and NXMT indicate that the network is active and that data is being transmitted on the local area network (LAN) respectively.

4.1.1.5 Micro-controller I/O pins

All I/O pins, other than the RS232 level shifted serial ports mentioned above, are connected via a 1KΩ series resistor and electrostatic discharge (ESD) protection IC to connectors at the board's edge. The ESD protection ICs used are PACDN006 six channel ESD protection diode arrays from California Micro Devices (Calmicro 2001). Each I/O pin is connected to an LED to indicate activity.

4.1.1.6 Other micro-controller pins

The eight digital inputs and the three independent interrupt pins: the non-maskable interrupt pin (NMI), interrupt one (INT1) and interrupt three (INT3), are connected via 1KΩ series resistors and ESD protection to connectors, as are the I/O pins.

The clock output pin (CLKOUT) and RESETOUT have ESD protection but not the current limiting resistors.

The device can be reset by depressing a push button connected to the $\overline{\text{RESET}}$ pin.

4.1.1.7 Real-time clock

Jumpers on the board can be configured to access a real-time clock on the board. A DS1302 (Dalsemi 2000) from Dallas Semiconductors is used as shown on the AWC86DK schematic. Using a DS1302 allows the user to make use of the MicroRTOS real-time clock functions. A 3V Lithium battery enables the real-time clock to keep time when power has been removed from the board.

4.1.1.8 ESD protection

It was felt that ESD protection was required as a layer of protection due to the cost of the micro-controller modules and the difficulty in replacing them where they to become damaged. ESD protection comes in the form of the PACDN006 ICs. The schottky diode at the power connector offers some protection against negative voltage transients. ESD protected RS232 level conversion ICs were selected though only the MAX232E was available. The unprotected MAX235 is the other level conversion IC used. The postfix 'E' on the Maxim RS232 level converter part numbers denotes that the product has integrated ESD protection. Various ESD protection devices were considered for the I/O pins. The PACDN006 ICs were chosen for their small packages, availability and low cost. The ICs for the first board were supplied for free. Only the shipping charges were required.

4.1.1.9 Indicators

Hex inverters, 74HS04As (Motorola 1996), are used to supply the current for the LEDs[1]. 390Ω resistors are used for current limiting.

4.1.2 Further development

A second controller development board was made. With the second board populated data could be transmitted between the boards.

[1] 74LS04s work equally well.

As mentioned previously the idea was to include a modem to enable the unit to be used with a dial-up Internet account. This would require a point-to-point data link layer protocol, of which there are two for such applications:

- Serial Line Internet Protocol (SLIP) (Romkey 1988).
- Point-to-Point protocol (PPP) (Simpson 1994).

SLIP would be mostly useless outside the lab as it is very simple and was designed before the need for dynamically assigned IP addresses. Point-to-Point protocol supports dynamic IP address assignment, which is one of the reasons that ISPs require their dial-up users to use PPP when connecting to their network. Dynamic IP address assignment is required because of the limited number of Internet Protocol version 4 (IPv4) addresses available and the inefficiency of their use. Version six of the Internet protocol (IPv6) solves this problem by increasing the address size from 32-bit for IPv4 to 128-bit (Tanenbaum 1996). Although Microchip have released an application note detailing the implementation of PPP on a PIC (Loewen 1998) it is far from being RFC compliant. Developing an RFC compliant PPP implementation is a non-trivial task.

MicroRTOS comes in three flavours from version 2.0 upwards:

- one including a PPP client,
- one including a PPP server, and
- one with no PPP capabilities.

Unfortunately these PPP implementations will only work with Xecom modems which are: (1) expensive, (2) only certified for use in America and Japan, and (3) are data fax modems only and could not be used to send audio over the line in the standard PSTN manner. The source for the PPP implementations is also not provided and therefore cannot be modified if required.

Conexant (Conexant 2003) manufacture modem components and a controllered modem with a voice codec from them was considered even though this would mean writing another implementation of PPP for the AWC86. Off the shelf external modems were considered for testing purposes. It was decided to ignore the modem issue until after the application layer software — which is the core of this project — had been developed using Ethernet in place of the modems and PSTN. This would not be a problem provided the design took into account the limitations of the to-be-included modem.

The Technikon's network was used as the medium for transferring data. Two computers were used when communication between the units was required. One computer was attached via the serial port to each unit for programming and user interaction. This configuration would change later with the acquisition of new computers and an unswitched hub.

Each unit was assigned a static IP address to allow communication to take place. Xecom provided a dynamic host configuration protocol (DHCP) client for MicroRTOS version 3 — again without the source — which would allow the units to be dynamically assigned IP addresses when they were connected to the Technikon's network. Unfortunately we were unable to get the DHCP client to work with the Technikon's DHCP server. Even attaching the unit directly to the DHCP server did not work. The client did however work with the HaneWIN DHCP server (Hanewinkel n.d.) which the author set up on his computer (after having disconnected it from the Technikon's network). The unit was connected directly to the new DHCP server's network card. Though nice to have, it was decided to leave the DHCP plan as it was not critical to the operation of the project and the source was not available so it could not be modified to integrate better with the other code or be altered to get it to work with the Technikon's DHCP server.

4.2 Software

With the completion of the controller development board, software development and testing could begin. Initial programs were mostly tests to learn how to work with MicroRTOS and to become more familiar with the C programming language. Initial work was done using MicroRTOS version 2.0, which was supplied before the AWC86 modules arrived. Later a beta version of MicroRTOS 3.0 was used which would be replaced by a more stable version. MicroRTOS provides real-time support using prioritized tasks. MicroRTOS can manage 64 tasks but is limited to 32 by the processor. Tasks are executed at start-up, according to their priority, therefore tasks that are not immediately required will be programmed to suspend themselves at start-up. MicroRTOS uses the tiny memory model, therefore all programs compiled for it are limited to 64KB for both code and data memory.

Initially tests were done using the sample programs and variations on the sample code. The Real-time Transport Protocol (RTP) software was developed before the SIP code and therefore early programming ventures were centred on RTP. The complete Real-time Transport Protocol specification is not available as a single RFC but is split into the core document, with information common to all implementations, and profile specifications. The profile specification chosen depends on the payload carried in RTP.

Word size	Little endian	Big endian
16-bit	0 1	1 0
32-bit	0 1 2 3	3 2 1 0

Table 4.1: Comparison of little and big endian byte order. The bytes are read from left to right with byte 0 representing the least significant byte and byte 3 the most significant byte.

With early programs written for the purpose of learning and without a suitable codec to use, text was the payload used. A simplified version of RFC 2793, RTP payload for text conversation, was used as the profile specification before switching to an audio payload (Hellstrom 2000). One of the first programs could send and receive RTP packets with UDP as the transport protocol. Flow charts for this program are shown in figure 4.3 and 4.4.

At start-up the RTP receive and transmit tasks will initialise and suspend themselves. Once the user command has been activated, the user will be prompted for the destination IP address. The payload type is derived from the Multipurpose Internet Mail Extensions (MIME) type assigned at compile-time. The initial synchronization source (SSRC), sequence number and timestamp values used by RTP are compile-time assigned. The UDP port is opened and mailboxes are initialised to allow data to be passed between tasks. The RTP receive task is then resumed. It will check for packets on the selected UDP port every 50ms. Each time the user presses a key an RTP packet is built and the transmit task is resumed to allow the packet to be sent. The sequence number is then incremented. The 1KHz clock for timestamp incrementing required by RFC 2793 was not implemented. Like the sequence number, the timestamp is simply incremented each time a packet is sent. When a packet is received, it's header is written to the screen in hexadecimal followed by the character payload. The program will exit when the '0' key is pressed. The AWC86 uses little endian byte order whereas RTP requires network byte order (big endian). A comparison of the two byte orders can be seen in table 4.1. No endian conversion was done in this program. Also only ASCII characters were supported and not the full UTF-8 transformation scheme.

Programs were expanded incrementally from that early program. The first addition being endian conversion on the transmit side. Initially no endian conversion was done on the receive side as the packet headers were not being analysed at this stage. The next increment was an expansion of the program structure. Tasks were added for the RTCP components though these were not populated with code for a while. A menu was added to the user command to increase functionality and to aid ease of use.

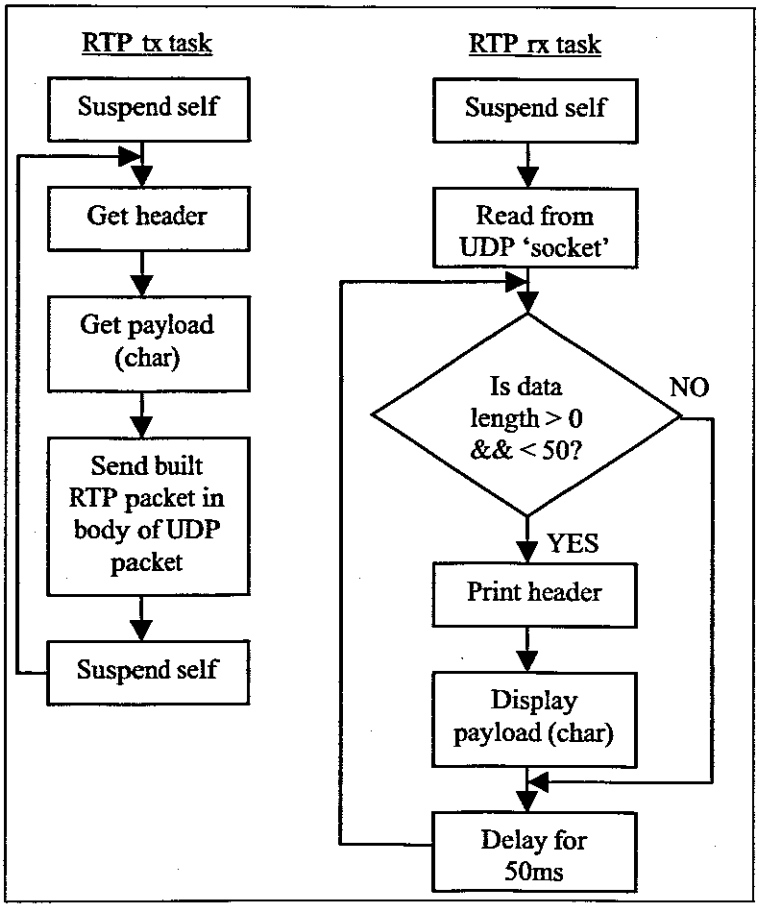


Figure 4.3: Early RTP program flow chart: Tasks.

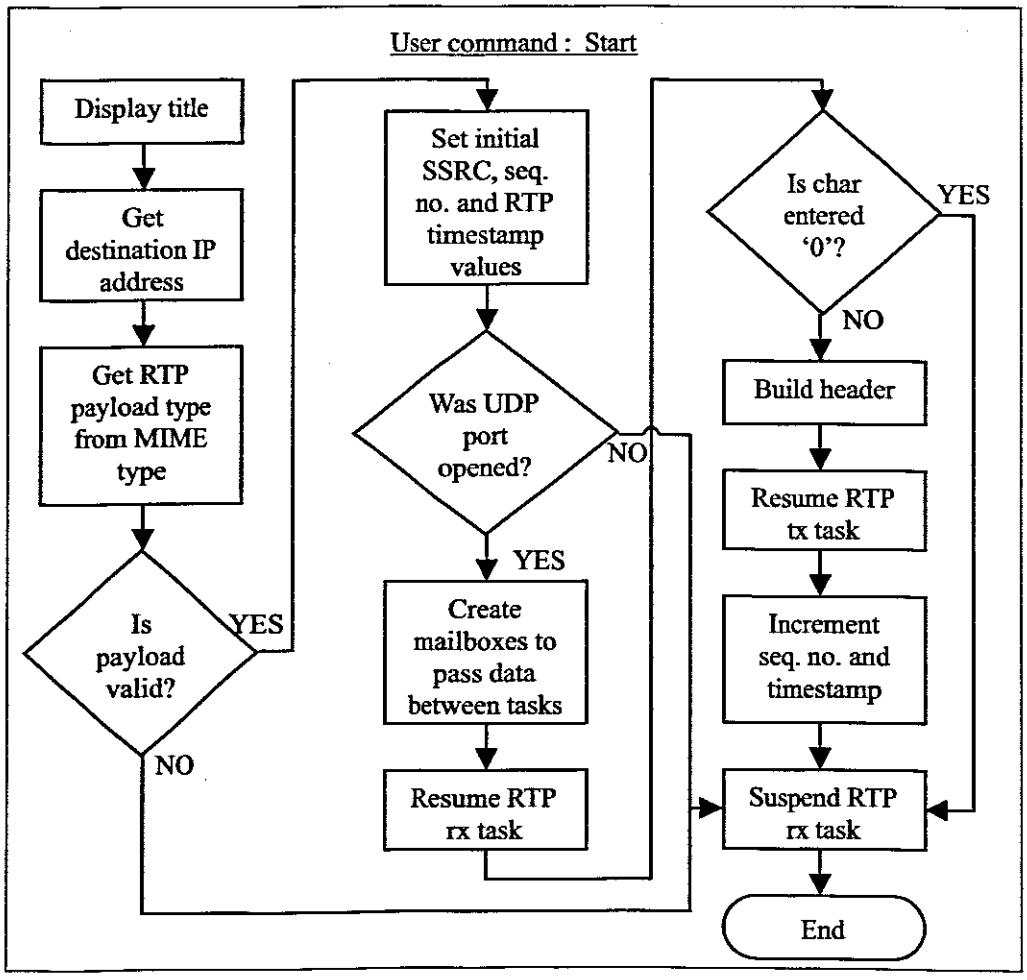


Figure 4.4: Early RTP program flow chart: User command.

The next step was to adapt the program to more closely resemble its intended use — audio transportation. By this time the author had realised that it was infeasible to implement a voice codec on the micro-controller. The processor with its operating system would be too slow to run a reasonably low bit rate voice codec and the remaining phone software. Neither had the author found a suitable hardware voice codec solution. One item that the author had acquired was a Microsoft Windows based G.729 Annex A (G.729A) voice codec application program interface (API). This API was made available for free from VoiceAge for research and development purposes only (VoiceAge n.d.). Though the author would have preferred a G.723.1 codec, G.729A was a close second. G.723.1 is a dual rate speech codec that transmits at 5,3Kb/s or 6,3Kb/s (ITU-T 1996*b*), whereas G.729A has a bit rate of 8Kb/s (ITU-T 1996*a*). Two problems immediately presented themselves: the author was unfamiliar with the methods required to access hardware in Windows, and more importantly the author did not have a 32-bit Windows C compiler. The solution chosen was to develop a DOS C program to emulate an imagined G.729A codec's interface to the micro-controller. The computer's parallel port would be used to transfer the data to and from the AWC86 board.

The program was similar to the one described previously except that the RTP transmission code on the AWC86 is written as an interrupt service routine (ISR) which is activated by a signal from the encoder code in the PC program. The RTP receive code is written as a task that will check the UDP port every 10ms and signal the decoder, which is implemented as an ISR in the PC code, if an RTP packet has arrived. The data transmitted was simply ten bytes with the values 0 through 9. This was to simulate a G.729A frame which is 10 bytes long. The frame received by the decoder interface was 'decoded' to the screen. The handshaking used to communicate between the AWC86 and the codec emulator is shown in figure 4.5. The same handshaking system is used for both reading from the encoder and writing to the decoder. The numbers in the diagram correspond to the text below. The text is written from the perspective of the AWC86.

1. Set data port as an output.
2. Set CODEC_OUT high to generate an interrupt at the decoder.
3. Wait for CODEC_IN to go high signalling that the decoder's port has been configured.
4. Set CODEC_OUT low.
5. Wait for CODEC_IN to go low. This ends the configuration handshaking.
6. Put data byte on the port.
7. Set CODEC_OUT high to signal that data is ready to be read.

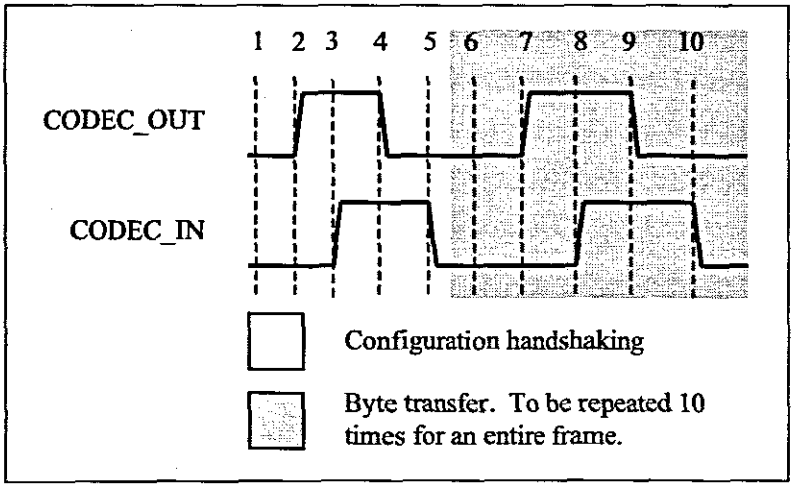


Figure 4.5: This image shows the handshaking used to accomplish writing a 10 byte data frame to the decoder. The same handshaking is used for reading from the encoder. Refer to the text for an explanation of the numbered points.

8. Wait for CODEC_IN to go high after the decoder has read the data.
9. Set CODEC_OUT low to acknowledge that the byte has been read.
10. Wait for the decoder to set CODEC_IN low.

Items 6 – 10 will be repeated ten times to transfer a complete data frame.

After these tests work began on what would become the final version of the RTP implementation. Once RTP was developed, the signalling protocols were added.

Chapter 5

Final design — Implementation

The implementation makes use of standardised Internet protocols for its operation. As such the system can be described in terms of the Internet model. The emphasis of this project is on the application layer. Besides being layered the system can also be split vertically by functionality. There are three areas of functionality. These are:

- Signalling.
- Media transport.
- Quality of service.

Figure 5.1 illustrates the protocols used and their relation to one another. The protocols developed in this project have been highlighted in the image. Solid lines indicate actual interaction in this implementation while dotted lines represent potential or considered interactions.

The session initiation protocol (SIP) was chosen as the signalling protocol. SIP was originally designed only to operate on top of the user datagram protocol (UDP). Later support for the transmission control protocol (TCP) was added. SIP takes the session description protocol (SDP) as its payload for session description and negotiation. The real-time transport protocol (RTP) was chosen as the media transport protocol and uses UDP at the transport layer. In this application RTP is used to transport G.729 Annex A (G.729A) encoded audio frames. The accompanying RTP control protocol (RTCP) is also implemented. All data is transported in Internet protocol version four (IPv4) datagrams. The current implementation and testing uses 10Mbps Ethernet at the data link / physical layer. The system has been designed so that it could operate over a dial up modem link instead of Ethernet.

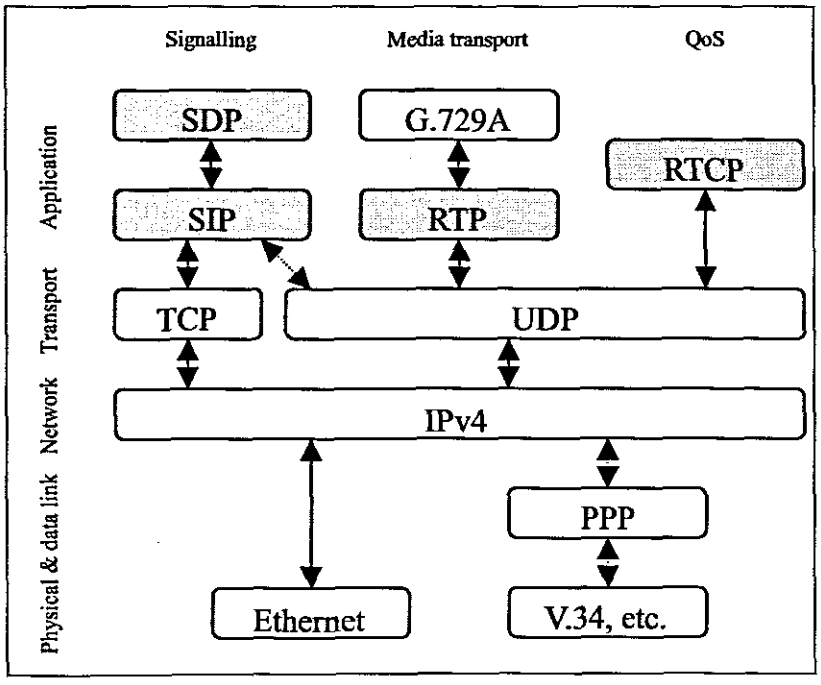


Figure 5.1: Internet telephony protocol stack.

Each of the following three chapters is devoted to the implementation of a column in figure 5.1. Chapter 6 covers the signalling section and explains the implementation of SIP and SDP. Chapter 7 covers the media transport section and explains the choice of the G.729 annex A codec, the development of the hardware and the interface to the controller board and the real-time transport protocol. The RTP implementation is discussed in the chapter. Chapter 8 covers the RTP's control protocol, RTCP, which is useful for quality of service analysis.

Chapter 6

Signalling

The session initiation protocol (SIP) is used to provide the signalling capabilities for the system. SIP is an application layer protocol that provides the means for creating, modifying and terminating sessions. The session description protocol (SDP) is used to describe sessions and media for negotiation and capabilities exchange. SDP is carried in the body of a SIP message.

6.1 SIP

The session initiation protocol was chosen as the signalling protocol. Though research began while the session initiation protocol was defined by RFC 2543 (Handley, Schulzrinne, Schooler & Rosenberg 1999), coding only began after the revised documents had been put on the standards track in June 2002. SIP is now defined in a number of documents with the core document being RFC 3261 (Rosenberg et al. 2002). The following documents were consulted in developing the signalling component of the system:

- RFC 3261, SIP: Session Initiation Protocol (Rosenberg et al. 2002).
- RFC 3263, SIP: Locating SIP Servers (Rosenberg & Schulzrinne 2002*b*).
- RFC 3264, An Offer / Answer Model with the Session Description Protocol (Rosenberg & Schulzrinne 2002*a*).

The procedures in RFC 3263 were not implemented as the final destination IP address was always known to either unit and the implementation has no notion of the domain name system (DNS). SIP has been designed to use both UDP and TCP at the transport layer.

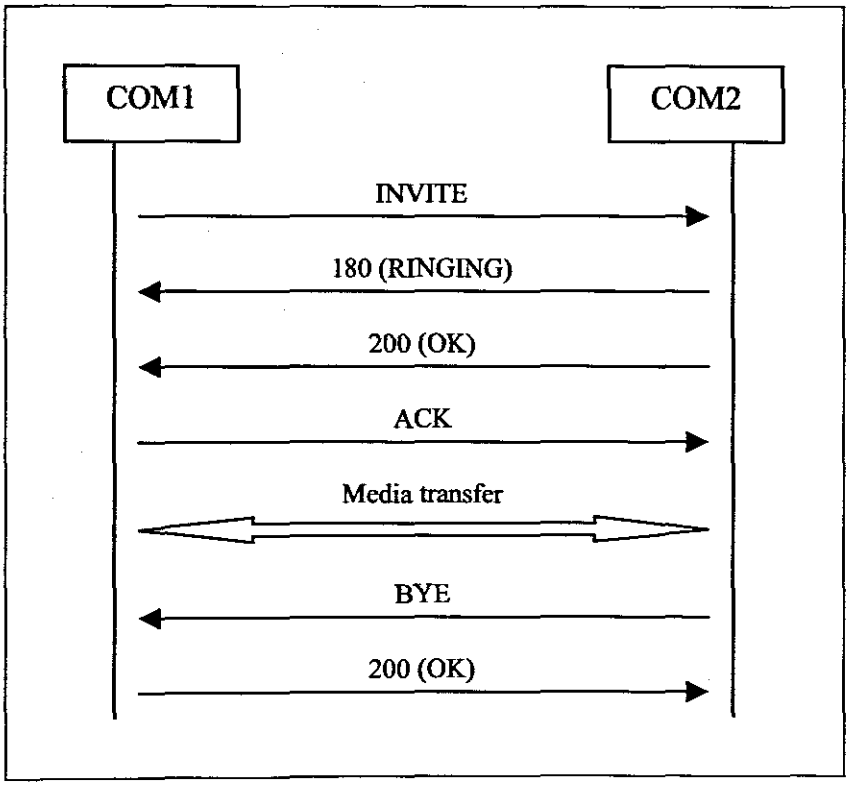


Figure 6.1: Typical sequence of SIP messages.

6.1.1 Functionality

This section will cover the functionality implemented from the user's perspective. The user will be able to place a call, answer a call and end a call. Currently no other functionality has been implemented for reasons that will be explained in the next section. It was never intended that the signalling protocol be implemented in it's entirety but rather that the functionality required for the unit to operate be included.

6.1.2 Implementation

A subset of the SIP specification has been implemented. The reasons for this being concerns regarding the limitations of the controller module and the limited scope of the implementation. The Internet phones have been designed for two user voice only communication. Signalling and media transfer will take place directly between the two units with no intermediary servers. Therefore only user agents have been developed.

A typical sequence of messages can be seen in figure 6.1.

User agent one referred to as Com1 will initiate the call by sending an INVITE with an SDP offer in the body to Com2 (user agent two). In the current implementation SIP URIs and IP addresses are entered manually. Both units listen for TCP packets on the default

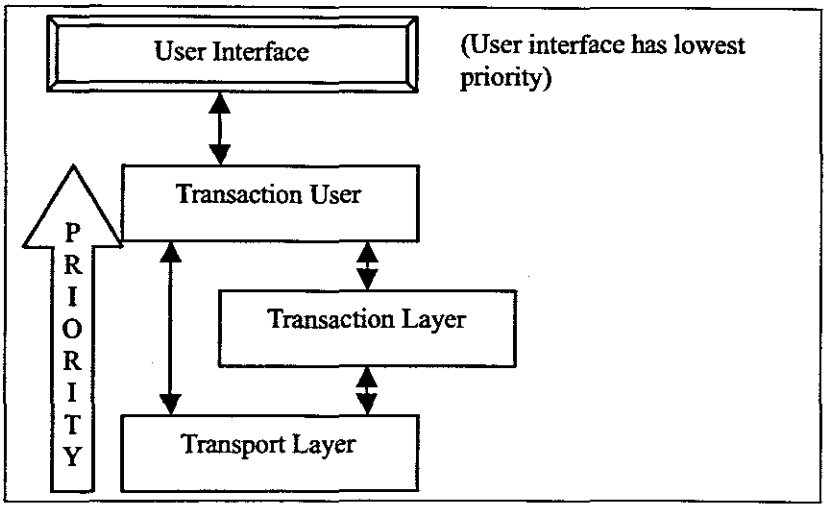


Figure 6.2: SIP implementation with each task representing a SIP layer.

SIP port. On receipt of the INVITE request Com2 will immediately send a 180 response to indicate that the phone is now ringing. When the user answers the call a response with a status code on 200 is sent indicating the setup was successful. The 200 response will contain an SDP answer to complete the offer answer negotiation system. Com1 replies with an acknowledge message to complete the three way handshake. 1xx responses are not counted in the handshaking as they are provisional and do not effect an end of a transaction. The multimedia session will now begin. After some period of time Com2 will decide to terminate the session (hang up). To do this it will send a BYE request. Com1 will reply with a 200 response and the session will be terminated. Note that three SIP transactions took place in the above example: the invite transaction, the session ending transaction, and the acknowledge for the invite is considered it's own transaction.

The implementation has been implemented in layers as described in the specification. The specification considers the syntax and encoding to be the lowest layer but it was decided not to implement this as a layer but rather integrated with the other layers. The implementation is represented by figure 6.2.

Each of the three layers has been implemented as a task with the transaction user with the highest priority and the transport layer at the lowest. The transaction user task communicates with a user command which provides a user interface. The user command has a lower priority than any of the other programmed tasks.

Information is transferred to and from the user via the user interface. The user interface can only communicate with the transaction user (TU). The transaction user can communicate with both the transaction layer and the transport layer as required. The transaction layer can only support one transaction at a time and therefore the CANCEL method — required by the RFC — has not been implemented. This means that the calling party cannot hang up before the callee has answered. However the callee is given a finite amount of time in which to answer the call so the function is not significantly

missed.

The SIP call flow example above can be described in more detail in terms of the layers of which SIP is comprised. After Com1 and Com2 have been started and the initial data entered, Com1's user decides to place a call and enters Com2's contact details using the user interface. Up till this point both units would have the transport layer set to listen on the default port, as a server, for INVITE requests. Both units default to user agent servers at start-up. Com1's user interface will inform the TU that the user wishes to make a call to Com2. The TU will build the INVITE message with an SDP offer in the body and pass it to the transaction layer for transmission. The unit has become a user agent client (UAC). The transaction layer will send the message to the transport layer to be sent. It will then set the transport layer to wait for a response from Com2. Com2's transport layer picks up the INVITE and passes it up to the TU. The TU will generate a 180 (RINGING) response, pass it to the transaction layer and inform the user interface that the phone is now ringing. Com2's transaction layer passes the response to the transport layer to be sent. The transport layer of Com1 will receive the response and pass it up the the transaction layer which will inform the transaction user of receipt of the response. The TU will pass a message to the user interface which will alert the user that the unit called is ringing. When the Com2 user answers the phone, the user interface will inform the TU which will build a positive final response — 200 (OK) — with the SDP answer in the message body. This message will be passed to the transaction layer. The transaction layer will pass this message to the transport layer to be sent. Sending this response will terminate the current transaction. Com1's transport layer receives the 200 response and passes it to the transaction layer. The transaction layer will pass the message to the TU and will terminate the current transaction. The transaction user will generate an acknowledge (ACK) message and pass it directly to the transport layer for transmission. Com2's transport layer will receive the ACK and pass it to the TU. This ends the three way handshaking required for the invite message. Both sides will revert to user agent servers set to listen for SIP requests and will begin their media session. Some time later Com2's user will decide to hang up the phone. The user will do this via the user interface which will inform the TU of the user's decision. Com2 will switch to a UAC and the TU will generate a BYE request. The BYE will be passed to the transaction layer which in turn will pass the message to the transport layer for transmission. The transport layer in Com1 will receive the request and pass it up to the TU. The TU will generate a response. In this case a 200 (OK) response and will pass it to the transaction layer. The transaction layer will relay the message to the transport layer to be sent. As this is a final response, the transaction will terminate, informing the TU of it's termination. The TU will indicate to the user interface that the call has ended. On Com2's side, the transport layer will receive the message and pass it to the transaction layer which will inform the TU of the receipt of the response. The transaction will terminate and indicate this to the TU. The TU will inform the user via the user interface that the call has completed.

RFC 3261 has strict rules regarding in which messages SDP offers and answers may be sent. AN SDP offer may be present in the INVITE if the caller wishes to make the offer, or the 200 (OK) response if the callee makes the offer in an initial INVITE transaction. For the first option, the answer must be in the final positive response, i.e. 200 (OK). The answer may also be present in the preliminary responses. For the second option, the answer must be in the acknowledge following the INVITE transaction. Only the first option is supported by the implementation. It was deemed unnecessary to duplicate the answer in any preliminary responses.

6.1.2.1 interface_CMD

The user interface is implemented as a user command that is started by the user. The user interface communicates between the user and the transaction user (TU) implemented as transaction_user_tsk, which is the top layer of SIP. At start-up all tasks are suspended. Once started the interface will display it's title, enter an infinite loop and present the user with a menu. If exit is selected then the program will break out of the loop. The only other valid option is to begin the VoIP phone program. Firstly message boxes will be created to safely transfer data between tasks. The user will be prompted to enter the local SIP universal resource indicator (URI) and IP address. Various state is initialised for SIP to be activated. The transaction user task is resumed. It will configure the unit as a user agent server (UAS) listening for SIP INVITE messages which indicate that someone is trying to contact the user. The user command will loop checking for input from the user or messages from the TU every 200ms. Their are three valid user inputs: Place call, answer call, and end call. If the user wishes to place a call they will be prompted to enter the destination SIP URI and IP address. The TU task is resumed and informed that the user wishes to place a call. Answering the call and ending the call both resume the TU task with the appropriate message. The messaging system between the user command and the TU task is illustrated by the following code snippet.

```
SIP.tu2interface = BUSY0;          // initialise current message from TU
SIP.lastfromtu = BUSY0;            // and last message from TU
[...]
for(;;)
{
  if(ConHit())                     // if button pressed
  {
  [...]
  }
  else if(SIP.tu2interface != BUSY0) // if message from TU
```

```
    {
      if(SIP.tu2interface == RCVD1XX)  // received provisional response
      {
        if(SIP.t2isubmessage == R1TRYING) // response status code is 100
          printf("trying");
        else if(SIP.t2isubmessage == R1RINGING) // response status code
          printf("RINGING");                    // is 180
        else                                 // other provisional response
          printf("1xx");
        SIP.tu2interface = BUSY0;       // reset TU message
        SIP.lastfromtu = RCVD1XX;       // store last message received
      }
      [...]
    }
  }
```

The user interface accepts seven messages from the transaction user task:

RCVD1XX Indicates that a provisional response (a response with a status code of between 100 and 199) has been received. Two submessages are available — 100 TRYING and 180 RINGING. The user will be notified accordingly. See the previous code snippet.

CALLINPLACE Indicates to the user that a call is already in place.

CANTANSWER Indicates to the user that the call cannot be answered.

ENDSESSION Indicates to the user that the media session and call has ended. The program will break out of the loop (inner loop) where it checks for responses from the user or TU. The user command will loop back (outer loop) to where the user is presented with a menu.

STARTSESSION Indicates to the user that the media session has begun.

CANTENDCALL Indicates to the user that the call cannot be ended at this point.

RINGING Indicates to the user that an INVITE has been received and a 180 RINGING response has been sent.

6.1.2.2 transaction_user_tsk

The transaction_user_tsk performs the role of the transaction user. The TU task like all tasks is programmed as an infinite loop and is suspended at start-up. The task is resumed

when the user command or other tasks wish to communicate with it. The code below shows the outer working of the task.

```
/*
 * transaction_user_tsk()
 */
void transaction_user_tsk()
{
  for(;;)
  {
    (void)SuspendTask(TU_TSK_PRI);

      /* from below */
      /* Transaction to TU */
      if(SIPmsg->transaction2tu != BUSY0)
      {
      }
      /* Transport to TU */
      else if(SIPmsg->tport2tu != BUSY0)
      {
      }
      /* from above */
      if(SIPmsg->frominterface != BUSY0)
      {
      }
  }
}
```

Each of the message sections contains a switch for determining which message was received and dealing with it appropriately.

6.1.2.3 transaction_tsk

SIP is a transaction based protocol and the transaction task handles each transaction. The program is limited to one transaction at a time and as such the CANCEL method cannot be implemented. The task is suspended at start-up and is resumed by the transaction user when a transaction is required. The transaction task can act as either a user agent client or server. Both the UAC and UAS contain two transaction types: an INVITE

transaction and a non-INVITE transaction. The outer structure of the task can be seen below.

```
/*
 * transaction_tsk()
 * Contains client and server transactions.
 * Each transaction contains INVITE and NON-INVITE finite state machines.
 */
void transaction_tsk()
{

  for(;;)
  {
    (void)SuspendTask(TXN_TSK_PRI);

      switch(SIPmsg->ua_type)
      {
      /*  C L I E N T  */
      case UAC:
        /*  I N V I T E  */
        if(SIPmsg->transaction_type == INVITET)
        {
          /* client transaction state machine */
        }
        /*  N O N - I N V I T E  */
        else if(SIPmsg->transaction_type == NONINVITET)
        {
          /* client transaction state machine */
        }
        break;
      /*  S E R V E R  */
      case UAS:
        /*  I N V I T E  */
        if(SIPmsg->transaction_type == INVITET)
        {
          /* server transaction state machine */
        }
        /*  N O N - I N V I T E  */
        else if(SIPmsg->transaction_type == NONINVITET)
        {
```

```
        /* server transaction state machine */
      }

  }
}
```

The finite state machines for the INVITE and non-INVITE transactions are shown in appendices B.1 – B.4. The timers required for the state machines have not been implemented — timers were avoided after the hassles surrounding the interrupt servicing in the RTP implementation — though code is in place to include them. Most counters are not in fact required as TCP, which is reliable, is used at the transport layer. Most counters are for retransmissions.

6.1.2.4 transport_tsk

The transport task handles all dealings with the Internet transport layer. Like the transaction task it contains a switch statement with cases for the UAC and UAS. The transport task is resumed as required. The calling task will suspend itself in order for the transport task to run as the transport task has the lowest priority.

```
/*
 * transport_tsk()
 */
void transport_tsk()
{
  for(;;)
  {
    (void)SuspendTask(TPORT_TSK_PRI);          /* Suspend self */

      switch(SIPmsg->ua_type)
      {
      case UAC:   /* C L I E N T */
        if(SIPmsg->transport_task == SEND)      /* Send request */
        {
        }
        else if(SIPmsg->transport_task == RECEIVE) /* Wait for response */
        {
        }
```

```
        else if(SIPmsg->transport_task == END)  /* Session has ended */
        {
        }
        else                                    /* Error occured */

        (void)ResumeTask(SIPmsg->calling_task); /* Resume calling task */
        break;
      case UAS:   /* S E R V E R */
        if(SIPmsg->transport_task == LISTENO)   /* Listen for INVITE */
        {
        }
        else if(SIPmsg->transport_task == RECEIVE) /* Wait for request */
        {
        }
        else if(SIPmsg->transport_task == SEND) /* Send response */
        {
        }
        else if(SIPmsg->transport_task == END)  /* Session has ended */
        {
        }
        else                                    /* Error occured */

        (void)ResumeTask(SIPmsg->calling_task); /* Resume calling task */
        break;
      }

  }
}
```

6.1.2.5 Building a request

The syntax for SIP messages and SDP descriptions can be described in augmented Backus-Naur Form (ABNF) — a popular format for describing Internet protocols, itself described in RFC 2234 (Crocker & Overell 1997). SIP requests can be described generally as:

```
SIP-message     =  Request / Response
Request         =  Request-Line
                   *( message-header )
                   CRLF
```

```
[ message-body ]
```

This means that a request must consist of a request-line, zero or more message-headers, a new line (carriage return, line feed), and an optional message-body. HTTP/1.1 (Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999) is the base on which SIP is modelled (Rosenberg et al. 2002). SIP has a very flexible syntax. It allows for varying amounts of whitespace in most fields. Headers can be placed in any order, though headers with multiple header fields or multiple headers of the same type must be kept in the same order. For systems where the maximum transmission unit (MTU) becomes a factor many headers have an abbreviated form. This implementation uses a very limited and less flexible version of the syntax. Full header names are not supported where abbreviated names are allowed. The following is the partial ABNF for requests built by the implementation. This can be compared with the SIP captures and the ABNF in the SIP RFC (Rosenberg et al. 2002) for a better understanding of the messages as constructed by the implementation.

```
Request         =  Request-Line CRLF
                   Via CRLF
                   To CRLF
                   From CRLF
                   Max-Forwards CRLF
                   Call-ID CRLF
                   CSeq CRLF
                   Contact CRLF
                   [ Content-Disposition CRLF
                     Content-Type CRLF
                     Content-Length CRLF ]
                   ;The above optional items only included if a message-
                   ;body is present
                   CLRF
                   [ message-body ]

Request-Line    =  Method SP Request-URI SP SIP-Version CRLF
Method          =  INVITEm / ACKm /  BYEm
Request-URI     =  SIP-URI
SIP-Version     =  "SIP" "/" 1*DIGIT "." 1*DIGIT

Via             =  "v" HCOLON via-parm
via-parm        =  sent-protocol LWS sent-by SEMI via-params
via-params      =  via-branch
```

```
To         =  "t" HCOLON name-addr *( SEMI to-param )
to-param   =  tag-param

From         =  "f" HCOLON from-spec
from-spec    =  name-addr *( SEMI from-param )
from-param   =  tag-param

Max-Forwards  =  "Max-Forwards" HCOLON 1*DIGIT

Call-ID  =  "i" HCOLON callid
callid   =  word  "@" word

CSeq  =  "CSeq" HCOLON 1*DIGIT LWS Method

Contact         =  "m" HCOLON contact-param
contact-param   =  name-addr
name-addr       =  LAQUOT addr-spec RAQUOT
                   ;Display-name is not used
addr-spec       =  SIP-URI

Content-Disposition   =  "Content-Disposition"
disp-type             =  "session"
                         ;currently only "session" is supported

Content-Type     =  "c" HCOLON media-type
media-type       =  m-type SLASH m-subtype
                    ;As only SDP sessions are supported, the media-type
                    ;will always be "application/sdp"

Content-Length   =  "l" HCOLON 1*DIGIT
```

SIP_build_request basically concatenates each header to the previous headers in the request to form a complete request. Firstly the request line is built, followed by the via and to headers. If a to tag has is available it is appended to the to header field. This is followed by the from, max-forwards, call-ID, CSeq and contact header fields. If a body has been passed to the function it is appended following the necessary headers. Only SDP session data can be sent by the implementation.

There is one special case for building requests and that is the initial INVITE request. It is constructed by calling SIP_initial_invite which will initialise the necessary header

fields before calling SIP_build_request, the general purpose SIP request building function. Other calls to SIP_build_request will just collect the required information in the calling task before calling the function.

SIP_initial_invite gets passed a pointer to storage space for the message, a pointer to the already created SDP initial offer and a pointer to the SIP structure for storing and retrieving SIP state. The method is set to INVITE and the via branch is generated. As this implementation is based on RFC 3261 the via branch begins with the following string 'z9hG4bK'. As no tag is present in the 'to' header field in an initial INVITE, the relevant variable is set to NULL. A tag is generated for the 'from ' field. The maximum forwards value is set to the constant MAXFWDS, which is set to the recommended value of 70. This field, as the name suggests, limits the number of times a message can be forwarded before reaching its destination. The call-id field is generated next. The call-id is the local SIP URI with a random number prepended to it. Usually the call-id field will be a random string followed by the URI without the user name part, e.g. 123-DFewf345-34@example.com. The CSeq number is generated next. CSeq is used to identify and order transactions. The initial CSeq number is a randomly generated 32-bit number that is less than 2^{31}. The method is INVITE and the message body is an SDP offer. The call to SIP_build_request from SIP_initial_offer can be seen below.

```
message = SIP_build_request(message, mthd, sip->dialog.remote_uri,
                            sip->sipver, sip->tporttype, sip->local_ip,
                            sip->viabranch, sip->dialog.remote_uri,
                            sip->dialog.dialog_id.remote_tag,
                            sip->dialog.local_uri,
                            sip->dialog.dialog_id.local_tag,
                            sip->maxforwards,
                            sip->dialog.dialog_id.call_id,
                            sip->dialog.local_seq_no,
                            sip->dialog.local_uri,
                            sdp);
```

6.1.2.6 Building a response

Similar to requests, the general form of a response can be shown by the following ABNF notation.

```
Response        =  Status-Line
                   *( message-header )
```

```
CRLF
[ message-body ]
```

Responses are created in the same manner as requests. First the status line in generated. Reason phrases in the status lines are not required and have not been implemented for memory conservation reasons. Instead of leaving out the reason phrase completely a space has been included as a marker for future inclusion. Note that in the Ethereal (Ethereal 2004) captures, the reason phrase of a space is not shown in the SIP dissector — it is valid as per the ABNF syntax — though it can be seen in the raw data. After the status line, the via and to header fields are generated. If a to tag is available it is included else a new tag is generated. The from, call-ID, CSeq and contact headers are built and appended to the response. If a body is to be included it is added now after the necessary headers. Like the request building function, only SDP session data is allowed in the body. There is no need for any other body types in this implementation.

6.1.2.7 Matching messages to a transaction

SIP is a transaction based protocol. A transaction is started when a UAC sends, for instance, an INVITE. All responses from the UAS need to be matched to the relevant transaction. The same occurs for UASes. Messages received that do not correspond to a transaction must be passed to the TU. Although the implementation only supports one transaction at a time, the matching process is still used. Two functions are called from the transport task to check whether a received message matches a current transaction:

SIPmatch_response is called from the UAC to determine whether the received response matches the current transaction. For a response to match a transaction, the branch in the topmost via must match that of the request that started the transaction. As proxies are not used in this implementation only one via entry is found. The method in the CSeq must match that of the request that started the transaction.

SIPmatch_request is called from the UAS to determine whether the received request matches a transaction created by a previously received request. A request matches a transaction if the branch in the topmost via matches the equivalent of the request that started the transaction. The sent-by value in the top via must match the transaction-creating request's sent-by value and the method of the request must match that of the request that began the transaction unless the method is an ACK and the transaction is an INVITE transaction.

6.1.2.8 Extracting dialogue state

Received SIP messages need to be parsed in order to extract useful data. The following is the state required to identify and maintain a dialogue[1]:

dialog This is a variable used by the author to indicate whether the unit is in a dialogue or not.

dialog_id.call_id A dialogue is identified by this and the following two components. The call-ID is extracted from the request that lead to the dialogue being established.

dialog_id.remote_tag In a UAC the remote tag is the to tag from the response, whereas in a UAS it is the from tag from the request.

dialog_id.local_tag The remote tag is the from tag from the request in a UAC and the to tag from the response in a UAS.

local_seq_no The local sequence number is the CSeq generated for the request in a UAC and initially empty in a UAS. If the unit previously a UAS decides to send a request within the dialogue — becomes a UAC — then the generated CSeq will become it's local sequence number.

remote_seq_no The remote sequence number is the reverse of above, i.e. the client at the start of the dialogue will have an empty remote sequence number field.

local_uri The local SIP URI.

remote_uri The remote user's SIP URI. Can be extracted from the relevant to and from headers.

remote_target The remote target is the URI extracted from the contact header field in the response for a UAC and the request for a UAS. The URI in the contact header is the actual URI of the endpoint. Remote target is not used as this implementation is designed for direct endpoint-to-endpoint communication and the URI in the contact field will match that in either the to or from header fields.

secure Secure is a flag that is not implemented. The flag indicates secure SIP communication.

route_set The route set must be set to the URIs in the record-route header field of the response for a UAC and the route-route from the request for a UAS. This field is always empty as proxies and therefore the record-route header are not used.

[1]The American spelling of dialogue is used in the code as the SIP specification is written in American English and it is shorter and less likely for statements to exceed the page width.

Two functions are available to parse received messages — passed to the TU — for dialogue state:

SIP_analyse_response This function is called by the UAC on establishing a dialogue. It extracts the remote tag from the to header field of the response. This completes the state required by the UAC at the start of a dialogue.

SIP_analyse_request This function is called by the UAS on receiving a request that may lead to a dialogue being formed. The remote user's IP address and the branch are extracted from the via header field. The call-ID, remote URI, remote tag and remote sequence number are gathered from the request. The SIP URI present in the to field is compared to the local SIP URI. If it matches then the URI is stored with the dialogue state, else the function returns an error indicator. The route set is set to zero as no servers have been traversed.

6.1.2.9 Matching a request to a dialogue

It must be established whether a request received after a dialogue has been created belongs to the dialogue or not. With the implementation in it's current form this will be a BYE request, if correctly formed, for the established dialogue. A request matches a dialogue and is acceptable if the dialogue ID matches that of a dialogue in place and the sequence number is greater than the current sequence number if the user agent has sent a request pertaining to this dialogue. Sequence numbers for transactions must be monotonically incremented during a dialogue. If this is the first request sent by the user agent for this dialogue then a sequence number will have been generated. Any future transactions within the dialogue will require the sequence number to be incremented. Therefore the function parses and compares the dialogue ID components; call-ID, remote tag and local tag to those stored with the dialogue state before checking the validity of the sequence number.

6.1.3 Problems encountered

One of the problems encountered involved the initialisation of strings within the software. Most strings are defined as pointers with the memory initialised at compile time to the required size by using strings of '0's. The program was producing garbled header fields and was behaving erratically. After evaluating the assembly code and the makefile provided by Xecom, it was found that a compiler argument was causing the problem. The compiler was set to conserve space by allowing identical strings to share the same memory. Therefore many of the strings shared the same memory locations and changes

to one string would affect all the others. The strings were initialised to different values to bypass this optimisation without removing the argument from the compiler call. Some problems were encountered because of the way the tasks are suspended or delayed during inter-task communication. A static code checker called splint (info@splint.org 2004) was used to help fix code and catch potential errors. Most issues, however, were related to the specifications and rereading the documents allowed the problems to be resolved.

6.2 SDP

The session description protocol is carried in the body of session initiation protocol messages and is used for describing sessions. The describing of sessions is required to inform parties as to the configuration required in order to join a session. It is used in SIP for negotiating session parameters. It is interesting to note that the session description protocol specification states that:

> SDP is not intended for negotiation of media encodings. *(Handley & Jacobson 1998)*

SDP is defined in RFC 2327 and its use with SIP is found in RFC 3264. Additionally the media attribute 'inactive' is used as specified in RFC 3108, Conventions for the use of the Session Description Protocol for ATM Bearer Connections. Note that 'inactive' has been included in Internet drafts that when / if approved will replace RFC 2327.

6.2.1 Functionality

SDP is a simple rigidly structured format for describing multimedia sessions. Each session element is represented in the form <type>=<value> where 'type' is a case sensitive single character and 'value' is a string dependant on 'type'. There can be no whitespace on either side of the equals sign (Handley & Jacobson 1998). The text below (Handley & Jacobson 1998) shows the format of an SDP description and what each 'type' stands for.

```
Some lines in each description are required and some
are optional but all must appear in exactly the order given here (the
fixed order greatly enhances error detection and allows for a simple
parser). Optional items are marked with a '*'.

Session description
```

```
        v=  (protocol version)
        o=  (owner/creator and session identifier).
        s=  (session name)
        i=* (session information)
        u=* (URI of description)
        e=* (email address)
        p=* (phone number)
        c=* (connection information - not required if included in all media)
        b=* (bandwidth information)
        One or more time descriptions (see below)
        z=* (time zone adjustments)
        k=* (encryption key)
        a=* (zero or more session attribute lines)
        Zero or more media descriptions (see below)

Time description
        t=  (time the session is active)
        r=* (zero or more repeat times)

Media description
        m=  (media name and transport address)
        i=* (media title)
        c=* (connection information - optional if included at session-level)
        b=* (bandwidth information)
        k=* (encryption key)
        a=* (zero or more media attribute lines)
```

The session description is split into two levels: the session level and the media level. Information regarding the session as a whole is provided in the session level while individual media attributes are described at the media level. For items that occur at both levels the item at the media level overrides the session level item. The SDP specification states that either a phone number or e-mail address item must be present in a session description. RFC 3264 changes the 'must' to 'may'. RFC 3264 specifies that only one session description must be present in each message during the offer/answer process.

6.2.2 Implementation

For the purposes of this project, the session description protocol has been further simplified and is implemented as shown below. A main factor in the simplification was a

concern regarding the memory requirements of the signalling section. Each program is limited to 64KB for both its code and data memory. This was a concern as both SIP and SDP require the storing and parsing of relatively long strings besides the other protocol requirements.

The following is an actual session description captured using Ethereal (Ethereal 2004). First is the packet hex data — with ASCII characters displayed in the right column — followed by the session description in human readable form and detailed description of the session description.

```
0160                  76 3d 30 0d 0a 6f 3d 63 6f 6d 31 20 33      v=0..o=com1 3
0170  33 32 38 30 31 37 38 34 20 33 33 32 38 30 31 37   32801784 3328017
0180  38 34 20 49 4e 20 49 50 34 20 31 35 35 2e 32 33   84 IN IP4 155.23
0190  38 2e 33 33 2e 32 34 35 0d 0a 73 3d 2d 0d 0a 63   8.33.245..s=-..c
01a0  3d 49 4e 20 49 50 34 20 31 35 35 2e 32 33 38 2e   =IN IP4 155.238.
01b0  33 33 2e 32 34 35 0d 0a 74 3d 30 20 30 0d 0a 61   33.245..t=0 0..a
01c0  3d 73 65 6e 64 72 65 63 76 0d 0a 6d 3d 61 75 64   =sendrecv..m=aud
01d0  69 6f 20 35 30 30 34 20 52 54 50 2f 41 56 50 20   io 5004 RTP/AVP
01e0  31 38 0d 0a 61 3d 72 74 70 6d 61 70 3a 31 38 20   18..a=rtpmap:18
01f0  47 37 32 39 2f 38 30 30 30 0d 0a                 G729/8000..
```

```
v=0
o=com1 332801784 332801784 IN IP4 155.238.33.245
s=-
c=IN IP4 155.238.33.245
t=0 0
a=sendrecv
m=audio 5004 RTP/AVP 18
a=rtpmap:18 G729/8000
```

```
    Session Description Protocol
        Session Description Protocol Version (v): 0
        Owner/Creator, Session Id (o): com1 332801784 332801784 IN IP4
155.238.33.245
            Owner Username: com1
            Session ID: 332801784
            Session Version: 332801784
            Owner Network Type: IN
            Owner Address Type: IP4
            Owner Address: 155.238.33.245
        Session Name (s): -
```

```
Connection Information (c): IN IP4 155.238.33.245
    Connection Network Type: IN
    Connection Address Type: IP4
    Connection Address: 155.238.33.245
Time Description, active time (t): 0 0
    Session Start Time: 0
    Session Start Time: 0
Session Attribute (a): sendrecv
Media Description, name and address (m): audio 5004 RTP/AVP 18
    Media Type: audio
    Media Port: 5004
    Media Proto: RTP/AVP
    Media Format: 18
Media Attribute (a): rtpmap:18 G729/8000
    Media Attribute Fieldname: rtpmap
    Media Attribute Value: 18 G729/8000
```

`v=0`

The first field is the protocol version number. RFC 2327 describes version 0.

`o=com1 332801784 332801784 IN IP4 155.238.33.245`

The origin field contains a username, session-ID, session-version, network type, address type and address. The username is extracted from the SIP URI for the local user. The session-ID is calculated as a 32-bit value with a maximum of $(2^{30}) - 1$. Session-version is initialised to the same value. Though the values are initialised the same, they are stored separately in the SDP structure and so can be modified independently if necessary. It was decided to scale down the 64 bit values that RFC 3264 require for the session ID and version fields to save memory and simplify processing as a 16 bit processor is being used. The network type is set to 'IN' representing the Internet. 'IN' is the only acceptable value. The unit only uses IP version 4, therefore 'IP4' is the only valid option for the address type field. The address field contains the IP4 address of the unit on which the session description is being built. The address is represented in dotted decimal notation that is the 32 bit IP address is split into four bytes which are displayed in decimal separated by dots, e.g. 155.238.33.245.

`s=-`

RFC 3264 recommends that the session name be set to space or dash for unicast sessions.

This implementation always sets the session name to dash. When a received session name is parsed, only a dash is considered valid, though this could easily be changed to allow for a space if required.

`c=IN IP4 155.238.33.245`

The connection field's values; network type, address type, and connection address are set to the equivalent values in the origin field. The parser ignores the connection field's values, assuming them to match those in the origin field. Being a unicast two party system, the SDP message's origin information will always match that in the connection field. The implementation therefore does not allow for third party call setup.

`t=0 0`

The time fields are used to indicate the start and stop time of a conference session (Handley & Jacobson 1998). Both start and stop fields are set to zero, and expected to be zero in received messages. This is done as start and stop times are determined by external means, i.e. the user with SIP starting and terminating calls. There are no explicit call start and stop times.

`a=sendrecv`

The implementation only supports one stream per session, therefore it was decided that the stream type attribute field must only be present at the session-level description. There are four possible attributes; 'sendrecv', 'sendonly', 'recvonly', and 'inactive'. The default is 'sendrecv' therefore leaving the attribute field blank is equivalent to the attribute 'sendrecv'. These attributes only apply to the media stream (transported using RTP) and do not affect the transmission of RTCP packets. Therefore an 'inactive' attribute will cause a media session with only RTCP packets being transmitted between units.

`m=audio 5004 RTP/AVP 18`

Only a single media description is supported in this implementation and as the system, as implemented, only transports audio, the media type is set to 'audio'. The port value is set to the local port from which RTP packets will be sent or received. The RTCP port is derived by incrementing the RTP port number by one. A port value of zero means that the implementation does not wish to use that stream. The transport protocol field is set to 'RTP/AVP' as the stream is sent using RTP in conjunction with the audio / visual profile (Schulzrinne & Casner 2003). The last supported field within the media description contains the RTP payload type with which the user wishes to communicate.

`a=rtpmap:18 G729/8000`

An attribute field accompanies the media description and contains additional information about the media being sent or received. This attribute is known as an 'rtpmap' and essentially maps an encoding name and clock rate to a payload type. It is recommended that this attribute be included though it is mostly useful for mapping encoding names to dynamic payload types. The payload type is the same as in the media description. A function, SDPget_encname returns the encoding name for the payload type selected. The clock rate is set to 8000. The only payload type allowed is 18, which indicates G.729.

G.729 has a sample rate of 8KHz. This attribute field's values are never evaluated but assumed to be correct if the media description is valid.

6.2.2.1 SDP functions

SDP_initial_offer sets up the fields required to build the initial SDP offer and then calls SDP_build.

SDP_answer extracts the data from the received offer using SDP_parse and uses that data to fill the offer fields of the SDP structure. The function will then construct the data for the necessary fields for the SDP answer and call SDP_build.

SDP_analyse_answer extracts the data from the received answer using SDP_parse and fills the answer fields of the SDP structure.

SDP_build compiles an SDP message based on the offer or answer SDP data with which it is supplied. The function calls SDPget_encname in order to get the encoding name matching the payload type required in the rtpmap attribute field.

SDP_parse does some simple tests to determine whether the session description is valid and extracts data from fields that will be required for use in the program.

SDPget_username extracts the username from a SIP URI or returns '-' if no username is present.

SDPget_session_id generates a 32 bit value and ensures that it is no larger than $(2^{30})-1$.

SDPget_encname is a small function that takes an RTP payload type and returns it's name, e.g. G729 for payload type 18.

Chapter 7

Media transport

Encoded audio is passed to the real-time transport protocol (RTP) for transmission by the host to the remote target. UDP is the underlying transport protocol. The encoded audio payload is extracted from RTP packets received from the transport layer and passed to the codec for decoding.

The media to be transported between endpoints is G.729 Annex A (G729A) encoded audio. G.729 Annex A is a simplified version of the G.729 voice codec maintaining its interoperability with G.729 (ITU-T 1996*c*). G.729A has a sample rate of 8KHz (ITU-T 1996*c*). 80 samples, corresponding to 10ms of audio, are encoded into 10 byte frames (ITU-T 1996*c*). The low bit rate — 8Kbps — makes the codec suitable for low speed, e.g. dial-up, connections.

7.1 Voice codec

The codec section consists of:

- An analogue user interface
- The voice codec
- And the interface to the real-time transport protocol.

The codec, as the name suggests, performs two functions: encoding and decoding, and as such the above-mentioned components can all be split in two parts corresponding to the codec's functions. The codec section can also be split from a physical perspective. The

encoders and decoders (and analogue interfaces) are additional hardware that is connected to the controller board.

7.1.1 Functionality

7.1.1.1 Encoder

Audio is sampled from a microphone at 8KHz. Every 10ms of audio is encoded into a 10 byte frame that is sent to a buffer for an 8 bit parallel interface. An interrupt is generated when this event occurs and the frame along with the current RTP timestamp is read and placed into a first in first out (FIFO) buffer. The RTP implementation can extract the data from the FIFO buffer for transmission.

7.1.1.2 Decoder

Every 10ms, the period of an encoded frame, a FIFO buffer for storing G.729A frames from the real-time transport protocol is checked. If there is a frame in the buffer it is passed to the decoder via an 8 bit parallel interface. The frame is decoded and output to a speaker.

7.1.2 Implementation

The description of the implementation is split in parts along the physical divide between the controller board and the codec boards.

7.1.2.1 Codec boards

Many hours were spent searching for an appropriate codec IC. Two codecs were searched for in particular: the G.723.1 and G.729 codecs. G.723.1 is a dual bit rate speech codec designed to operate at 5,3 and 6,3Kb/s. G.729 is a speech codec that operates at 8Kb/s. Both are therefore suited to low speed data links.

For a long time all searches lead to companies licensing code for various DSP chips. As the intellectual property rights for G.729 (and G.723.1) are owned by various organizations, a G.729 consortium enabling one-stop licensing was created. Licenses can be obtained from Sipro Lab Telecom.

Eventually the author found something closer to what he had been searching for. A G.729 annex A voice codec IC designed for 'use in memory-based voice recording and playback applications'. The chip was the MAS3504D from Micronas.

Contacting Micronas the author was informed that the chip was no longer available but had been replaced by the MAS3549F and that it could be purchased via an Italian company, TARA. TARA were able to supply the author with samples of the MAS3519F, a chip from the MAS35x9F family that not only had a G.729A codec but included an MP3 decoder. The MAS3519F was never viewed as an ideal solution but rather as an available solution. This was because the codec was designed for record and playback applications and was therefore not as well suited to real-time work as other solutions. The IC can not encode and decode data simultaneously. Any changes made to the configuration of the codec are done using the I^2C interface which is relatively slow. The codec works in pages of 50 G.729A frames and changes will only take place on a boundary between pages, i.e. only every 500ms. Therefore two chips are required for full duplex operation. A single MAS3159F may be undesirable even for half duplex communication as the program would need to keep track of page boundaries and would need to include additional code to deal with the page boundary issue when communicating with a unit not using the same codec. Half duplex implementations requiring short switching times between transmitting and receiving would still require two chips. The MAS3519F comes in a 64 pin PLQFP, surface mount package which is not well suited to prototyping. The package is not easy to solder by hand and sockets for the device are expensive and are surface mount components themselves. There were however a number of factors in its favour:

- As a dedicated audio codec and not a DSP solution it would not require any additional development tools for implementation. A DSP solution would have required getting software development tools as well as a programmer and possibly an in-circuit emulator. Software for the DSP would be required. This must either be developed or licensed from a software developer. Use of the G.729A codec requires licenses from the various patent holders.
- The licensing fees for the MP3 decoder and the G.729A codec in the MAS3519F have been incorporated into the cost of the product.

Printed circuit boards needed to be developed for codec chips. One of the main concerns was cost. The surface mount codec chips could not easily be removed if required. The PCBs on which the surface mount components would be mounted would be made by an outside company and not by the Technikon's PCB plant and the surface mount ICs would be populated by an external company, both of which would increase costs. The Technikon's PCB plant cannot produce a solder mask or through-hole plating. Three designs were considered incrementally:

7.1.2.1.1 Design 1: The MAS3519F is mounted on a daughter board. The daughter board connects to the motherboard, which contains all the additional circuitry. The motherboard is connected to the AWC86 controller board. The motherboard has sockets for 2 daughter boards, i.e. one an encoder and the other a decoder. It is necessary to have some components (capacitors mostly) mounted on the daughter board, as these components are required close to the codec IC. The idea behind the daughter boards was to allow the ICs to be reused or replaced if necessary. The motherboard can be altered without having to buy more codec ICs.

7.1.2.1.2 Design 2: Sockets for the codecs' PLQFP64 package are very expensive. 'Homebrew' daughter boards as described in design one will be quite big due to the number of pins on the IC. A number of the additional components need to be placed on the daughter board. There are potential routing and connection hassles. Using the daughter boards increase track lengths unnecessarily. Keeping the codec ICs reusable for other projects is unnecessary and outside the scope of my project. With careful design, the motherboard will not need any corrections or adjustments. Therefore it was decided to scrap the daughter board idea and attach the codec ICs directly to the motherboard. The same motherboard design as in design 1 is used.

7.1.2.1.3 Design 3: Instead of having a PCB with both an encoder and decoder onboard, each codec could have it's own board. The boards will be fairly generic, allowing the user to configure each as either an encoder or decoder. This design, like design 2, will have the codec ICs soldered directly on to the boards.

Design 3 was chosen for the codec board design. Advantages of the design are that it allows for a certain amount of reusability and the boards can be swapped if necessary. Disadvantages include an increased component count and, like in design 2, if the boards need to be redesigned all the components need to be reordered including the codec ICs. There is no local supplier for the codec ICs, they are ordered from Italy.

7.1.2.1.4 Design specifics: The board's dimensions are 75mm x 60mm allowing two boards to be placed next to each other and connected to the controller board as shown in figure 7.1.

At roughly the centre of the board is the MAS3519F audio encoder / decoder module. The MAS3519F is a highly configurable device with various power supply configurations and media interfaces, as illustrated in figure 7.2. It was decided that a single supply would be used. 2,85V was chosen as the supply voltage. To achieve this, an LM1117, 2,85V low-dropout voltage regulator, was used. The input to the regulator comes from

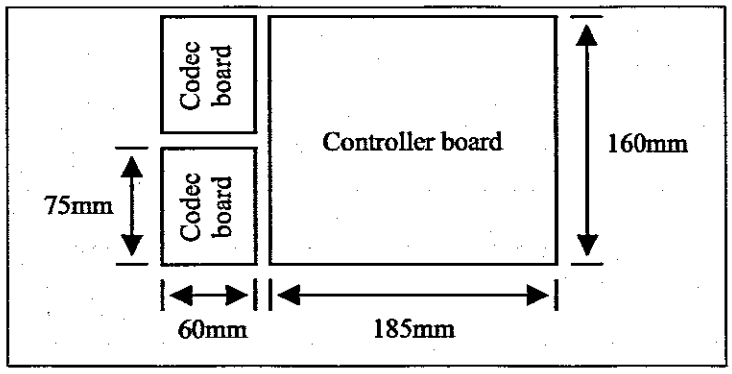


Figure 7.1: Positioning of codec boards with respect to the controller board and their dimensions.

the controller board's 5V supply. Only two digital interfaces are implemented; I^2C and parallel interfaces. The I^2C interface is required for configuring the device. The absolute maximum input voltage for the I^2C pins is 6V but there is no mention of a recommended maximum input voltage for these pins. It was felt that it would be safer to operate within the MAS3519F's supply voltage range. As the AWC86 has 5V I/O, interface circuitry was required. The level shifter circuit described in section 18 of the I^2C-bus specification was used for this purpose. Figure 7.3 shows the diagram from the above-mentioned specification.

There is no ambiguity regarding the safe voltage range for the parallel interface and it's control lines — they are not 5V tolerant. Low voltage devices with 5V tolerant inputs and outputs were used to interface with the controller board. The 74LCX245, bi-directional transceiver, was used for the eight data lines. A 74LCX126, quad buffer, was used for the two control lines, which are inputs. The device's reset pin is also connected to the 74LCX126. The output only pins do not have any interface circuitry as the output voltages fall within the accepted range for the AWC86 controller module. Three additional pins are connected to the edge connectors.

On the analogue side, microphone in, line in, and headphones output have been implemented. The MAS3519F can therefore be configured as a G.729A encoder with line or microphone input, a G.729A decoder, or an MP3 decoder with analogue output only. Figure 7.4 shows a block diagram of the codec board. The schematic can be seen in appendix C.1

The parallel I/O on the MAS3519F default to outputs. A control line is required to switch the transceiver data direction on the encoder board to ensure that be ports are configured correctly before writing to the codec.

A second codec using the PC based codec from VoiceAge was implemented. This will be described in the chapter 9.

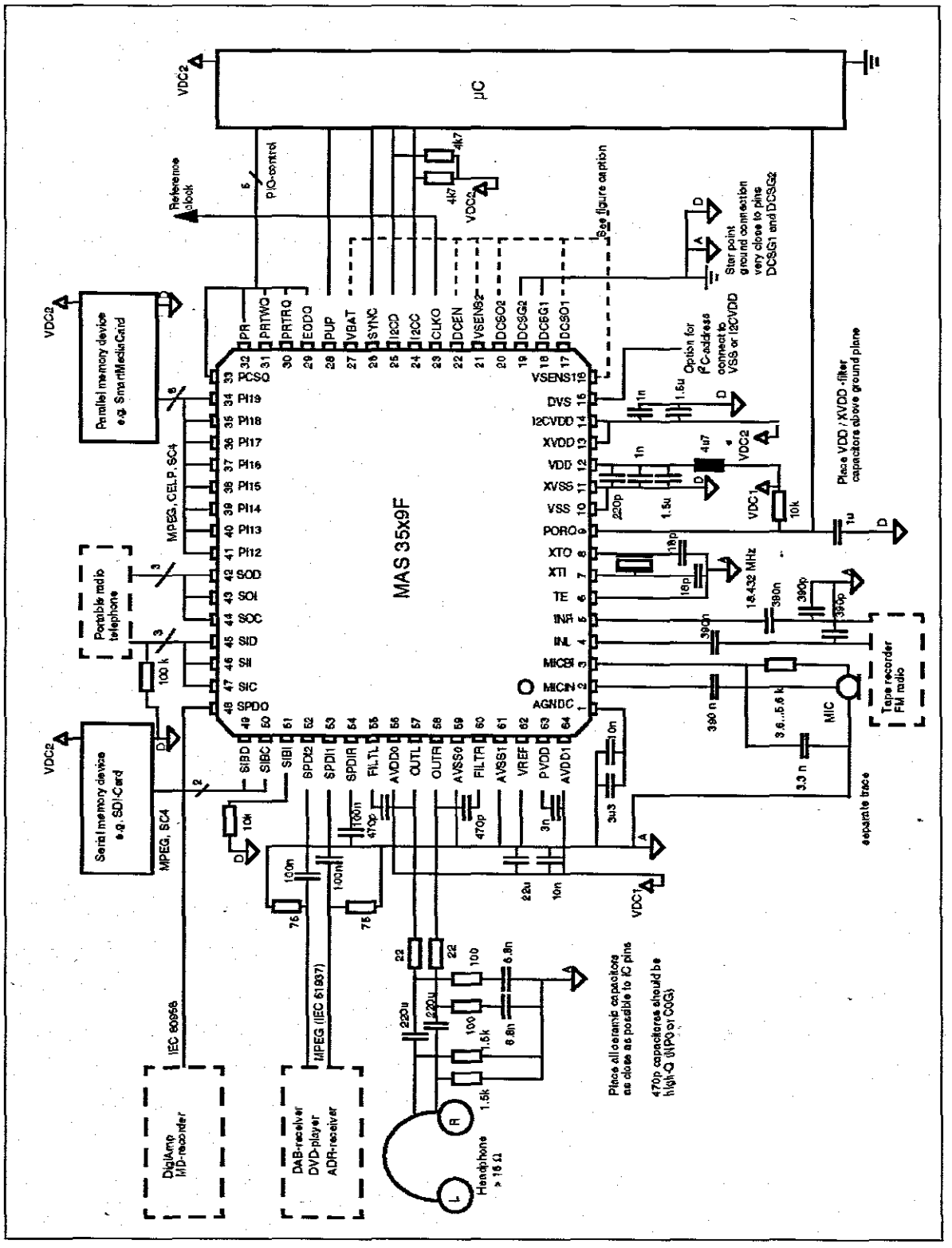


Figure 7.2: Typical application of MAS3159F (Micronas 2001).

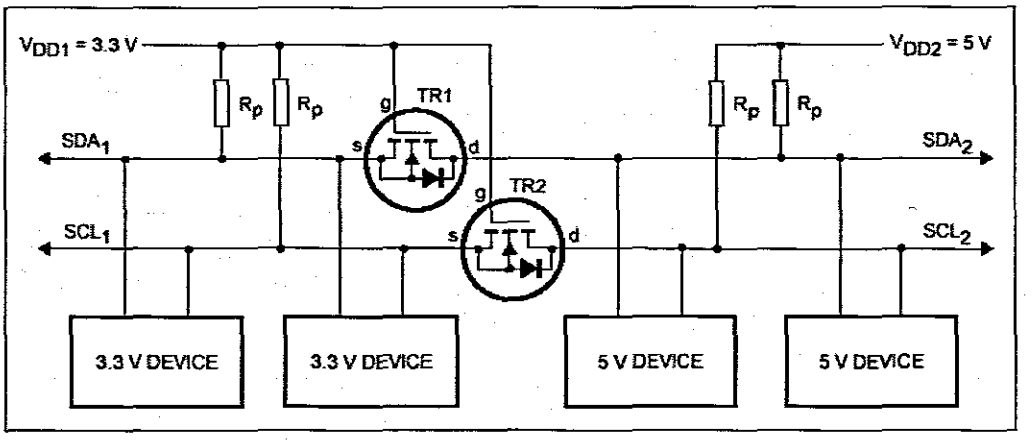


Figure 7.3: Circuit to connect I^{2}C devices operating at different voltages (Philips 2000).

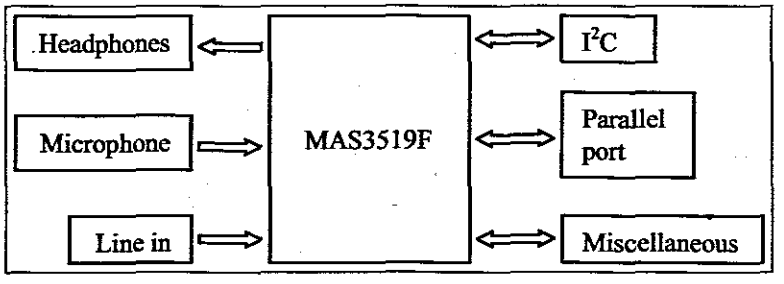


Figure 7.4: Block diagram of codec board.

7.1.2.2 Controller board

The codec interface code is closely linked to the RTP code. It has been implemented in the same compiled program as the real-time transport protocol. The codec chips are controlled and configured through the RTP_tsk. Data frames, and the RTP timestamp for frames to be transmitted, are sent and received via FIFO buffers to RTP_tx_tsk and RTP_rx_tsk respectively.

Two digital interfaces to the codec IC were implemented as described in the MAS3519F datasheet. An I^{2}C interface is required to configure the device and the parallel interface (PIO) is used to transfer data between the AWC86 module and the MAS3519F. The hardware is configured as mentioned in the previous section. Five controls lines are associated with the parallel port. Selected lines are used depending on how the parallel port is to be used. Two of the three modes of operation are implemented in the code:

- Direct memory access (DMA) mode input.
- Output.

The timing diagram for the input mode is show in figure 7.5.

Data can only be sent to the parallel interface while the PIO end of DMA ($\overline{\text{EOD}}$) line

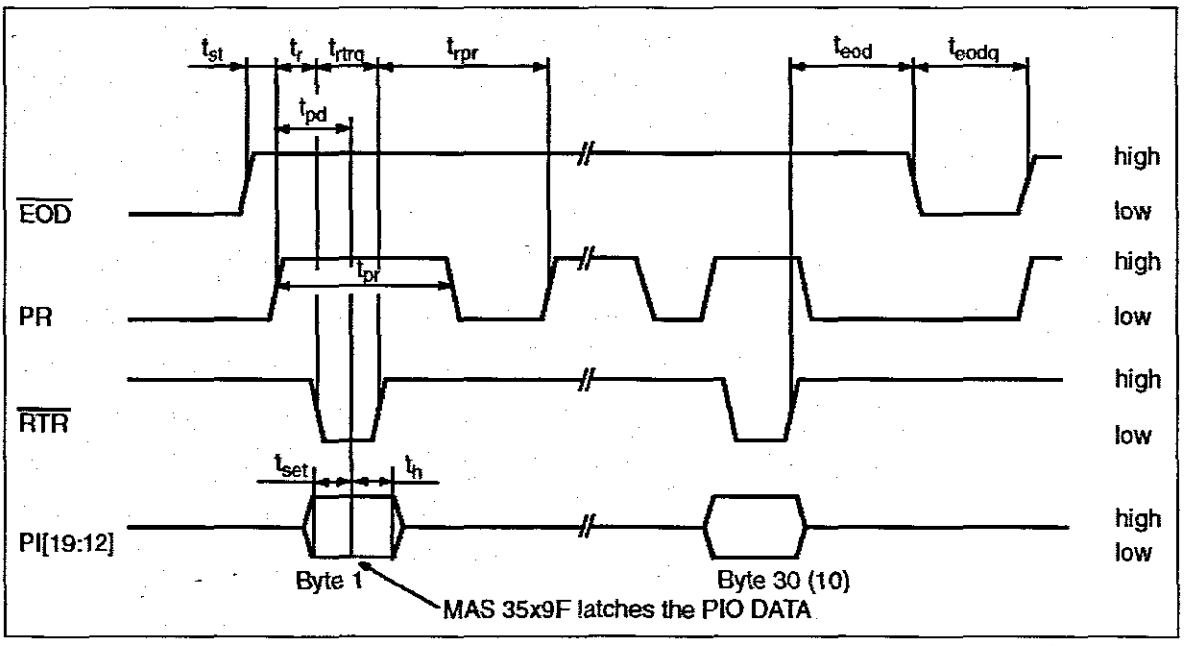


Figure 7.5: Parallel input interface timing diagram (Micronas 2001).

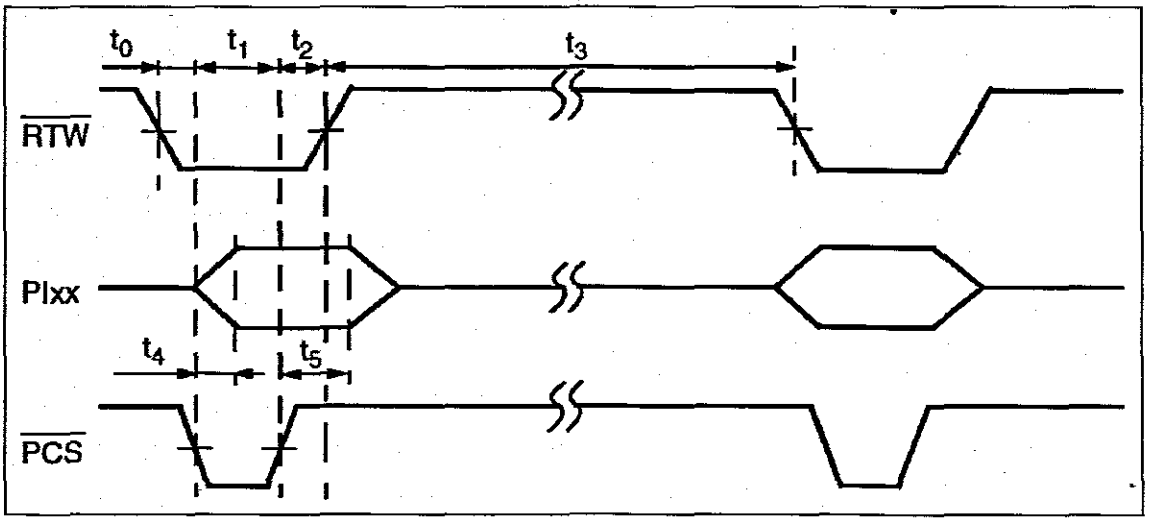


Figure 7.6: Parallel output interface timing diagram (Micronas 2001).

is high. The controller sets the PIO DMA request (PR) line high when there is a byte ready to be transferred to the MAS3519F. The MAS3159F will pull the ready to read ($\overline{\text{RTR}}$) line low when it is ready to read a byte and set it high again once the byte has been read. When configured for G.729 operation, one frame — 10 bytes — can be sent before the $\overline{\text{EOD}}$ line goes low again. The data is sent on pins PI12 – PI19 with PI12 being the least significant bit.

The timing diagram for the output mode is shown in figure 7.6.

Ready to write ($\overline{\text{RTW}}$) is an output line from the MAS3519F and will go low when a byte is available to be transferred. The PIO chip select ($\overline{\text{PCS}}$) line is controlled by the micro-controller to indicate the reading of the data to complete the handshaking.

An inverter is required on the $\overline{\text{RTW}}$ line from the encoder codec board as it is used to indicate when a frame is ready to be transferred to the controller. The interrupt on the controller pin is only generated on a low to high transition or active high level-sense.

Though the datasheet states that the voice codec is compliant to the G.729 Annex A standard, it does not explicitly mention the bit packing of the generated G.729 frame (Micronas 2001). It does however state that the data is passed to the parallel interface is in byte swapped order. Therefore the data read by the controller swaps the byte order before transmitting the data as it has been assumed that the generated data is in the order specified by the RTP G.729 profile (Schulzrinne & Casner 2003). The receiver byte swaps the data again before sending it to the codec for decoding.

7.1.2.2.1 Start-up and configuration of codecs At start-up the codecs are inactive as the $\overline{\text{reset}}$ pin is held low. During RTP configuration, if the codecs are required, the function codec_setup is called. Codec_setup accepts the argument 'code' which indicates whether the function should configure the encoder or decoder. First the I/O pins are configured to allow for communication with the codec via its parallel port. Note that for the decoder the (micro-controller) parallel data pins are only set to outputs after the configuration has completed successfully. The i2c_encode or i2c_decode functions are called which communicate I^2C sequences that have been devised to configure the codec as required. If the returned value is TRUE normal operation continues else the $\overline{\text{reset}}$ line is pulled low again. The returned value from i2c_encode / i2c_decode is also returned by the codec_setup function to indicate to the calling function, in this case rtp_start_session whether the configuration was successful. If the codec_setup function returned a value of TRUE and the encoder is required then the interrupt on the $\overline{\text{RTW}}$ line will be initialised. This will allow the AWC86 to respond each time a frame is ready to be read from the MAS3519F. The codec boards are deactivated by simply pulling the $\overline{\text{reset}}$ line low.

7.1.2.2.2 Reading from the codec The codec reading utility is implemented as a task with a relatively high priority. The task uses a semaphore to indicate when an interrupt has occurred.

The first time the task is run a semaphore is created that will signal the task when an interrupt is generated on the $\overline{\text{RTW}}$ line. After this initialisation, the task will wait, essentially suspend itself, for a semaphore signal. Once this has occurred all maskable interrupts are disabled and the interrupt pin is configured as an input to be used in the handshaking procedure. The data frame is read and the RTP timestamp stored. The timestamp value and the frame, which is byte swapped, are then put into a FIFO buffer where they can be accessed by the RTP transmission task. The interrupt pin is set back

to an interrupt source and the maskable interrupts are enabled. The code returns to waiting for the next semaphore signal. Note that the timestamp should correspond to the sampling instant the first octet of the encoded audio in a frame. The actual sampling instant is not known to the micro-controller. Concerns over timing lead to getting the timestamp value after reading a frame from the port.

7.1.2.2.3 Writing to the codec Like the codec reading task, writing to the codec's parallel port is implemented as a high priority task. The task is quite simplistic. The codec write task suspends itself at start-up. It waits for 40ms after it has been resumed to allow data to begin collecting in the FIFO buffer. The task then enters an infinite loop. The first duty is to check whether it should break out of the loop. Assuming that doesn't happen the task will suspend itself for 10ms. This allows the task to loop every 10ms — the amount of audio, in time, that a single encoded frame of G.729 represents. The task will then check that there is at least one frame available in the FIFO buffer and transfer it, after byte swapping the frame, to the parallel interface.

7.1.2.2.4 Problems encountered The most time-consuming software problem of the project was in the codec interface software. After the codec board had been attached to the system and the I^2C interface and commands had been figured out, tests began to transmit data received from the codec board. The section of code given the task of reading G.729A frames from the codec IC was implemented as an interrupt service routine (ISR). The first high to low transition of the $\overline{\text{RTW}}$ line for each data frame was used to trigger the interrupt. The signal had to be inverted as the interrupt on the AWC86 can only be generated by low to high transitions. The program would crash each time it was run. After some time, the cause was narrowed to one of three things, namely the interrupt service routine, the FIFO buffer, or a memory related problem which only became evident with the ISR included. Numerous tests were conducted where lines of code removed or altered or inserted at different places. After a while it became evident that no single line of code seemed to cause the problem, though the program crashed at roughly the same point every time it was run. The author was wary of using semaphores as he had been informed that the semaphore controls were still undergoing beta testing. Attempts to obtain a more recent version of the operation system or to ascertain whether the semaphore functions had been proven to be stable were fruitless. Nonetheless, it was decided to implement the codec read ISR as semaphore controlled task. This solved the problem. Whilst conclusive proof of the cause of the problem could not be determined, the following theory seems to explain what happened. MicroRTOS runs 'programs' called tasks. Each task is assigned a priority number, an ID number, and a stack. This information is stored in a task table. It is felt that as the ISR is an independent function and not a task, it lived outside the control of the operating system and may have accessed memory allocated to other tasks.

This would have caused the system instability.

The data sheet states that the frame sent from the parallel interface is in byte swapped order. It does not however state whether frames sent to the MAS3519F for decoding need to be byte swapped or not. Attempts to contact Micronas for technical support were futile. The supplier was helpful though neither they nor the engineer they recommended could answer my questions.

Section 9.5 will explain other issues with the codecs.

7.2 RTP

The encoded audio is transported using the real-time transport protocol (RTP). The real-time transport protocol has an accompanying and very closely linked control protocol called the RTP control protocol, or RTCP. RTCP will be described in the section on Quality of Service. The real-time transport protocol is specified in RFC 3550 (Schulzrinne et al. 2003) and the RTP profile for audio and video conferences with minimal control is specified in RFC 3551 (Schulzrinne & Casner 2003). These two documents were not used in the development of the RTP implementation as they were only approved after development was complete. Initial work on RTP used RFC 1889 (Schulzrinne, Casner, Frederick & Jacobson 1996) and RFC 1890 (Schulzrinne 1996) as reference as they were the appropriate standard documents before the previously mentioned RFCs made them obsolete. During the development of the software the author discovered the ongoing work being done on RTP. Internet drafts are meant to be cited as work in progress and not used for development however the RTP drafts had gone through a number of revisions and were nearing a final state when they were discovered. Therefore it was decided to work from the then most recent draft documents. As the obsolete Internet drafts may become difficult to obtain all references in this text will refer to the most recent RFCs unless otherwise stated.

7.2.1 Functionality

This implementation has been designed for two user unicast operation. It was not seen as necessary to implement the means for allowing the implementation to scale to large sessions. The resources of the controller would limit usage to small sessions, i.e. sessions with few participants.

7.2.1.1 Session initialisation and control

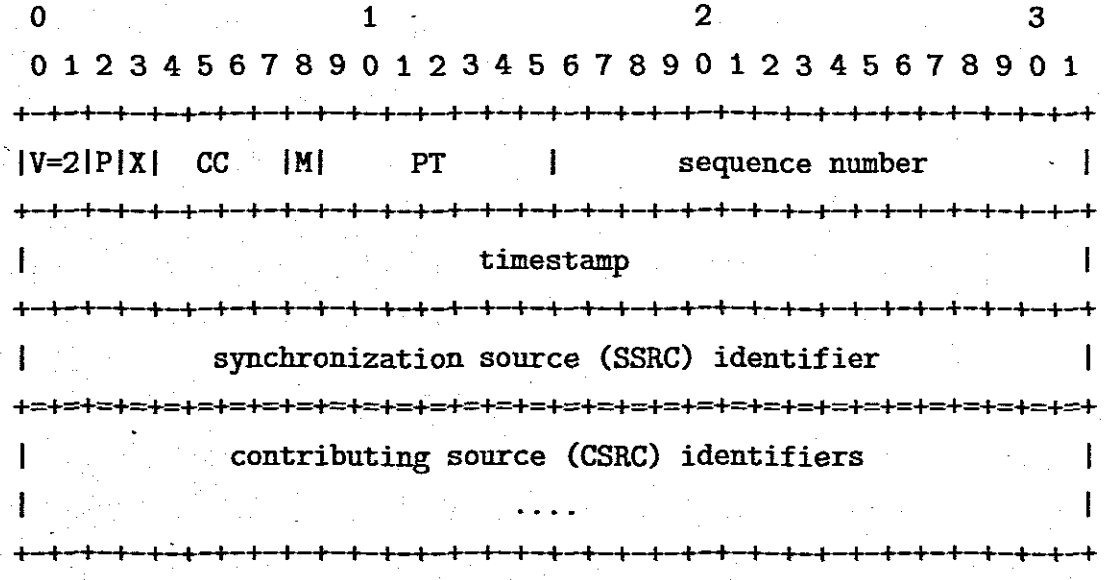
Pseudo-random values are generated for the synchronisation source identifier (SSRC), and initial sequence number and RTP timestamp at the beginning of a session. Other initialisation including configuring the codecs is done if required.

SSRC's are required to be unique within an RTP session. An SSRC collision occurs if another participant's application happens to have selected the same SSRC value. If this happens the unit must send an RTCP BYE packet and select a new SSRC value.

An RTP session is ended quietly if no RTP or RTCP packets are received from a participant for a certain period of time. Therefore RTP sessions can be ended simply by refraining from sending any further packets. The preferred method of ending a session is to send an RTCP BYE packet.

7.2.1.2 RTP send

The RTP audio profile states that each RTP packet should contain 20ms of audio. A G.729 frame represents 10ms of audio, therefore two frames are sent per RTP packet. Encoded audio is placed into a FIFO buffer when read from the codec. An RTP packet will be built provided there are at least two frames in the buffer. The timestamp from the first audio frame extracted from the FIFO buffer is used as the timestamp for the packet. The sequence number is incremented after the packet has been built and sent. This procedure will repeat itself every 20ms.



Above is the format of an RTP packet from RFC 3550 (Schulzrinne et al. 2003). The fields have the following meanings.

V The RTP version number, which is two for both RFC 3550 and RFC 1889.

P The padding bit. A one in this field indicates that there are padding bytes at the end of the packet.

X The extension bit indicates whether a header extension has been appended to the end of the fixed header.

CC CSRC count indicates the number of contributing source identifiers included after the fixed header. The field limits the number of CSRCs to 15.

M The marker bit is interpreted differently depending on the profile used.

PT The payload type field indicates which payload is carried by the packet.

Sequence number The sequence number, initialised to a random value, is incremented with each RTP packet sent. This enables the receiver to correctly order received packets.

Timestamp The RTP timestamp is initialised to a random number and is typically incremented at the sample rate of the media contained in the payload for audio encodings. The timestamp for each packet is that of the sampling time of the first byte in the payload.

Syncronization source (SSRC) identifier The SSRC identifies the source of the media sender (or receiver in RTCP packets). Note that a single implementation with multiple sources, e.g. audio and video, will have multiple SSRCs. The SSRC is randomly selected and must be unique within the RTP session.

Contributing source (CSRC) identifiers The CSRC list is created by RTP mixers to indicate which synchronisation sources contributed to the payload in the packet.

7.2.1.3 RTP receive

When a packet is received, it is checked to see whether it is a valid RTP packet and that there isn't an SSRC collision. Note that packets with a CSRC count (CC) field with a non-zero value will be silently discarded. The CC field indicates the presence of contributing sources (CSRC) that are inserted by RTP mixers. An RTP mixer will insert its own SSRC in the SSRC field and place the SSRCs of the contributors in the CSRC fields. Therefore the current implementation cannot work with RTP mixers. If the packet is from a new source then data pertaining to that source will be set. If the sequence number of the received packet is within an acceptable range the payload/s will be extracted and put into a FIFO buffer to be accessed by the decoder. The interarrival jitter is also calculated to be used in the RTCP packets.

7.2.2 Implementation

Excluding the codec interface tasks, RTP is comprised of three tasks (plus two RTCP tasks), an interrupt service routine, a number of functions, and a user command for test purposes. The three tasks are:

- RTP_tsk which provides the control for each session.
- RTP_rx_tsk which receives RTP packets and places their payloads in a FIFO buffer if the packets are valid. The buffer is the link between the Real-time Transport Protocol and the G.729A decoder.
- RTP_tx_tsk collects G.729 frames from a FIFO buffer linking the task to the G.729A encoder, builds the RTP packet and passes it to the transport layer for transmission.

The AWC86 uses little endian byte order while RTP packets are sent in network byte order (big endian). The RTP send and receive tasks will therefore do endian swaps before sending or after receiving RTP packets so that the correct byte order is maintained. The bit ordering in each byte is unaffected. The RTP implementation makes use of the sample code found in the RTP specification (Schulzrinne et al. 2003). The sample code used can be seen in appendix C.5.

7.2.2.1 RTP_tsk

RTP_tsk, like all other tasks, is implemented in an infinite loop as recommended by the MicroRTOS programmer's guide (AWC 2000). The task suspends itself at start-up. The task is called (resumed) by other tasks when certain events occur. Once the task has been resumed, it performs some checks before entering a switch which will determine its course of action based upon which event prompted the task to be resumed. The switch statement contains the following cases based on these events:

- Start session
- End session
- SSRC collision which is an end session followed by a start session
- Adjustment for altering the RTP session based on changes supplied by SIP. This has not been implemented but remains as a reminder for future development.

A default case is available to catch invalid event types.

7.2.2.1.1 Start session Information from the interface is passed to rtp_start_session which returns a boolean [1] value depending on whether the session could be properly initialised. A structure, RTP_session, containing all the state required for an RTP session is populated in the function. Random numbers are used to initialise the base sequence number and timestamp, and SSRC. The random number generator can be found in Algorithms in C (Sedgewick 1990). UDP 'sockets' are opened for sending RTP and RTCP packets. UDP receive 'sockets' are bound to the RTP and RTCP receive tasks using the UDPbind() function supplied by MicroRTOS. This allows the tasks only to be run when packets have been received and not run periodically to poll for incoming packets. A function to generate the unit's canonical name (CNAME) is called. This function will convert the unit's IP address into an ASCII string to be used as the canonical name. The canonical name will be explained in the RTCP section. A message box is created to allow the session data to be safely accessed by the other tasks. Next the necessary codec components and RTP and RTCP tasks are initialised based upon the type of operation required, e.g. RTP transmit only.

7.2.2.1.2 End session The RTP mode (e.g. RTP transmit only) is passed to the rtp_end_session function which returns a boolean indication of success. In the function, the codecs are disabled and the UDP 'sockets' are disabled except for the RTCP transmit socket. If an RTCP BYE packet needs to be sent then the RTCP transmit task is activated before disabling the UDP 'socket'. The required tasks are reset and the session data message box is deleted.

7.2.2.1.3 SSRC collision If an SSRC collision is detected then the rtp_end_session function is called followed by rtp_start_session.

7.2.2.2 RTP_rx_tsk

The RTP receive task is resumed when a packet has arrived at the specified port from the destination IP address. The RTP session data message box is located before working with the received packet. The compulsory part of the RTP header, i.e. the first 12 bytes, is extracted. The packet is silently discarded if the RTP version is not two, the payload type is not what is expected, or the CC field is not set to zero. The next step is dependant on the SSRC of the remote RTP source. If there is an SSRC collision RTP_tsk is resumed to handle the event. If the SSRC is new then the new source is initialised. If however the SSRC is known then the sequence number of the packet needs to be checked

[1] The C compiler used predates the 'boolean' keyword. Boolean here refers to a 0 / 1, FALSE / TRUE value.

using the function sequence_check. New sources are on probation until a set number of packets have arrived in sequence. The function returns FALSE if the source is still under probation. The function will increment the received packet count and return TRUE if the packet received is in order with a permissible gap or there has been a large gap followed by two consecutive packets. Duplicate or misordered packets are discarded by this implementation. Interarrival jitter is calculated after the sequence check. If there are one or more valid frames in the payload then they are placed in the FIFO buffer accessed by the decoder interface. The audio profile (Schulzrinne & Casner 2003) does not use header extensions, therefore any header extensions will be ignored, skipped over, before the payload.

The interarrival jitter is calculated using the fixed point version of the code in appendix A.8 of RFC 3550. This can be seen in appendix C.5 of this document. The timestamps of both units are incremented at 8KHz but are started from different initial values. The timestamp from the first received RTP packet is assumed to match the timestamp of the receiver less 20ms, i.e. the sampling instant of the first byte if no other delays are counted. Using this difference timestamps can be compared using the timestamp of the receiver task. Interarrival jitter will be explained further in chapter 8.

7.2.2.3 RTP_tx_tsk

Maskable interrupts are disabled while the RTP transmit task is running as a precaution against other maskable interrupts being triggered while the task is resumed, in particular the timestamp incrementing ISR. After the session data message box has been located and the end_task value has been read and it is not TRUE, then the FIFO buffer from the encoder is checked. If there are at least two frames in the buffer then an RTP packet is built with a payload of two G.729 frames and passed to the transport layer (UDP) to be sent. The timestamp from the first frame taken out of the buffer is used as the timestamp for the RTP packet. The sequence number and frame count are incremented accordingly. The padding, extension and CSRC count fields are all set to zero. The marker bit is only useful if silence suppression is implemented. The marker bit is always zero as the implementation sends a continuous stream of G.729 frames regardless of the input. This task will repeat itself every 20ms — the amount of encoded audio, measured as time, carried in each RTP packet.

7.2.2.4 TS_ISR

This interrupt service routine increments the RTP timestamp at 8KHz. This is the sample rate of G.729A, the audio payload. Timer 1, a 16 bit timer part of the AM186ES micro-

controller within the AWC86 module, is used to generate the interrupt. The timer 1 interrupt can be masked and so the RTP timestamp is not strictly monotonic.

Chapter 8

Quality of service

Although this system does not guarantee timely delivery or other factors associated with quality of service, it does however provide indications of the quality of service. This information is conveyed using the RTP control protocol (RTCP).

8.1 RTCP

RTCP is closely linked to the real-time transport protocol and is defined in the same document (Schulzrinne et al. 2003). The data carried in the RTCP packets is not directly used by this implementation.

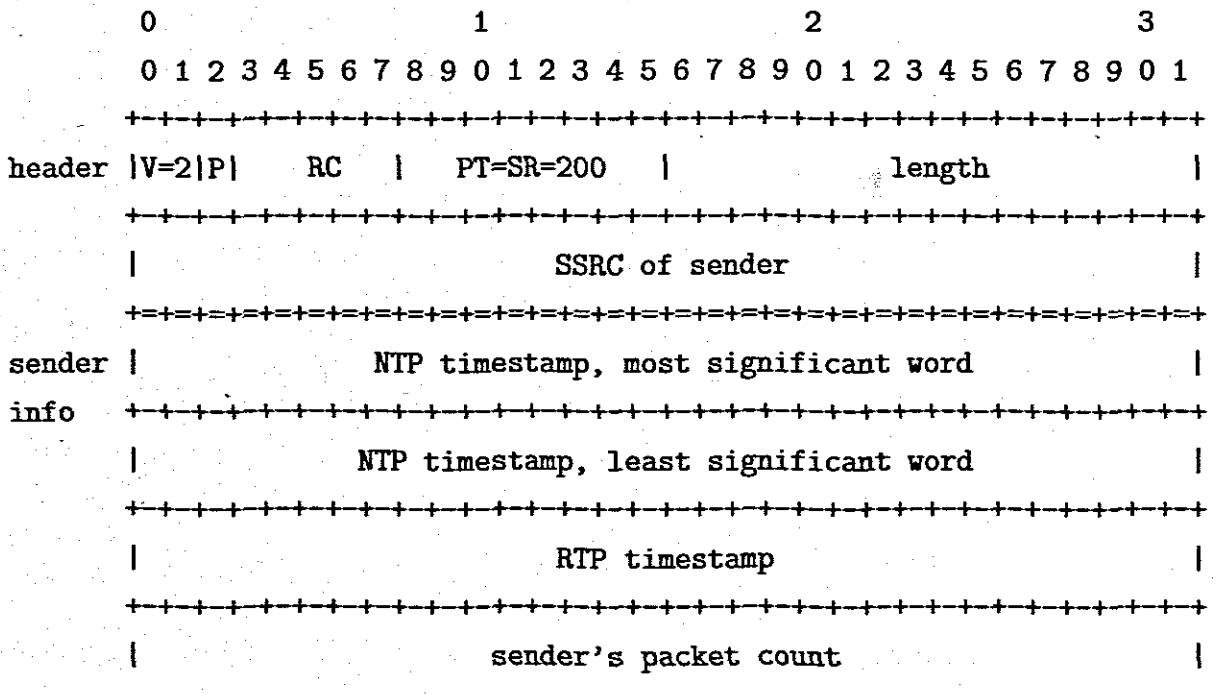
8.1.1 Functionality

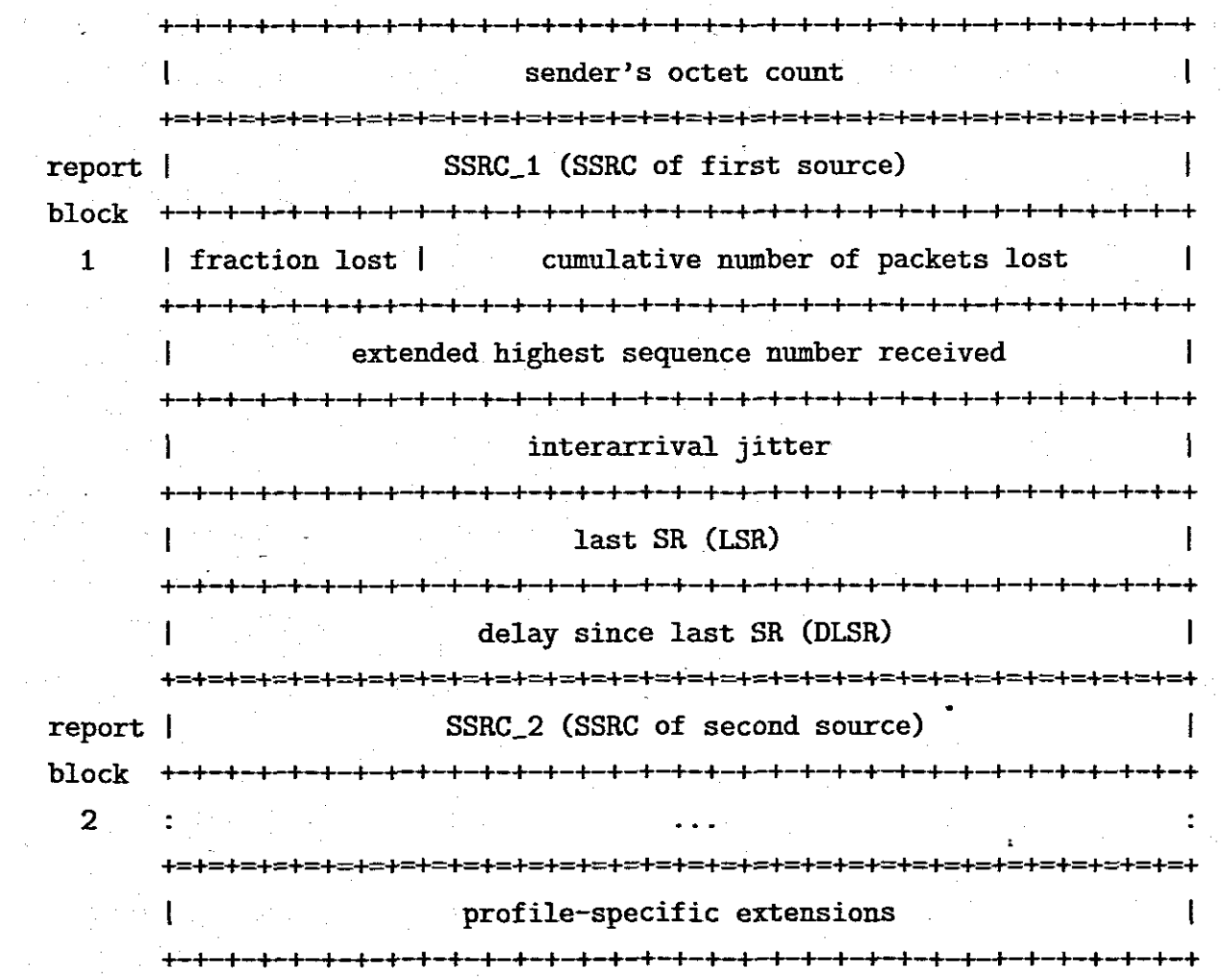
RTCP provides quality of service information through sender reports (SR) and receiver reports (RR). Source description packets (SDES) contain information about a source. The only compulsory SDES packet is the CNAME, or canonical name, a persistent transport level identifier for an RTP source (Schulzrinne et al. 2003). Bye packets indicate that a participant is leaving the session. Application-defined packets are also available as part of the RTCP specification. RTCP packets are sent as compound packets with at least a receiver report (or sender report) and an SDES packet containing the participant's canonical name. The canonical name can be used to identify a user if their SSRC changes or to provide a link between, for example, separate audio and video streams from the same host. The wallclock time, usually network time protocol (NTP) (Mills 1992) time, and RTP timestamp can be used to synchronise streams (Schulzrinne et al. 2003). RTCP packets are sent at approximately five second intervals for a two unit session, except the

first packet which is sent approximately 2,5 seconds after starting the RTP session. The intervals vary with the number of participants so that RTCP packets do not use up too much bandwidth and so that there aren't floods of packets at certain times. The RTCP interval scaling has not been implemented as the system has been designed specifically for a two user environment. The 2,5s and 5s intervals are randomized over a range of 0,5 to 1,5 times the interval value (e.g. $0,9 \times 5\text{s} = 4,5\text{s}$).

8.1.1.1 Sender reports

If the unit has sent an RTP packet in the time since two RTCP packets have been sent then a sender report forms the first part of an RTCP compound packet. An important field is the NTP timestamp. This is the wallclock time at which the packet was sent. A unit may use elapsed time from a common system clock if it has no concept of wallclock time or it may simply set the field to zero if it has no elapsed timer either. An RTP timestamp in the report corresponds to the time in the NTP timestamp. This relationship can be used for synchronisation purposes. Elapsed time was used in the implementation even though a real-time clock is available. The real-time clock is an external device with a serial interface. It was felt that reading the clock would consume unnecessary time. No other RTP implementations would be run concurrently on the unit so therefore it was not necessary to use a clock available to the whole system. The NTP timestamp has been implemented as the elapsed time in seconds since the base timestamp. Reception report blocks are appended to a sender report if any RTP packets have been received since the last report. These report blocks have the same fields as receiver reports. Below is the format of a sender report from RFC3550 (Schulzrinne et al. 2003).





V As in the RTP packets, the version is two.

P The padding bit as in RTP headers.

RC Reception report count indicates how many reception report blocks are contained in the packet.

PT The packet type is set to 200, the value for sender reports.

Length The length of the entire packet in 32-bit words less one.

SSRC of sender The SSRC of the source generating the packet.

NTP timestamp As explained above. The most significant word contains the seconds part of the timestamp while the least significant word contains the fractional part.

RTP timestamp This is the RTP timestamp matching the time in the NTP timestamp field.

Sender's packet count The number of RTP packets sent by the SSRC.

Sender's octet count The number of bytes of payload sent by the SSRC.

SSRC_1 (SSRC of first source) The SSRC of the first source in a reception report block if present. Reception report blocks are only built if RTP packets have been received from the source.

Fraction lost The fraction of RTP packets lost since the last RTCP packet was sent. The number is calculated as the number of packets lost divided by the number of packets expected. This value is stored as a fixed point number. Dividing the value stored by 256 will reveal the number in fractional form.

Cumulative number of packets lost The total number of packets lost since the start of the session.

Extended highest sequence number received Sequence numbers are 16-bit. The extended sequence number is a 16-bit word prepended to the sequence number to count sequence number rollovers.

Interarrival jitter The interarrival jitter is best explained in the RFC (Schulzrinne et al. 2003):

> An estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer. The interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the difference in the "relative transit time" for the two packets; the relative transit time is the difference between a packet's RTP timestamp and the receiver's clock at the time of arrival, measured in the same units.
>
> If Si is the RTP timestamp from packet i, and Ri is the time of arrival in RTP timestamp units for packet i, then for two packets i and j, D may be expressed as
>
> $$D(i,j) = (Rj - Ri) - (Sj - Si) = (Rj - Sj) - (Ri - Si)$$
>
> The interarrival jitter SHOULD be calculated continuously as each data packet i is received from source SSRC_n, using this difference D for that packet and the previous packet i-1 in order of arrival (not necessarily in sequence), according to the formula
>
> $$J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16$$

Whenever a reception report is issued, the current value of J is sampled.

Last SR The middle 32-bits from the NTP timestamp of the most recently received sender report.

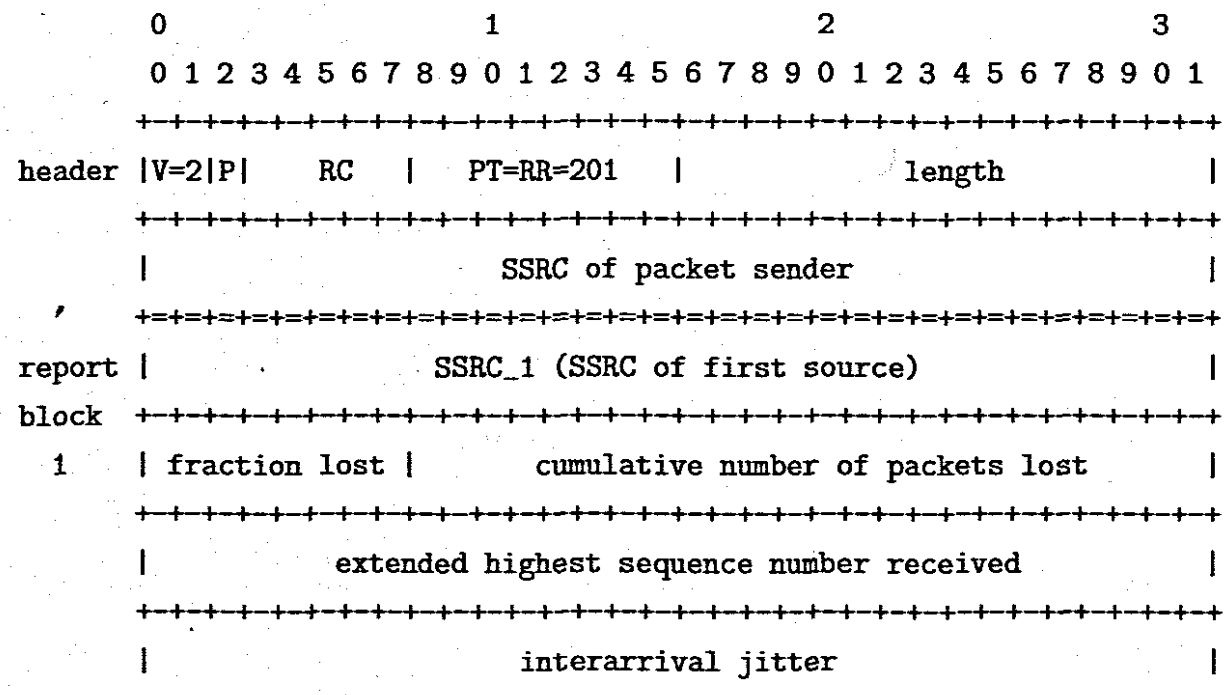
Delay since last SR The delay since the last SR was received. The value is expressed in units of $1/2^{32}$ seconds

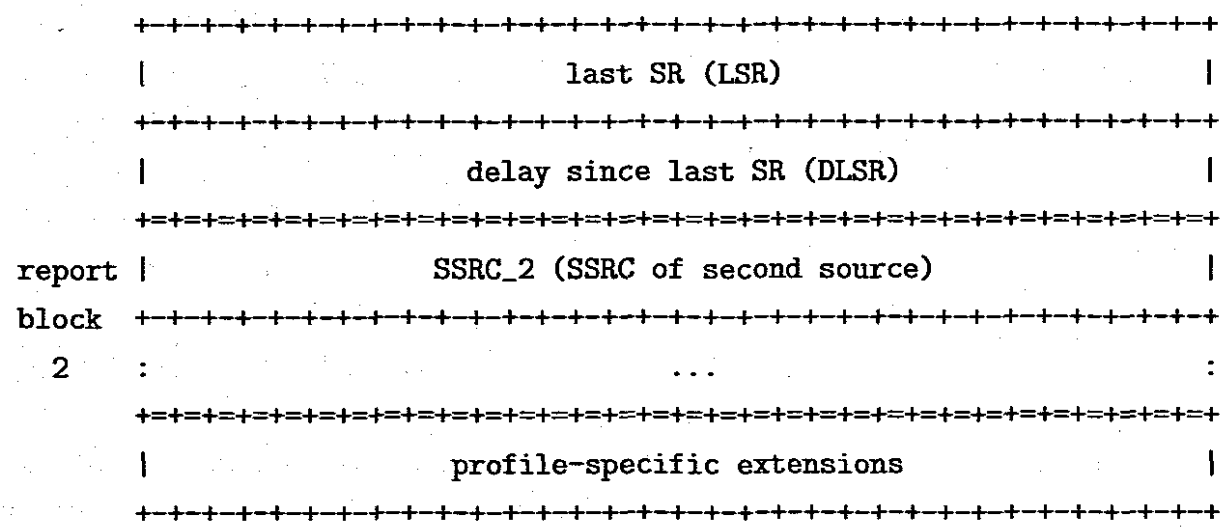
8.1.1.2 Receiver reports

Receiver reports contain information regarding real-time data received from a source. A subset of the information contained in the fields is:

- The number of RTP packets lost since the last RTCP packet was sent shown as a fraction using fixed point notation.
- The number of RTP packets that have not reached the sender of the report since the RTP session started.
- Interarrival jitter.
- The delay since receiving the last sender report from the source.

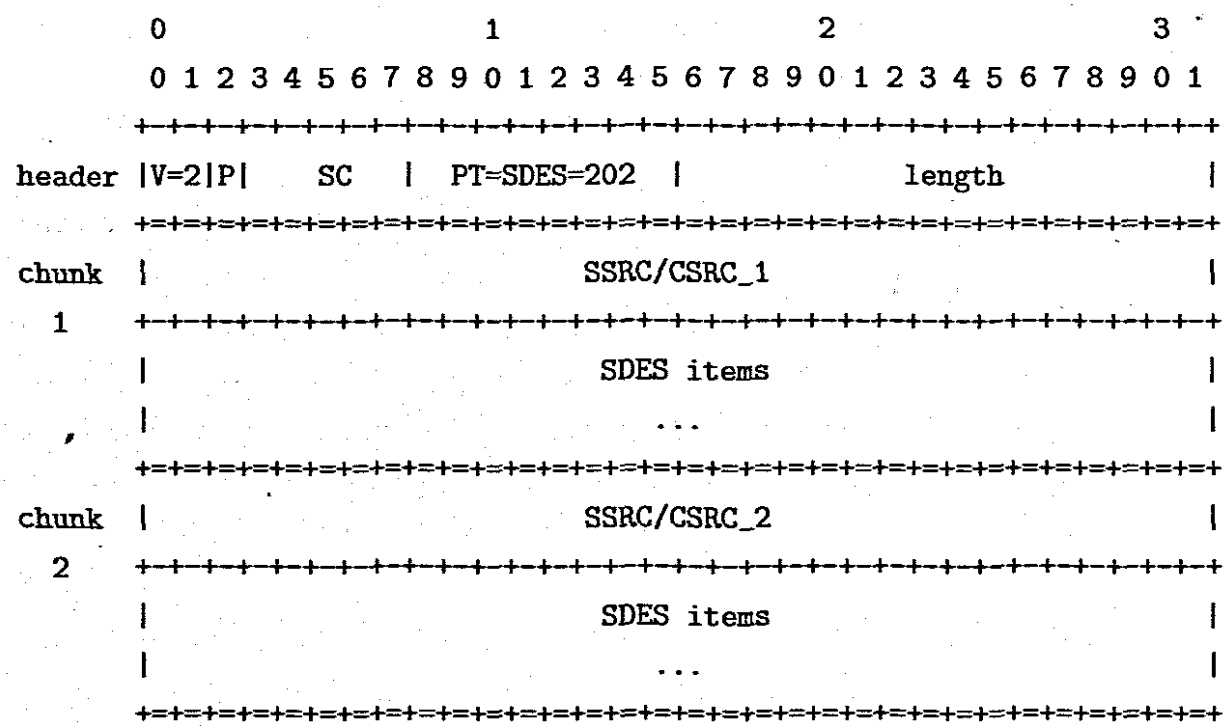
Below is the packet format for a receiver report from RFC 3550 (Schulzrinne et al. 2003). For an explanation of the fields see the section above on sender reports.

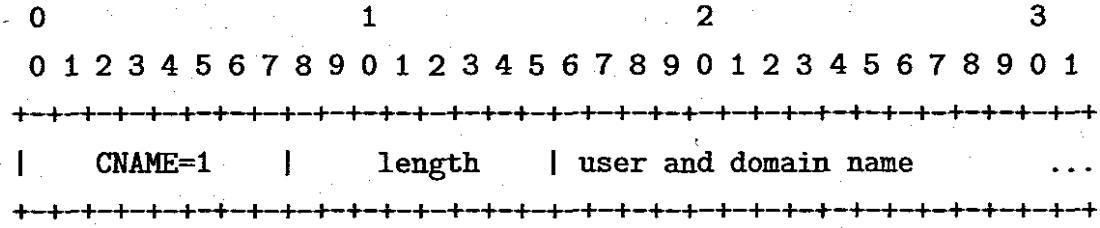




8.1.1.3 Source descriptions

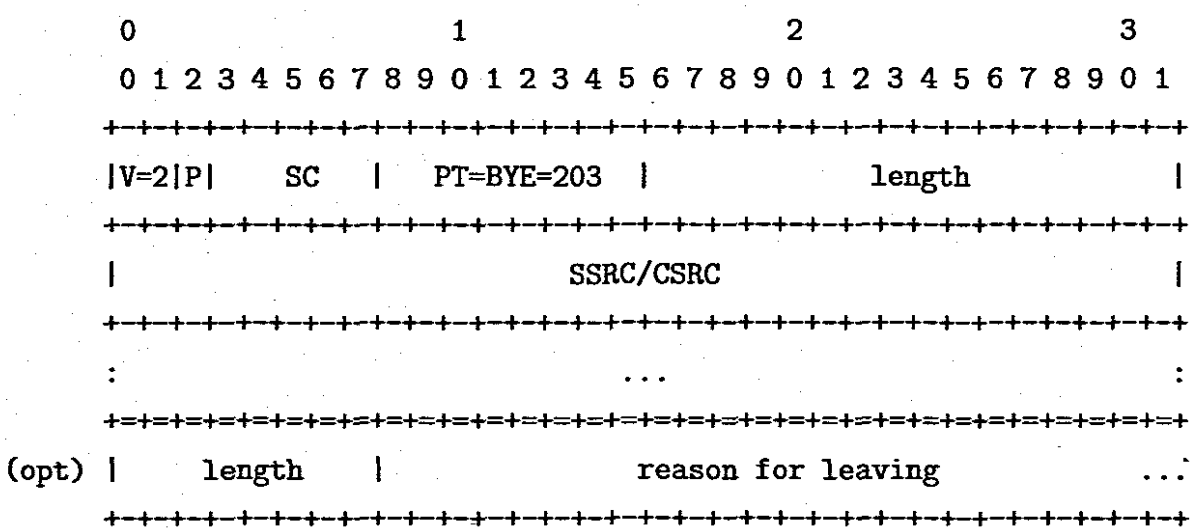
Source descriptions, as the name suggests, provide information about a source useful to other users such as their e-mail address. A source description packet can contain chunks for a number of sources. Each chunk can contain a number items describing the source like the e-mail address mentioned above. The only compulsory source description item is the canonical name identifier or CNAME. This is a name used to identify the end-point. It can be used to link multiple SSRCs to a particular user. Below are the formats of the SDES packet and the CNAME item from RFC 3550 (Schulzrinne et al. 2003). More information of the items available can be found in the RTP specification.





8.1.1.4 Bye packets

Bye packets are appended to a compound RTCP packet when a source wants to leave an RTP session. A single bye packet can be sent for multiple SSRCs and CSRCs. There is also an optional reason for leaving field which can be used to pass a text message explaining the reason for leaving to the receivers of the packet. The packet format below is from the RTP RFC (Schulzrinne et al. 2003).



8.1.2 Implementation

RTCP is closely linked to RTP and portions of the RTCP code are embedded in the RTP tasks and functions. RTCP comprises of two event handling tasks, namely RTCP_rx_tsk and RTCP_tx_tsk, and a number of functions which complete the RTCP implementation. Note that the RTCP implementation is somewhat limited as it was designed for a two source system that does not encrypt its RTCP packets.

8.1.2.1 RTCP_rx_tsk

The RTCP receive task is bound to the RTCP UDP port. It is resumed when a packet is received on the RTCP port (from the expected destination IP address). The received compound RTCP packet is tested to confirm its validity. The RTCP_validify function uses the check in appendix A.2 of the RTP specification (Schulzrinne et al. 2003). If the compound packet passes the validity test, it gets sent to the compound_parser function. The compound RTCP packet parser assumes that the packet has already passed the validity test and that the SSRC in the first packet is correct. If the SSRC differs in any other SR or RR packets within the compound packet then the whole packet will be considered invalid and discarded. If the SSRC in the SDES or bye packet does not match the remote SSRC then the packet is ignored. Note that most of the data carried in RTCP packets are not directly used within the implementation but can be monitored and analysed using external software like Ethereal (Ethereal 2004).

The function will loop through the compound packet parsing the component packets. Unsupported packet types are ignored. Sender reports, receiver reports, session description packets and bye packets are supported. The first packet is checked for any SSRC issues — collision, probation, new source. The RTP and NTP timestamps are extracted from the sender reports. The canonical name is obtained from the session description packet. All other session description items are ignored. The flag 'bye', initially set to zero is incremented if a bye packet is detected. The canonical name will be stored in the RTP session data structure if this is the first time a canonical name is received or it differs from the stored canonical name for the RTCP sender. The function will return the packet length in bytes or a code indicating that an SSRC collision has been detected or a bye has been received. The RTP receive task uses a switch statement to deal with the outcome of the validification and parsing processes. The packet will be discarded if it did not pass the validity test. RTP_tsk will be resumed if an SSRC collision is detected. Although loops are not detected, a case has been included for future development. The RTP_tsk will be resumed if a bye packet has been received and the default case handles normal non-bye operation. If an SSRC collision was detected in a compound packet containing a bye packet then operation will continue as normal. This is because a bye is sent if a collision is detected and the detector is about to change it's SSRC to a new value.

8.1.2.2 RTCP_tx_tsk

The RTCP transmit task has a switch statement to handle three events, the first packet to be sent, normal operation and the last packet to be sent. During the RTP session initialisation the RTCP transmit task will be resumed with the RTCP event set to FIRST. The first RTCP packet is sent a (randomized) 2,5 seconds after the session has begun. The reason

for the randomisation is to avoid a flood of RTCP packets from sources starting at the same time. Some variable initialisation occurs after the delay before the build_compound_rtcp function is called. Normal operation simply calls the build_compound_rtcp function and for the last packet a build_bye function is called after the build_compound_rtcp function.

build_compound_rtcp is really an intermediary function. It decides which function to call to build the first packet, i.e. build_SR or build_RR and then calls the function to build the session description packet which is appended to the previously built packet creating a compound RTCP packet. The length of the compound packet is the return value unless an error occurred in which case zero is returned.

The build_SR function first gathers and formats the sender information. The function build_report_block is called if a report block is required. This function only supports the inclusion of a single report block. This is a two user only implementation and therefore support for more than one report block is unnecessary. The common RTCP header components are populated and the packet is built. The packet length is returned to the calling function.

build_report_block compiles and formats data related to the remote sender. The formatted data is copied to the location provided by the calling function and the return value is the report block length.

The receiver report building function (build_RR) will determine whether a report block or simply an empty report (common header only) is required. build_report_block mentioned before is called to build the report block if required. The header is composed and the full packet is copied to the location provided by the calling function. Packet length is the returned value.

The build_SDES function will build a session description packet with a single chunk for the local source containing a single item — the canonical name. Again the packet length is the returned value.

The build_bye function will create a bye packet for a single SSRC and no contributing sources, i.e. only for the local source. The specification allows for an optional reason field. This was felt to be unnecessary and therefore not implemented.

Back in the RTCP_tx_tsk correctly formed packets need to be sent. If the function build_compound_rtcp returned zero — indicating that an error occurred — then no packet is sent. If there was an error building the bye packet the previous packets in the compound packet will still be sent. The RTCP event must then be set for normal operation. Next the task will check whether it should end the session due to a timeout. A timeout will occur if no RTP or RTCP packets have been received for TIMEOUT (typically set to

five) RTCP transmissions. The task will then suspend itself for a randomised five second delay before repeating the process.

8.1.2.3 Problems encountered

The RTP, RTCP and codec interface software were written as a single program. The implementation suffered from a number of memory related problems. The interrupt service routine problem mentioned in the codec interface section is an example of one of them. Insufficient documentation regarding the operating system and it's functions was a significant source of problems. Many of the functions are based on the standard C library functions. MemAlloc is the dynamic memory allocation function provided with MicroRTOS. The author was not provided with documentation on this function. It's equivalent available with the Borland compiler is farmalloc. The Borland C++ 4.52 help file states:

> 'Return Value
> farmalloc returns a pointer to the newly allocated block, or NULL if not enough space exists for the new block.'

MemAlloc however does not return null, or at least not the null expected. The author was testing for zero but the returned value in the case of MemAlloc failing turned out to be a 'relative null in an absolute address'. That is the segment part of the address was set to the segment in use and the offset set to zero. Therefore the returned value was never zero.

Chapter 9

Evaluation of design

9.1 Configuration

For much of the development process the hardware configuration shown in figure 9.1 was used. Unit one and two represent the Internet phone hardware. Each unit was attached via an RS-232 link to computer PC1. On PC1 terminal emulators, typically Hyperterminal, were run to communicate with the units via the RS-232 cable. A second computer, PC2, was used for monitoring, programming and FTPing the code to the units. An unswitched hub allowed PC2 to monitor all network traffic going through the hub. Port 1 on the hub, connecting it to the rest of the network is switched. Ethereal (Ethereal 2004) was used for capturing the network traffic.

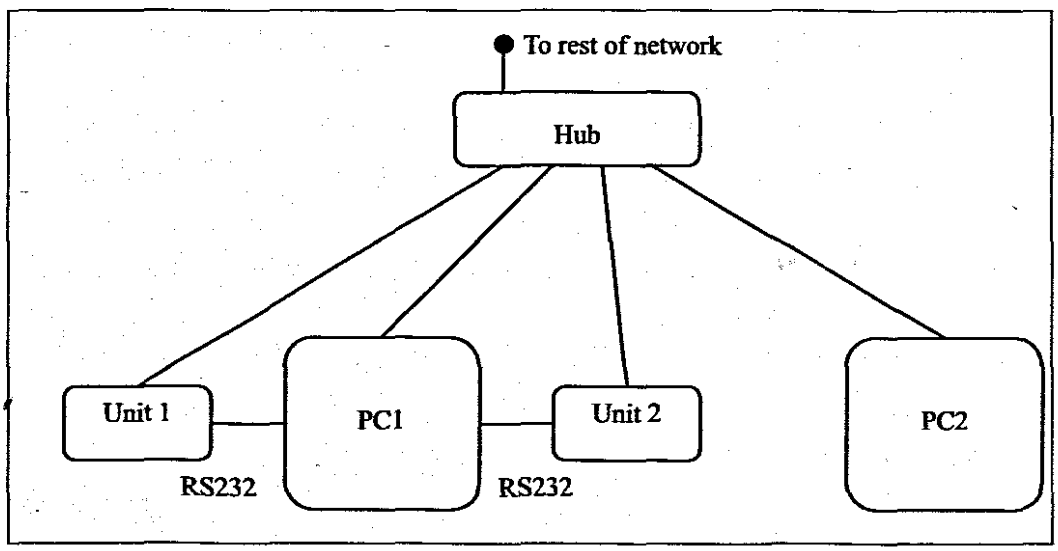


Figure 9.1: System test configuration.

9.2 Testing procedures

The testing procedures will be discussed where necessary in the following three sections. The signalling section was tested independently of the remaining sections — media transport and quality of service — which were tested together. The RTP program has a user interface similar to that described in the chapter on signalling.

9.3 SIP and SDP

A number of tests were run in order to determine whether the protocols were operating as specified. In early tests to determine whether the formation of requests and responses were correct, sections of code for building and analysing messages were transplanted to a computer program. This allowed errors to be detected and alterations to be made quicker than by running the program on the units. A java program called Messenger (or SIP messenger) was used to test correct packet formation and sequencing. Messenger is a simple program that will send and receive UDP packets and write their contents to the screen. One unit would be configured to communicate with SIPmessenger running on a computer (PC2 in figure 9.1). The transport protocol in the implementation was changed to UDP — with the retransmission timers ignored — and messages could be sent between the two user agents to check for correct operation. It appears that Messenger is no longer available. It was developed by Neil Deason of Ubiquity Software Corporation and was available for download from The SIP Center's test download area (*The SIP Center* 2003). After these tests, with the implementation tending towards it's intended operation, Ethereal was used for monitoring communications between the two units.

That Ethereal is able to detect the SIP messages sent between the units as being SIP messages and the SIP dissector parse them correctly is a good indication that the messages have been properly formed. The same goes for SDP descriptions. The values of certain fields still need to be manually checked to ensure that they satisfy the requirements.

The following SIP sequence indicates correct SIP operation for initialising and terminating a multimedia session. The user agent sending the INVITE is also the user agent sending the BYE message. The full packet capture can be seen in appendix B.5. Please note that this sequence, and the others in this document, has been edited to fit on the page. No values have been altered from the original file.

```
No. Time       Source           Destination      Protocol Info
 11 1.926308   155.238.33.245   155.238.33.241   SIP/SDP  Request: INVITE
```

```
                                                    sip:com2@ctech.ac.za,
                                                          with session
                                                          description
 13 1.945562   155.238.33.241   155.238.33.245   SIP      Status: 180
 23 4.486158   155.238.33.241   155.238.33.245   SIP/SDP  Status: 200 ,
                                                          with session
                                                          description
 25 4.506274   155.238.33.245   155.238.33.241   SIP      Request: ACK
                                                       sip:com2@ctech.ac.za
 57 13.929440  155.238.33.245   155.238.33.241   SIP      Request: BYE
                                                       sip:com2@ctech.ac.za
 59 14.124954  155.238.33.241   155.238.33.245   SIP      Status: 200
```

The following is the sequence as above with the second user agent terminating the call. The captured packets can be seen in appendix B.6.

```
No. Time       Source           Destination      Protocol Info
 12 3.194150   155.238.33.245   155.238.33.241   SIP/SDP  Request: INVITE
                                                    sip:com2@ctech.ac.za,
                                                          with session
                                                          description
 14 3.219308   155.238.33.241   155.238.33.245   SIP      Status: 180
 20 5.362120   155.238.33.241   155.238.33.245   SIP/SDP  Status: 200 ,
                                                          with session
                                                          description
 22 5.384534   155.238.33.245   155.238.33.241   SIP      Request: ACK
                                                       sip:com2@ctech.ac.za
 68 14.408346  155.238.33.241   155.238.33.245   SIP      Request: BYE
                                                       sip:com1@ctech.ac.za
 72 14.611209  155.238.33.245   155.238.33.241   SIP      Status: 200
```

The packet structures can be seen in the appendices referenced in the above two SIP sequences. Only the application layer messages have been expanded by the dissector as the structure of the lower layers is not relevant here and their inclusion would clutter the captures unnecessarily. All the network layers can be seen in the hexadecimal data. The following section refers to the messages in appendix B.5 and B.6.

The implementations listen for packets on the default SIP port of 5060. This can be seen as the destination port number in the INVITE messages and the source port in the responses to the INVITEs. The via branch parameter begins with the string 'z9hG4bK'

Transaction	Appendix B.5	Appendix B.6
INVITE	z9hG4bK63	z9hG4bK204
ACK	z9hG4bK155	z9hG4bK211
BYE	z9hG4bK53	z9hG4bK5

Table 9.1: Via branch parameters for the transactions in appendix B.5 and appendix B.6

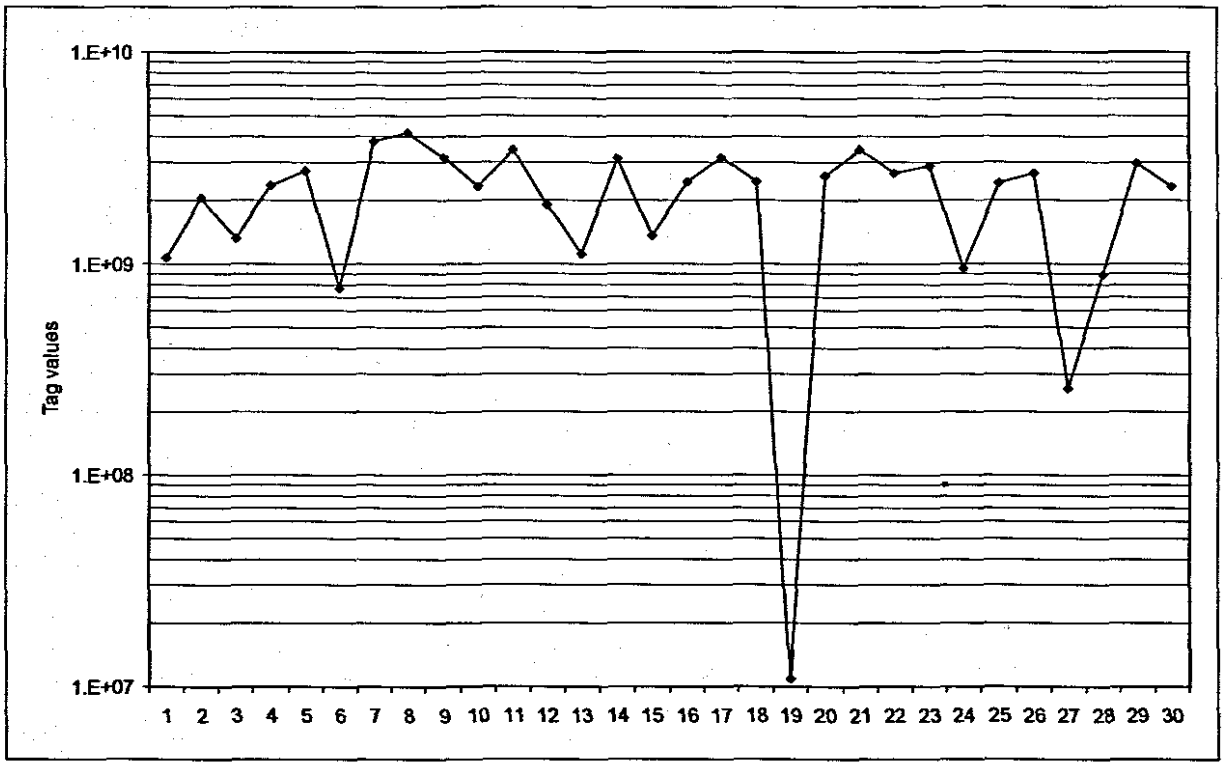


Figure 9.2: Graph showing the tags (Y axis) generated by both units over 15 dialogues.

indicating that RFC 3261 was used for development. The following few digits complete the branch. The branch parameter is used to identify a transaction (Rosenberg et al. 2002). Table 9.1 shows the branch values for the transactions in the two SIP captures in the appendices.

Dialogues are identified by the call-ID of the request that lead to the dialogue, a remote tag and a local tag (Rosenberg et al. 2002). The number preceding the URI in the call-ID field is randomly generated as are the tags. Figure 9.2 shows the tags generated by each unit over 15 dialogues.

In chapter 6.2.2 there is a sample capture of an SDP description which shows it's correct formation. The use of SDP in the offer / answer model can be seen in the captures in appendices B.5 and B.6. Appendix B.7 shows the SDP offer and answer in the relevant messages with the offer having 'sendonly' as the stream type attribute. The implementation currently only supports the initial INVITE for carrying the offer and the 200 response for the answer. To illustrate SDP's usage, the offer and answer from appendix B.5 is shown below as raw text.

```
v=0\r\n
o=com1 335094308 335094308 IN IP4 155.238.33.245\r\n
s=-\r\n
c=IN IP4 155.238.33.245\r\n
t=0 0\r\n
a=sendrecv\r\n
m=audio 5004 RTP/AVP 18\r\n
a=rtpmap:18 G729/8000\r\n

v=0\r\n
o=com2 211685113 211685113 IN IP4 155.238.33.241\r\n
s=-\r\n
c=IN IP4 155.238.33.241\r\n
t=0 0\r\n
a=sendrecv\r\n
m=audio 5004 RTP/AVP 18\r\n
a=rtpmap:18 G729/8000\r\n
```

The network data in the origin and connect fields are always the same as the units are never invited to join a session by a third party. Each unit generates it's own session ID and version as all values in the origin field pertain to the originator of the description. In this capture the offerer suggests bidirectional media flow and the answerer agrees.

9.4 RTP and RTCP

Ethereal was used to capture the RTP and RTCP packets on the network, which were analysed to determine whether they had been correctly formed. Below is a captured RTP packet with the RTP packet dissected by Ethereal. The RTP packet begins at address 0x2A and ends at address 0x49 in the hexadecimal data.

```
Frame 6 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
User Datagram Protocol, Src Port: 5004 (5004), Dst Port: 5004 (5004)
Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
```

Sample	Sequence number	Timestamp	SSRC
1	59545	330486760	815473930
2	41896	1801342234	3846146185
3	16240	258671283	3925913124
4	20804	974118520	3074051900
5	37647	1288246404	1622486497

Table 9.2: Sample of initial RTP values.

```
...0 .... = Extension: False
.... 0000 = Contributing source identifiers count: 0
0... .... = Marker: False
.001 0010 = Payload type: ITU-T G.729 (18)
Sequence number: 16240
Timestamp: 258671283
Synchronization Source identifier: 3925913124
Payload: 9731A004005F2B2A80349651012508FF...
```

```
0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  00 3c 14 00 00 00 80 11 aa ee 9b ee 21 f5 9b ee   .<..........!...
0020  21 f1 13 8c 13 8c 00 28 d0 1a 80 12 3f 70 0f 6b   !.......(....?p.k
0030  02 b3 ea 00 ae 24 97 31 a0 04 00 5f 2b 2a 80 34   .....$.1..._+*.4
0040  96 51 01 25 08 ff 85 20 1a 58                     .Q.%... .X
```

The default port number for RTP is 5004, which is both the source and destination port in the above packet. Port numbers for RTP are even with the RTCP port number being the RTP port plus one (Schulzrinne & Casner 2003). The RTP packet can be seen to be correctly formed as Ethereal is able to dissect it. The RTP version number is the same for both RFC 1889 and RFC 3550 (Schulzrinne et al. 2003). The payload type of 18 indicates that the payload contains G.729 data (Schulzrinne & Casner 2003). G.729 annex A is identical to G.729 on the wire and therefore is indicated by a payload type of 18. The packet has twenty bytes of payload as two G.729 frames are carried in each packet. The above packet is the first RTP packet in a stream. It can be seen that the sequence number, timestamp and SSRC do not start at predictable values. Table 9.2 shows a small selection of actual initial sequence, timestamp and SSRC values.

Below are five consecutive RTP packets from sample 2 in table 9.2. The 'delta t' field shows the difference in time in seconds since the last captured packet. Packets were captured in promiscuous mode without any filters therefore the number field, as in the previous captures, indicates the packet's place since the capture started which would potentially include unrelated packets. From the sequence of packets it can be seen that the payload type is set to the correct value, the SSRC remains constant and the

sequence numbers increase by one with each packet. It can be seen that the timestamp is incremented by 154 with each packet. This indicates that there is a 19,25ms delay between every second read from the encoder assuming a monotonically incremented timestamp. The timestamp is incremented at 8KHz — the sample rate of G.729. Images from the oscilloscope (appendix C.2) show that G.729 frames are made available at regular intervals, therefore according to the timestamp values a frame is read from the codec every 9,625ms.

No.	Delta t	Source	Destination	Protocol	Info
16	0.020524	155.238.33.245	155.238.33.241	RTP	Payload type=ITU-T G.72 SSRC=3846146185, Seq=41904, Time=1801343467
17	0.020522	155.238.33.245	155.238.33.241	RTP	Payload type=ITU-T G.72 SSRC=3846146185, Seq=41905, Time=1801343621
18	0.020522	155.238.33.245	155.238.33.241	RTP	Payload type=ITU-T G.72 SSRC=3846146185, Seq=41906, Time=1801343775
19	0.018471	155.238.33.245	155.238.33.241	RTP	Payload type=ITU-T G.72 SSRC=3846146185, Seq=41907, Time=1801343929
20	0.020529	155.238.33.245	155.238.33.241	RTP	Payload type=ITU-T G.72 SSRC=3846146185, Seq=41908, Time=1801344083

The graph in figure 9.3 shows the variation in time between RTP packets as detected by Ethereal. This capture is representative of all captures.

Each RTP packet is 32 bytes long without any extensions or contributing sources — a 12 byte header and two 10 byte G.729A frames. The RTP packets are carried in UDP packets whose header adds an extra 8 bytes. The Internet Protocol packets have a 20 byte header, which combined with it's 40 byte payload set the packet size from the network layer up to 60 bytes. Ignoring data link layer protocols the implementation requires a bandwidth of 24Kbps to transmit RTP packets.

$$Bandwidth\,(bps) = packet\ size\,(b) \times frequency^{1}$$

[1]Packets per second

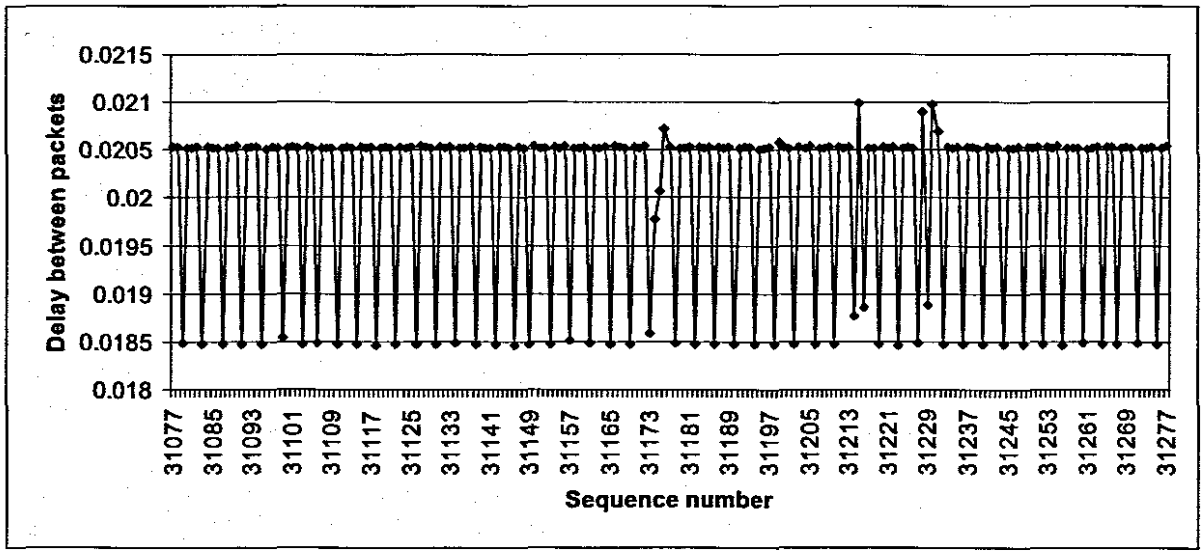


Figure 9.3: Graph showing the intervals between RTP packets as captured by Ethereal. The delay between packets is measured in seconds.

Below is an RTCP compound packet consisting of a receiver report, a source description with a single chunk containing a single canonical name item, and a bye packet. It can be seen to be correctly formed and has been dissected by Ethereal. The RTCP packets are being sent to and from port 5005, the standard RTCP port.

```
Frame 2142 (110 bytes on wire, 110 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                   Dst Addr: 155.238.33.245 (155.238.33.245)
User Datagram Protocol, Src Port: 5005 (5005), Dst Port: 5005 (5005)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Reception report count: 1
    Packet type: Receiver Report (201)
    Length: 7
    Sender SSRC: 3295023430
    Source 1
        Identifier: 2867170411
        SSRC contents
            Fraction lost: 41 / 256
            Cumulative number of packets lost: 94
        Extended highest sequence number received: 32627
            Sequence number cycles count: 0
            Highest sequence number received: 32627
        Interarrival jitter: 7
```

```
        Last SR timestamp: 2456813
        Delay since last SR timestamp: 0
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 6
    Chunk 1, SSRC/CSRC 3295023430
        Identifier: 3295023430
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.241
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Goodbye (203)
    Length: 1
    Identifier: 3295023430

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  00 60 1c 00 00 00 80 11 a2 ca 9b ee 21 f1 9b ee   .`..........!...
0020  21 f5 13 8d 13 8d 00 4c 91 5f 81 c9 00 07 c4 66   !......L._.....f
0030  11 46 aa e5 8c 6b 29 00 00 5e 00 00 7f 73 00 00   .F...k)..^...s..
0040  00 07 00 25 7c ed 00 00 00 00 81 ca 00 06 c4 66   ...%|..........f
0050  11 46 01 0e 31 35 35 2e 32 33 38 2e 33 33 2e 32   .F..155.238.33.2
0060  34 31 00 00 00 00 81 cb 00 01 c4 66 11 46         41.........f.F
```

For some reason, as can be seen in the above packet, the receiver is dropping RTP packets. Ninety four have been lost since the start of the session with 40 since the last RTCP packet was sent. The highest sequence number received is from the most recent RTP packet and the highest sequence number received in the previous receiver report is 32383 also from the most recent RTP packet. At the time of the previous receiver report 54 packets had been lost. Therefore 40 packets have been lost since the last RTCP packet was sent. Using the formula for calculating the fraction lost reveals the answer to be 0,1639 or 41 in fixed point notation according to the method in RFC 3550. The lost packets are a concern since Ethereal shows that no packets are being dropped. Therefore the problem resides at the receiver and may be caused by the RTP receiver task being pre-

empted often. Below is the previously received sender report represented as hexadecimal data preceded by the timestamp in decimal. The timestamp can be seen at the address $32_{(hex)} - 39_{(hex)}$, i.e. $25_{(hex)} = 37_{(decimal)}$ and $7ced0000_{(hex)} = 2095906816_{(decimal)}$.

```
    Timestamp, MSW: 37
    Timestamp, LSW: 2095906816

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  00 5c b9 07 00 00 80 11 05 c7 9b ee 21 f5 9b ee   .\..........!...
0020  21 f1 13 8d 13 8d 00 48 89 74 80 c8 00 06 aa e5   !......H.t......
0030  8c 6b 00 00 00 25 7c ed 00 00 cc 32 58 2e 00 00   .k...%|....2X...
0040  07 9c 00 00 98 30 81 ca 00 06 aa e5 8c 6b 01 0e   .....0.......k..
0050  31 35 35 2e 32 33 38 2e 33 33 2e 32 34 35 00 00   155.238.33.245..
0060  00 00 81 cb 00 01 aa e5 8c 6b                     .........k
```

Therefore it can be seen that the value of the last SR timestamp is correct. The delay since the last SR timestamp is zero as the packet has been built immediately after having received a BYE compound packet from the sender and the implementation of the calculation for the delay only has millisecond resolution. It is difficult to determine whether the interarrival jitter values are correct. The jitter calculations are done using the fixed point code from appendix C.5. It has been assumed that the interarrival jitter is reasonably accurate.

The following is a compound RTCP packet consisting of a sender report and source description. It is taken from the same capture as the previously analysed RTCP packet. The first RTP packet sent has a sequence number of 30680 and the last RTP packet sent prior the the RTCP packet being build is 332520. Therefore 1841 RTP packets have been sent so far during this RTP session. This matches the value in the sender's packet count field. Each RTP packet contains two G.729 frames, each of ten bytes. Therefore the sender's octet count (counting payload size only) is the packet count $\times$ 20. In this case 36820.

```
Frame 2027 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
User Datagram Protocol, Src Port: 5005 (5005), Dst Port: 5005 (5005)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
```

```
    ...0 0000 = Reception report count: 0
    Packet type: Sender Report (200)
    Length: 6
    Sender SSRC: 2867170411
    Timestamp, MSW: 35
    Timestamp, LSW: 1889730560
    RTP timestamp: 3425835056
    Sender's packet count: 1841
    Sender's octet count: 36820
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 6
    Chunk 1, SSRC/CSRC 2867170411
        Identifier: 2867170411
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.245
```

```
0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  00 54 4d 07 00 00 80 11 71 cf 9b ee 21 f5 9b ee   .TM.....q...!...
0020  21 f1 13 8d 13 8d 00 40 97 b3 80 c8 00 06 aa e5   !.......@........
0030  8c 6b 00 00 00 23 70 a3 00 00 cc 32 18 30 00 00   .k...#p....2.0..
0040  07 31 00 00 8f d4 81 ca 00 06 aa e5 8c 6b 01 0e   .1...........k..
0050  31 35 35 2e 32 33 38 2e 33 33 2e 32 34 35 00 00   155.238.33.245..
0060  00 00                                            ..
```

Each source description packet in this implementation consists of a single chunk with a single item — the canonical name. Each item in a chunk contains a length field and as such items are placed contiguously without any separators. The length field is used to reach the item boundaries. Chunks must start on a 32-bit boundary. A zero value byte is appended to the end of a list of items in a chunk to indicate its end. As chunks must start on a 32-bit boundary additional zero (padding) bytes may be required. The padding bit in the header refers to padding other than the end of chunk padding. Therefore it can be seen in the above packet (see the hexadecimal data) that the padding bit is zero even though there are four zero bytes (32 bits) at the end of packet. The item text is not null terminated (Schulzrinne et al. 2003).

Netcat (@stake 2004), a command line networking utility, was used to test a unit's reaction to an SSRC collision without modifying the unit's code. A unit was configured as a listener in half duplex mode with the destination address set to that of PC2 (see figure 9.1). This meant that the unit only sent RTCP packets. Once a packet had been received, it's SSRC was read from Ethereal and copied into an RTCP packet from a previous capture using a hexadecimal editor. The packet with the matching SSRC was then sent using netcat to the unit. The unit acted correctly by sending a compound RTCP packet containing a BYE packet from it's current SSRC before generating a new SSRC by restarting. Appendix C.3 shows four consecutive RTCP packets; the packet from the unit before the collision is detected, the packet with the matching SSRC, the BYE response indicating that the collision has been detected, and the next RTCP packet with a new SSRC. Note that in the packet that caused the SSRC collision it was only necessary to alter the SSRC in the first packet (receiver report) as the implementation only checks the SSRC in the first packet of a compound packet. This is a two user only, single media system so it is safe to assume that all SSRCs in a compound packet are the same. The SSRC collision response code will cause the sequence number and timestamp to be altered as well as the SSRC. This is irrelevant as the detection of a new SSRC at the other unit will cause it to update all the necessary state. The unit receiving the bye compound packet is programmed to continue as normal if it detects an SSRC collision in a compound bye packet. Note that the compound RTCP packets have empty receiver reports as the unit has not received any RTP packets.

The RTP specification recommends that two user, unicast implementations use randomised RTCP transmission intervals but may ignore the other transmission interval calculating algorithms. Only the randomised transmission intervals have been implemented. The interval is randomised over the range 0.5T and 1.5T, where T is the set interval. T has a value of 5 under normal operation and 2.5 for the initial value. A BYE packet is sent immediately when a user wishes to end an RTP session. The following two graphs — figures 9.4 and 9.5 — show the randomised delay of RTCP packets. BYE packets are not shown. The first graph shows both sides sending RTCP packets only (receiver reports), while the second shows one side as a sender and the other a receiver. The times for the initial packets are measured from the sending of the address resolution protocol (ARP) packet by one of the units.

The session will time out if no RTP or RTCP packets are received for TIMEOUT RTCP intervals. The constant TIMEOUT has been set to five — the suggested value. No BYE packet is sent as the other unit has been somehow disabled. The sequence of packets in appendix C.4 show that the unit will time out after sending five RTCP packets without receiving any packets. After the fifth RTCP packet, greater than 7.5s — 1.5×5s — is elapsed without another RTCP packet being captured proving that the unit has in fact timed out. The RTCP packet numbers in appendix C.4 are 33, 54, 79, 103, and 114.

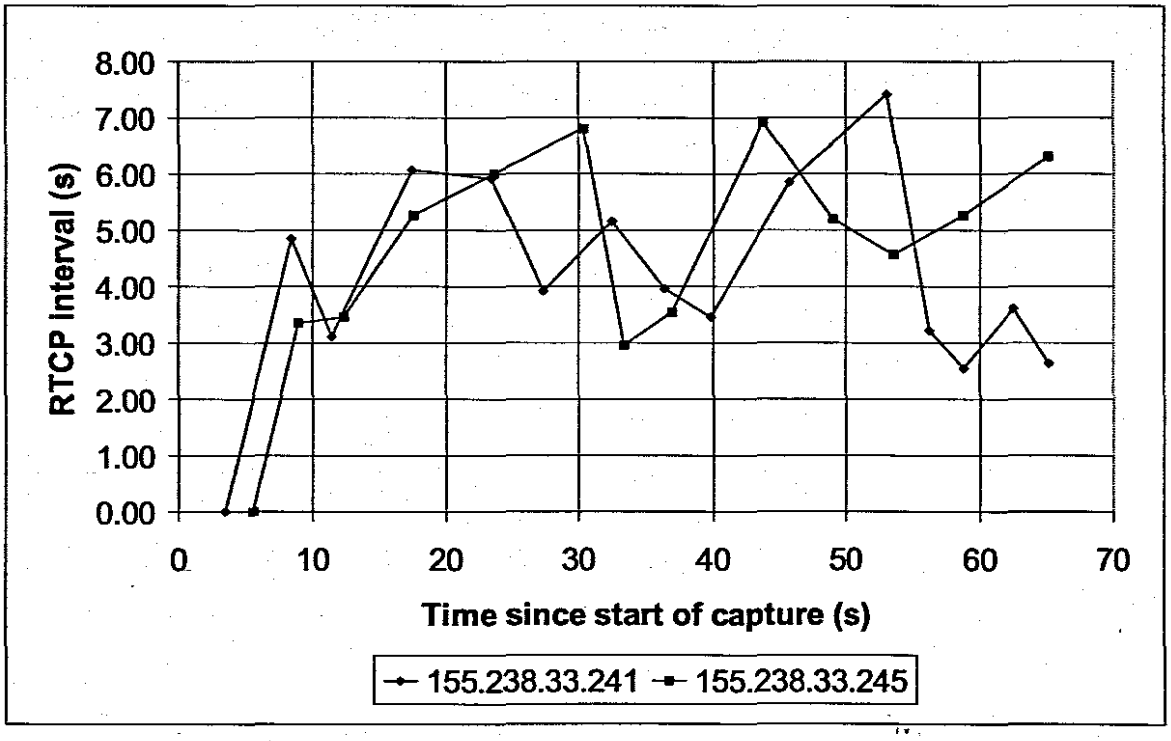


Figure 9.4: This graph shows the delay between RTCP packets by both units over the length of the session. Both units are sending receiver reports only. During this session the unit with the IP address 155.238.33.241 sent 15 packets while the other unit only sent 13.

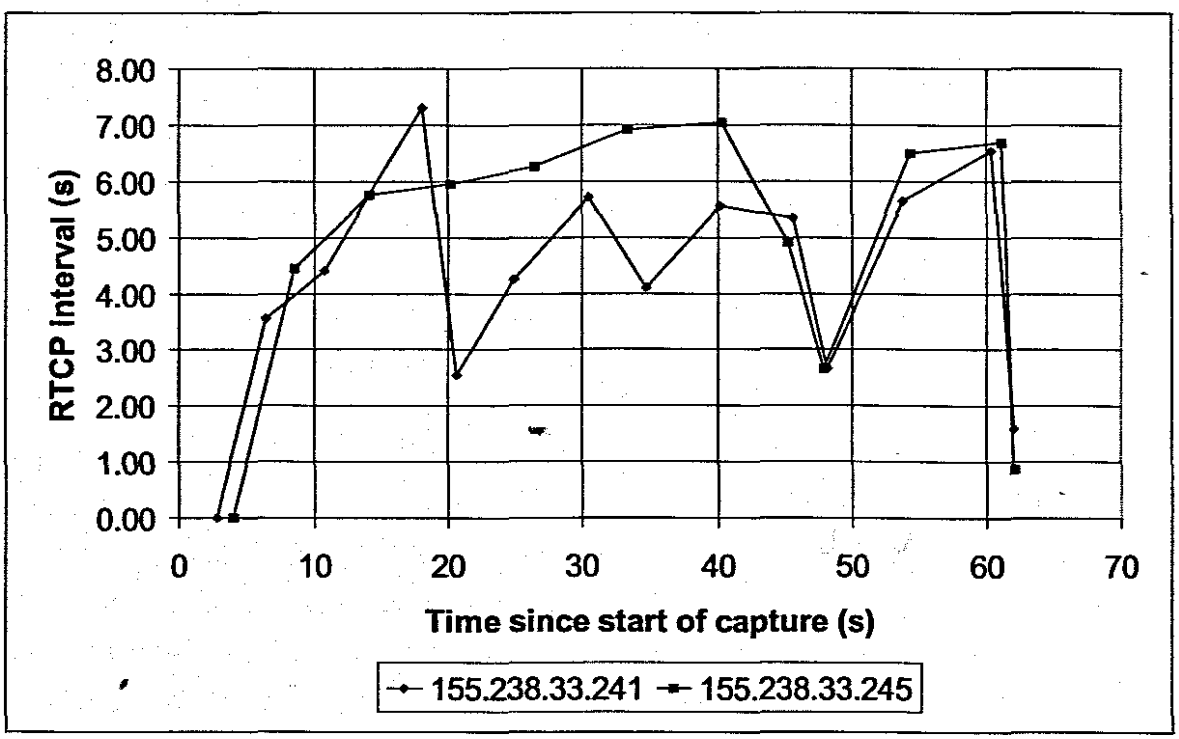


Figure 9.5: The delay between RTCP packets by both units during an RTP session. This session was of half duplex communication therefore one unit (155.238.33.241) is sending receiver reports and the other (155.238.33.245) sender reports. The final RTCP packets contain bye packets. 155.238.33.245 sent the first bye compound packet 0,88s after having sent its last RTCP packet. 155.238.33.241, also terminating, replies with a bye immediately after receiving the bye from 155.238.33.245. This bye packet is sent 1,59s after it's last RTCP packet was sent.

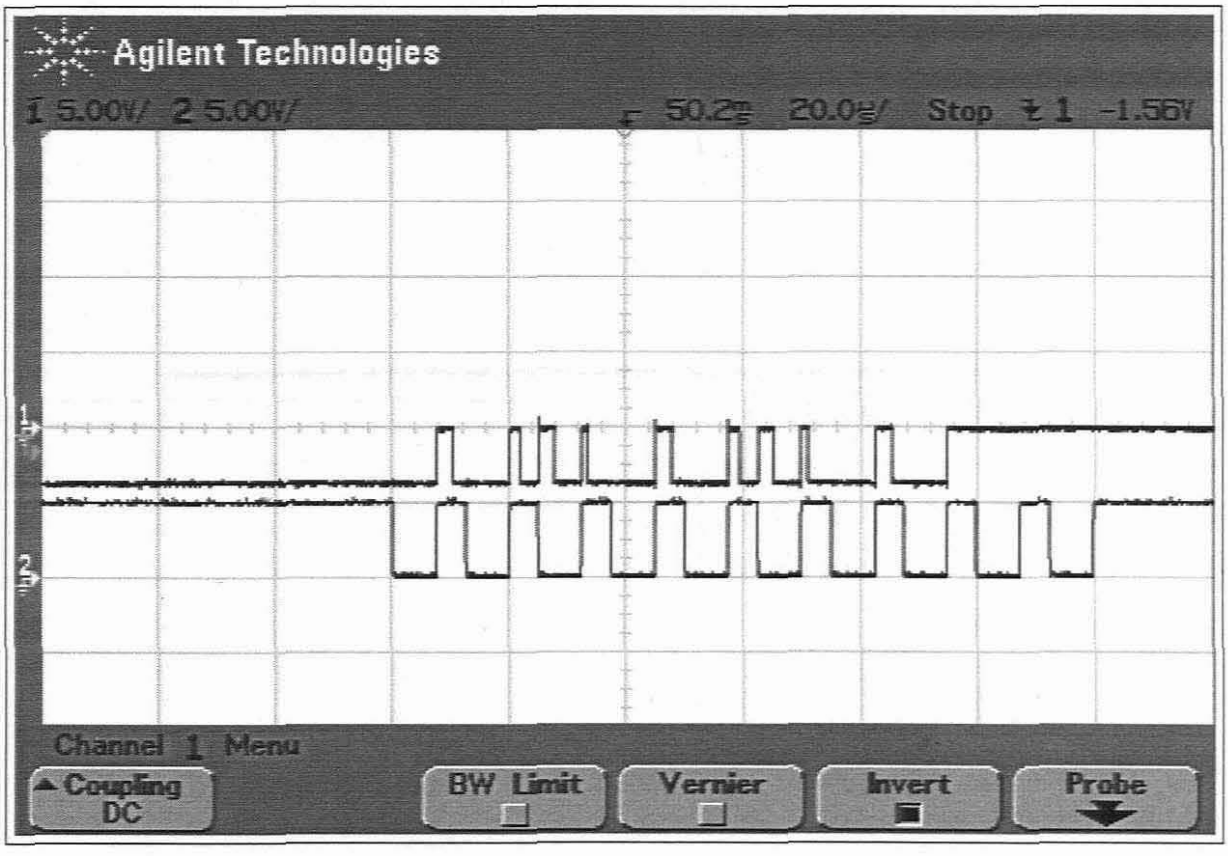


Figure 9.6: The oscilloscope capture shows the erroneous triggering of of the $\overline{\text{RTW}}$ line (channel one) when outputting a data frame from the codec. $\overline{\text{PCS}}$ is channel two. This capture was taken with the probes attached to the controller board after the $\overline{\text{RTW}}$ line has been inverted. Therefore channel one has been inverted to show how the signal would look at the codec.

9.5 G.729 Annex A

The codec section proved to be problematic. Five codec boards were made but only two proved to be usable. The PCB design and schematic have been checked by the author without any errors being detected. The boards comprise predominantly of surface mount components. Only the three ICs on each board were soldered professionally. The remaining components were all hand soldered without the benefit of surface mount tools. The very small 0603 package was chosen for the resistors and capacitors making soldering by hand very difficult. It is suspected that this soldering or handling of the codec ICs may have contributed to the problem. The author was only able to test half duplex operation with the two codec boards. The I^2C interface on both units was operational and could be used to configure the codecs. The parallel interface on the encoder board was causing problems in that the $\overline{\text{RTW}}$ pin would trigger on the rising and falling edge of some $\overline{\text{PCS}}$ pulses and so the data transferred to the controller was not necessarily the data intended for it. Figure 9.6 shows the problem. See figure 7.6 for the timing diagram.

Figure 9.7 shows the correct handshaking for the transfer of a frame using $\overline{\text{RTW}}$

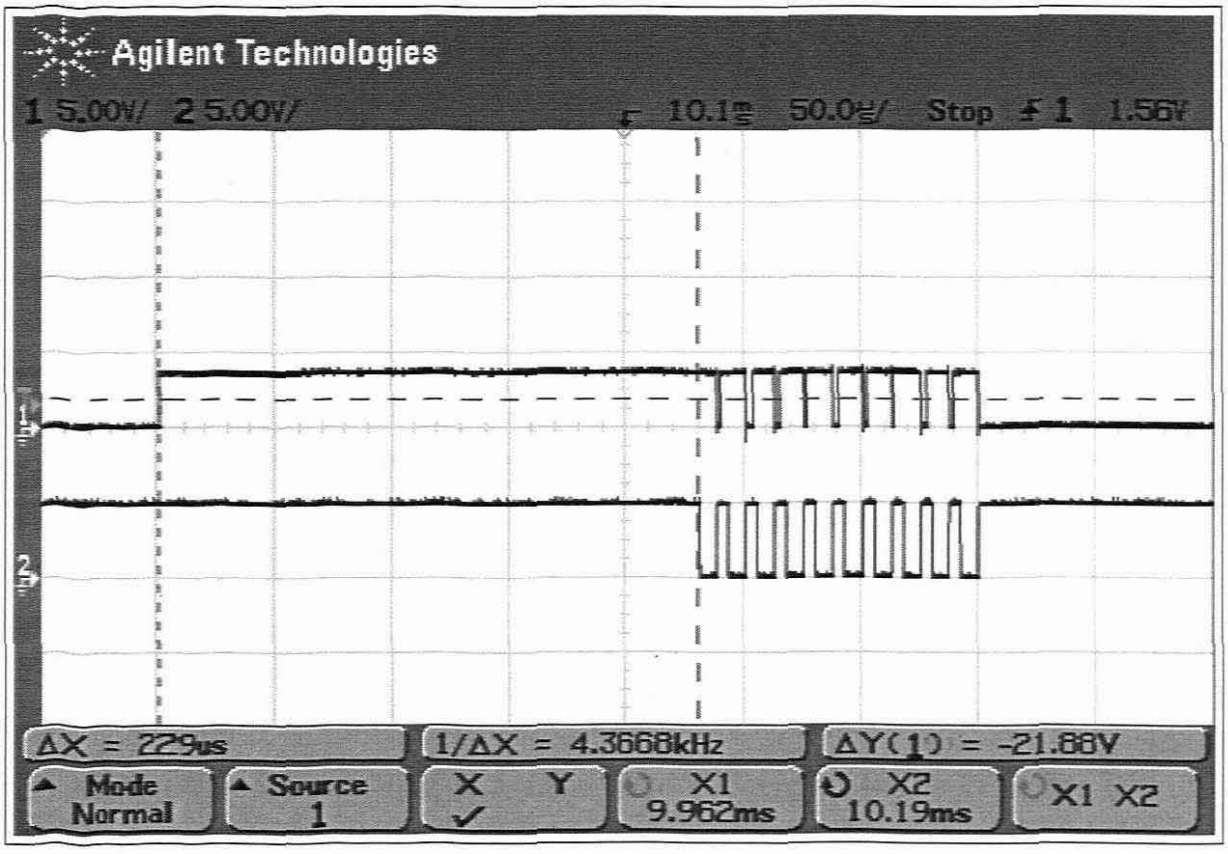


Figure 9.7: The oscilloscope capture shows the correct handshaking between $\overline{\text{RTW}}$ (channel one) and $\overline{\text{PCS}}$ (channel two). The capture is taken at the controller board and $\overline{\text{RTW}}$ is inverted.

(channel one) and $\overline{\text{PCS}}$ (channel two). From the capture is can be seen that it takes 229μs from the time the codec signals that the first byte is ready for transfer — generating the interrupt on the micro-controller — until the controller can pull the $\overline{\text{PCS}}$ line low for the first time. On a small set of samples the average time between the codec signalling and the controller triggering $\overline{\text{PCS}}$ for the first byte was 235,67μs.

Not much time was spent checking the parallel interface on the decoder board as the data it was receiving was already corrupt. That said, the capture in figure 9.8 shows the correct operation of $\overline{\text{EOD}}$ (channel one) with relation to PR (channel two). Note that the codec is considerably faster than the micro-controller and that other tests show that $\overline{\text{EOD}}$ goes low after $\overline{\text{RTR}}$ has gone high after reading the last byte in the frame. The timing diagram can be seen in figure 7.5.

The analogue component of the codec boards was operational. The MAS3159F includes a mixer that allows the analogue output to be a combination of the analogue inputs and the digital output (Micronas 2001). The mixer value was set to 50% meaning that the audio from the microphone would be passed to both the encoder and the analogue output. Listening on the headphones confirmed that the analogue component worked.

The author had acquired Microsoft Visual C++ 6 and it was decided to use the

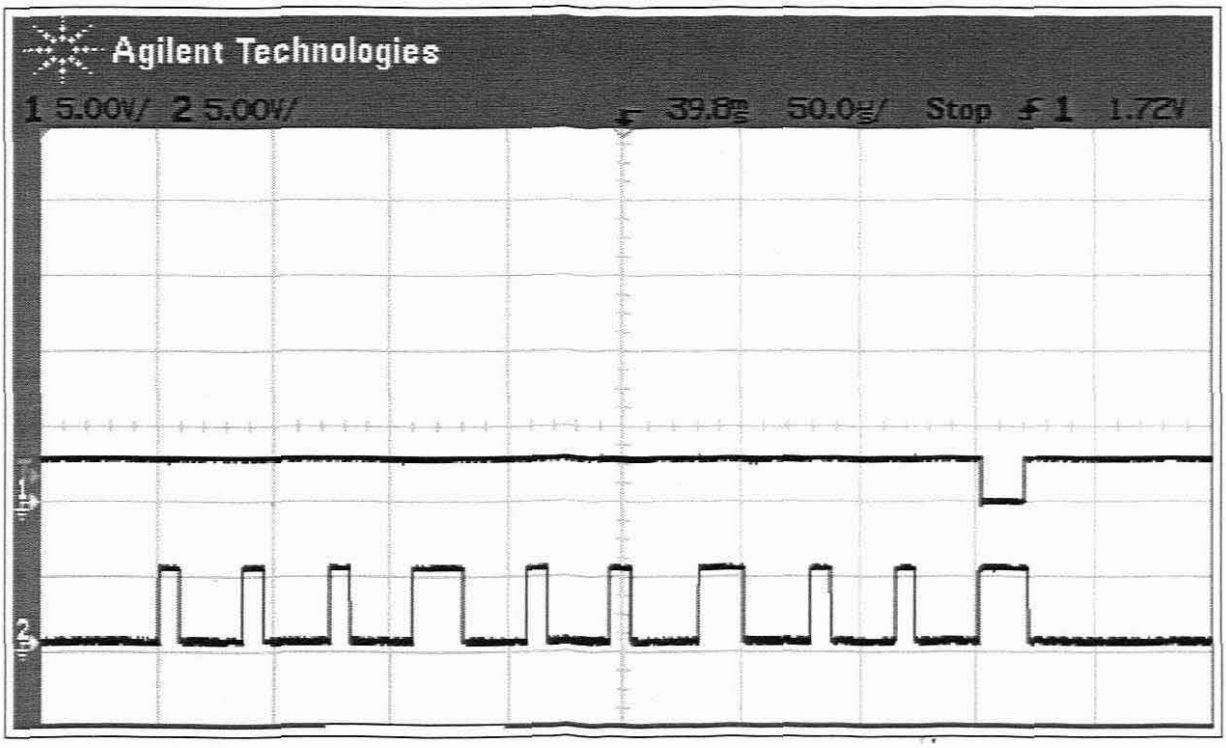


Figure 9.8: A frame being written to the codec represented by $\overline{EOD}$ and PR.

VoiceAge G.729 codec to emulate the MAS3159F. An encoder was developed but the minimum delay between frames was approximately 20ms, twice the required time. There were also some handshaking issues. Figure 9.9 shows the two problems. The handshaking is purely a software problem and as such could be resolved but the timing problem seemed more severe. It was decided to scrap the emulator plans and stick to using the codec boards as the faults did not invalidate the concept.

9.6 Discussion

Despite the problems evident, it can be seen that it is possible to develop a VoIP telephone with limited capabilities using a micro-controller. In this case a 16-bit, 40MHz 186 micro-controller with an operating system capable of running a maximum of 32 tasks on the micro-controller in the module. Each compiled program is further limited to 64KB for both code and data memory. The SIP implementation is capable of signalling the start and end of a call. SIP's needs are not particularly time critical and if memory permits it could be expanded towards a more complete implementation. SDP is adequate for the requirements of the system. It provides what little negotiation is required in a two user environment, with mostly identical units. RTP is capable of transferring the encoded audio data in real-time. RTCP is capable of providing basic quality of service parameters.

The signalling and the combined media transport and quality of service sections have been tested separately. Linking the two programs together would not have an impact on

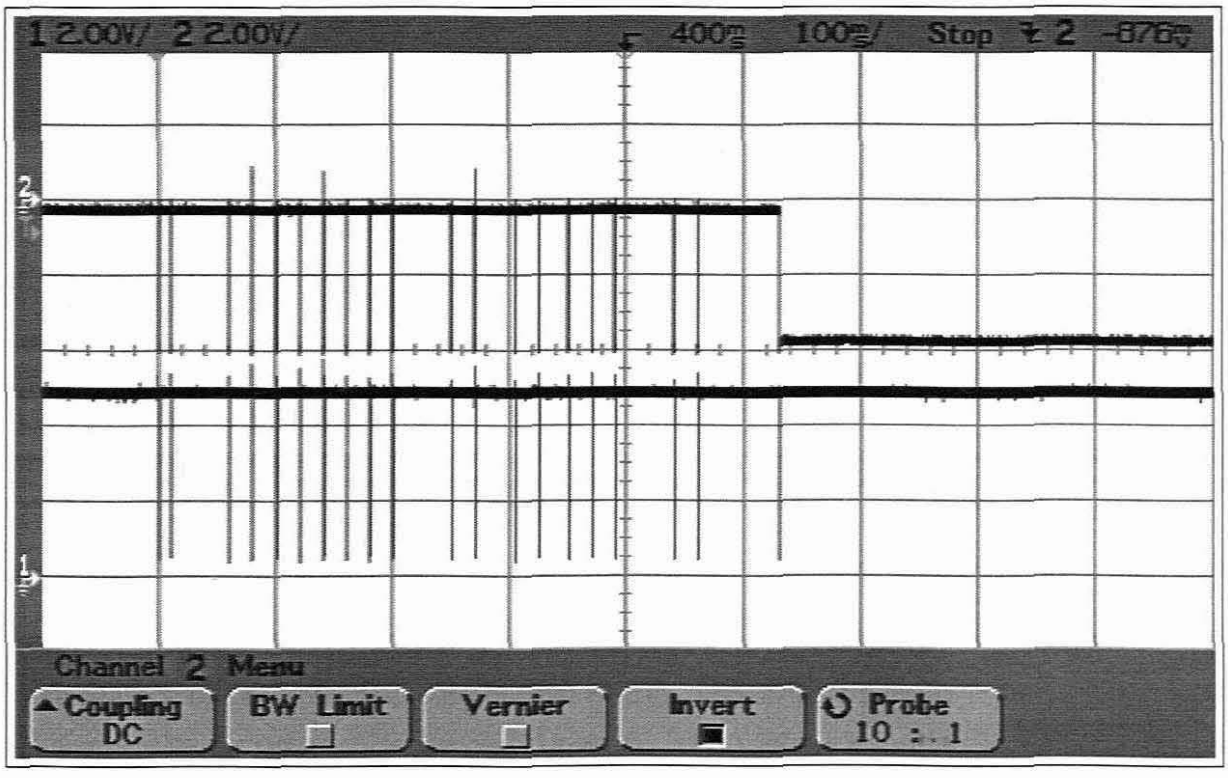


Figure 9.9: Oscilloscope capture of RTW (channel 2) and PCS (channel 1) of the PC based MAS3159F emulator.

the results shown. Data records, storage space offered in flash memory, could be used to pass data between programs however flash memory is not well suited to an environment with frequent data writes. Using mailboxes to pass data between tasks should work and a test run suggests that it does though only between tasks and not from a user command to a task. Both the task and user command of the one program were able to resume a task in the other compiled program. Minimal time was spend attempting to use the SIP and RTP programs together as it would not have serve much purpose in the context of this thesis other than to produce an interesting network capture. A few brief test were run but did not yield favourable results.

Chapter 10

Conclusions and recommendations

This project has shown that it is possible to implement the components for a basic VoIP application on a micro-controller based system. This system is limited by factors that are not be present or as noticeable in modern computer systems. For instance the controller module only contains 512KB of flash memory and 512KB of RAM. This is further divided into 64KB segments. Each compiled program is limited to a single segment for all its memory requirements. Data can be made accessible from anywhere in memory by using data records stored in flash memory. Each data record is only 32 bytes long and flash has a limited number of erase-write cycles. Mailboxes should be able to work across compiled program boundaries. This should be useful for future expansion.

The necessary hardware — two controller boards and the codec boards — were developed. The controller boards have been built to allow for reusability. The codec boards have a certain amount of reusability designed into them. The codec boards proved problematic and the parallel interface continued to behave erratically throughout testing.

A subset of the session initiation protocol was implemented which included the session description protocol for use in the offer / answer model required by SIP. The signalling section can be used to initialise and terminate a call.

A limited implementation of the real-time transport protocol and its a accompanying RTP control protocol were developed. The implementation is limited to a two user application and only G.729 is supported as the payload type.

G.729 annex A codecs were implemented in the form of MAS3159F ICs. Although the codec boards proved problematic, they were able to provide a payload to illustrate the real-time transfer of data although only half duplex communication could be tested. As the I^2C interface and the components not related to the parallel interface seemed to work fine, the author was able to illustrate the interface to a codec and the analogue interface

Chapter 10

Conclusions and recommendations

This project has shown that it is possible to implement the components for a basic VoIP application on a micro-controller based system. This system is limited by factors that are not be present or as noticeable in modern computer systems. For instance the controller module only contains 512KB of flash memory and 512KB of RAM. This is further divided into 64KB segments. Each compiled program is limited to a single segment for all its memory requirements. Data can be made accessible from anywhere in memory by using data records stored in flash memory. Each data record is only 32 bytes long and flash has a limited number of erase-write cycles. Mailboxes should be able to work across compiled program boundaries. This should be useful for future expansion.

The necessary hardware — two controller boards and the codec boards — were developed. The controller boards have been built to allow for reusability. The codec boards have a certain amount of reusability designed into them. The codec boards proved problematic and the parallel interface continued to behave erratically throughout testing.

A subset of the session initiation protocol was implemented which included the session description protocol for use in the offer / answer model required by SIP. The signalling section can be used to initialise and terminate a call.

A limited implementation of the real-time transport protocol and its a accompanying RTP control protocol were developed. The implementation is limited to a two user application and only G.729 is supported as the payload type.

G.729 annex A codecs were implemented in the form of MAS3159F ICs. Although the codec boards proved problematic, they were able to provide a payload to illustrate the real-time transfer of data although only half duplex communication could be tested. As the I^2C interface and the components not related to the parallel interface seemed to work fine, the author was able to illustrate the interface to a codec and the analogue interface

for the user.

Ethereal was the primary component used for testing and monitoring. Captures of network data were taken and could be viewed in Ethereal using the relevant protocol dissector and analysed manually for correctness.

The bandwidth required is sufficiently low so as to allow usage over dial-up Internet connections.

There is room for future work on the project. Protocols can be expanded as resources permit. The terminal emulator can be replaced by a keypad, display and a small controller. It is suggested that developers rather make use of other more recent developments with respect to processors and applicable software. System-on-chip options are available as are micro-controllers for which a growing of open source projects can be ported.

References

3GPP (2004), 3gpp. [Last visited January 2004].
*http://www.3gpp.org

AT&T (2003), AT&T consumer. [Last visited December 2003].
*http://www.consumer.att.com

AWC (2000), SLIM-LINK® SERVER PROGRAMMERS GUIDE for AWC86 and AWC86A MicroRTOS Version 1.50, Programmers guide revision a, Advanced Web Communication, Xecom.

AWC (2001), Awc86 slimlink web-controller & micrortos faq. AWC. Division of Xecom Inc. [Last visited October 2001].
*http://home.xecom.com/faq.asp

Axis (n.d.), ETRAX Multi-Chip-Module @ developer.axis.com. [Last visited March 2004].
*http://developer.axis.com/products/mcm/

Braden, R., Zhang, L., Berson, S., Herzog, S. & Jamin, S. (1997), Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification, Request for Comments 2205, Internet Engineering Task Force.

Buchanan, B. (1999), *Handbook of Data Communication and Networks*, Kluwer Academic Publishers, Boston.

Calmicro (2001), 6 Channel ESD Protection Array, PACDN006, California Micro Devices.

Casson, H. N. (1997), *The History Of The Telephone*, Project Gutenberg Etext. 1997 is the year the etext was 'published'. The bibliographic details of the original work are not known.

Cerf, V. G. & Kahn, R. E. (1984), A protocol for packet network intercommunication, *in* S. S. Lam, ed., 'Tutorial: Principles of communication and networking protocols', IEEE Computer Society Press, Silver Spring, MD.

Cohen, D. (1976), Specifications for the Network Voice Protocol (NVP), Request for Comments 741, Internet Engineering Task Force.

Comer, D. E. (1995), *The Internet Book: Everything you need to know about computer networking and how the Internet works*, Prentice-Hall, Englewood Cliffs, New Jersey.

Conexant (2003), Conexant Systems, Inc. - Driving the Digital home. Conexant Systems, Inc.[Last visited February 2004].
*http://www.conexant.com

Crocker, D. & Overell, P. (1997), Augmented bnf for syntax specifications: Abnf, Request for Comments 2234, Internet Engineering Task Force.

Dalsemi (2000), Trickle Charge Timekeeping Chip, DS1302, Dallas Semiconductor.

DARPA (n.d.), DARPA home. Defence Advanced Research Projects Agency homepage [Last visited February 2004].
*http://www.darpa.mil

Databeam (1998), A Primer on the H.323 Series Standard.
*http://www.databeam.com

Day, M., Aggarwal, S., Mohr, G. & Vincent, J. (2000), Instant Messaging / Presence Protocol Requirements, Request for Comments 2779, Internet Engineering Task Force.

Dyer, F. L. (1997), *Edison, his life and inventions*, The World Wide School, Seattle, Wash., chapter 36 (Appendix part 6). [Last visited March 2004].
*http://www.worldwideschool.org/library/books/hst/biography/Edison/chap36.html

Edison, T. A. (1892), Speaking-telegraph, Patent number 474230, United States Patent Office.

Ethereal (2004), The Ethereal Network Analyzer. Ethereal homepage [Last visited February 2004].
*http://www.ethereal.com

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999), Hypertext Transfer Protocol – HTTP/1.1, Request for Comments 2616, Internet Engineering Task Force.

Frenzel, L. E. (1989), *Communication electronics*, McGraw-Hill, New York.

Handley, M. & Jacobson, V. (1998), SDP: Session Description Protocol, Request for Comments 2327, Internet Engineering Task Force.

Handley, M., Schulzrinne, H., Schooler, E. & Rosenberg, J. (1999), SIP: Session Initiation Protocol, Request for Comments 2543, Internet Engineering Task Force.

Hanewinkel, H. (n.d.), http://www.hanewin.de/homee.htm. [Last visited February 2004].
*http://www.hanewin.de/homee.htm

Hellstrom, G. (2000), RTP Payload for Text Conversation, Request for Comments 2793, Internet Engineering Task Force.

IEC (n.d.), Voice over Internet Protocol. International Engineering Consortium Web ProForum Tutorials.
*http://www.iec.org

IETF (2004), IETF. [Last visited February 2004].
*http://www.ietf.org

info@splint.org (2004), Splint Home Page. [Last visited March 2004].
*http://www.splint.org

ISC (2003), Internet Software Consortium. ISC homepage [Last visited January 2004].
*http://www.isc.org

ITU (2004), ITU. [Last visited March 2004].
*http://www.itu.int

ITU-T (1993), Pulse code modulation PCM of voice frequencies, ITU-T Recommendation G.711, Telecommunication Standardization Sector of ITU.

ITU-T (1996*a*), Coding of speech at 8kbit/s using Conjugate Structure Algebraic-Code-Excited-Linear-Prediction (CS-ACELP), ITU-T Recommendation G.729, Telecommunication Standardization Sector of ITU.

ITU-T (1996*b*), Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s, ITU-T Recommendation G.723.1, Telecommunication Standardization Sector of ITU.

ITU-T (1996*c*), Reduced complexity 8 kbit/s cs-acelp speech codec, ITU-T Recommendation G.729 Annex A, Telecommunication Standardization Sector of ITU.

Jones, P. (2000), Packet-based multimedia communications systems, ITU-T Recommendation H.323 (Draft v4), Telecommunication Standardization Sector of ITU.

Lineo (2001), Lineo. No longer exists. Now redirected to Metrowerks' Linux solutions page.
*http://www.lineo.com

Loewen, M. (1998), Using PICmicro® MCUs to Connect to Internet via PPP, AN724, Microchip Technology Inc.

Maxim (1996), ±15kV ESD-Protected, +5v RS-232 Transceivers, MAX202E–MAX213E, MAX232E/MAX241E, Maxim Integrated Products.

Maxim (2000), +5v-Powered, Multichannel RS-232 Drivers/Receivers, MAX220–MAX249, Maxim Integrated Products.

MCI (2003), Mci. MCI Consumer Home [Last visited December 2003].
*http://consumer.mci.com/mcicom_home

Micronas (2001), MAS 35x9F MPEG Layer 2/3, AAC Audio Decoder, G.729 Annex A Codec, Preliminary data sheet, Micronas.

Mills, D. L. (1992), Network Time Protocol (Version 3) Specification, Implementation and Analysis, Request for Comments 1305, Internet Engineering Task Force.

Mockapetris, P. (1987), Domain Names - Implementation and Specification, Request for Comments 1035, Internet Engineering Task Force.

Motorola (1996), High-Speed CMOS Data, Technical report, Motorola.

Networks, A. (2001), www.uClinux.com. [Last visited February 2004].
*http://www.uclinux.com

Packetizer (2003*a*), Current Status of H.323 Series Documents. [Last visited February 2004].
*http://www.packetizer.com/iptel/h323/doc_status.html

Packetizer (2003*b*), Packetizer. [Last visited March 2004].
*http://www.packetizer.com/

Philips (2000), The i^2c-bus specification, Specification version 2.1, Philips Semiconductors.

Postel, J. B., Sunshine, C. A. & Cohen, D. (1984), The ARPA internet protocol, *in* S. S. Lam, ed., 'Tutorial: Principles of communication and networking protocols', IEEE Computer Society Press, Silver Spring, MD.

Qualcomm (1997), QUALCOMM PureVoice technology. PureVoice_Datasheet.pdf.
*http://www.qualcomm.com

Rigney, C. (2000), Radius accounting, Request for Comments 2866, Internet Engineering Task Force.

Roberts, L. G. (1984), The evolution of packet switching, *in* S. S. Lam, ed., 'Tutorial: Principles of communication and networking protocols', IEEE Computer Society Press, Silver Spring, MD. Roberts, L., "The Evolution of Packet Switching," Proc. IEEE, Vol. 66, November 1978.

Romkey, J. (1988), A nonstandard for transmission of IP datagrams over serial lines: SLIP, Request for Comments 1055, Internet Engineering Task Force.

Rosenberg, J. (2000), 'Parched For Services? here, Try A SIP', *Communications Solutions* . [Last visited January 2001].
*http://www.tmcnet.com/articles/comsol/0500/0500ays_rosenberg.htm

Rosenberg, J. & Schulzrinne, H. (2002*a*), An Offer/Answer Model with the Session Description Protocol (SDP), Request for Comments 3264, Internet Engineering Task Force.

Rosenberg, J. & Schulzrinne, H. (2002*b*), Session Initiation Protocol (SIP): Locating SIP Servers, Request for Comments 3263, Internet Engineering Task Force.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. & Schooler, E. (2002), SIP: Session Initiation Protocol, Request for Comments 3261, Internet Engineering Task Force.

Schlatter, C. (2003), Basic Architecture of H.323. Handouts [Last visited February 2004].
*http://www.switch.ch/vconf/ws2003/h323_basics_handout.pdf

Schulzrinne, H. (1996), RTP Profile for Audio and Video Conferences with Minimal Control, Request for Comments 1890, Internet Engineering Task Force.

Schulzrinne, H. & Casner, S. (2003), RTP Profile for Audio and Video Conferences with Minimal Control, Request for Comments 3551, Internet Engineering Task Force.

Schulzrinne, H., Casner, S., Frederick, R. & Jacobson, V. (1996), RTP: A Transport Protocol for Real-Time Applications, Request for Comments 1889, Internet Engineering Task Force.

Schulzrinne, H., Casner, S., Frederick, R. & Jacobson, V. (2003), RTP: A Transport Protocol for Real-Time Applications, Request for Comments 3550, Internet Engineering Task Force.

Schulzrinne, H. & Rosenberg, J. (1998), 'Internet Telephony: Architecture and Protocols an IETF Perspective'. [Last visited February 2004].
*http://www.jdrosen.net

Schulzrinne, H. & Rosenberg, J. (n.d.), 'A Comparison of SIP and H.323 for Internet Telephony'.
*http://www.jdrosen.net

Sedgewick, R. (1990), *Algorithms in C*, Addison Wesley, Reading, Mass., pp. 511–514.

Simpson, W. (1994), The Point-to-Point Protocol (PPP), Request for Comments 1661, Internet Engineering Task Force.

Skype (2004), www.skype.com. [Last visited March 2004].
*http://www.skype.com

@stake (2004), Network Utility Research Tools. [Last visited February 2004].
*http://www.atstake.com/research/tools/network_utilities/

Tanenbaum, A. S. (1996), *Computer Networks*, third edn, Prentice-Hall, London.

Telkom (2003*a*). Telkom residential price list. URL is correct as spelled below. [Last visited December 2003].
*http://www.telkom.co.za/./custtomer_to_automatic_exh.jsp

Telkom (2003*b*), International Tariffs. Telkom international tariffs [Last visited December 2003].
*http://www.telkom.co.za/./international.jsp

The SIP Center (2003). [Last visited February 2004].
*http://www.sipcenter.com

VoiceAge (2000), Massive encoding. White paper.
*http://www.voiceage.com

VoiceAge (n.d.), VoiceAge. No author or year on website. [Last visited February 2004].
*http://www.voiceage.com

Xecom (2003), Xecom, Inc - Leaders in telecommunications and communications components. [Last visited February 2004].
*http://www.xecom.com

XE.com (2004), XE.com. [Last visited January 2004].
*http://www.xe.com

Appendices

Appendix A

AWC86

A.1 AWC86DK schematic

Schematic provided by Xecom (Xecom 2003).

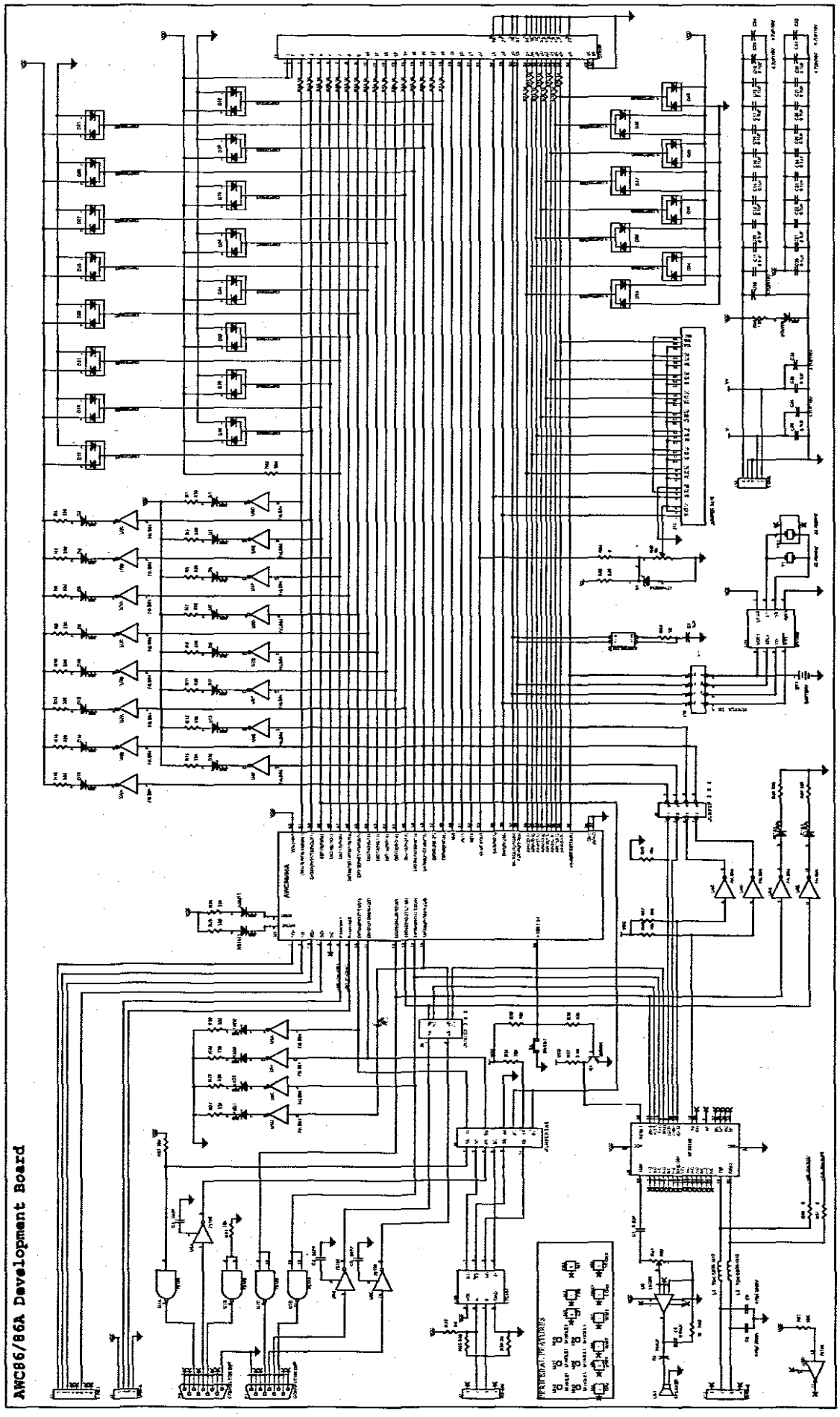


Figure A.1: AWC86 development board schematic.

A.2 Controller board schematic

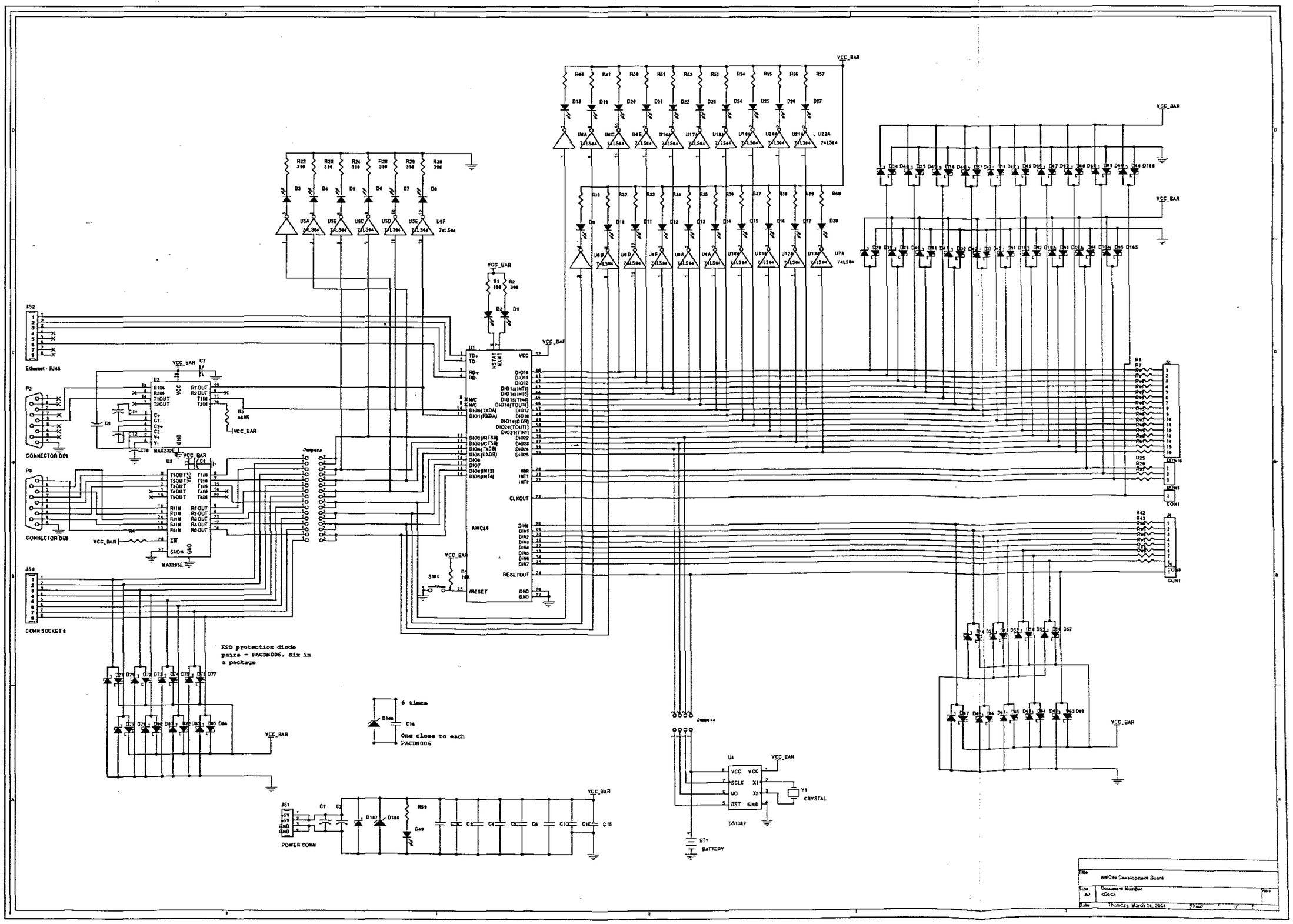


Figure A.2: Controller board schematic.

Appendix B

Signalling

Appendices B.1 – B.4 are from RFC 3261 (Rosenberg et al. 2002). A description of each finite state machine can be found in the same document.

B.1 INVITE client transaction

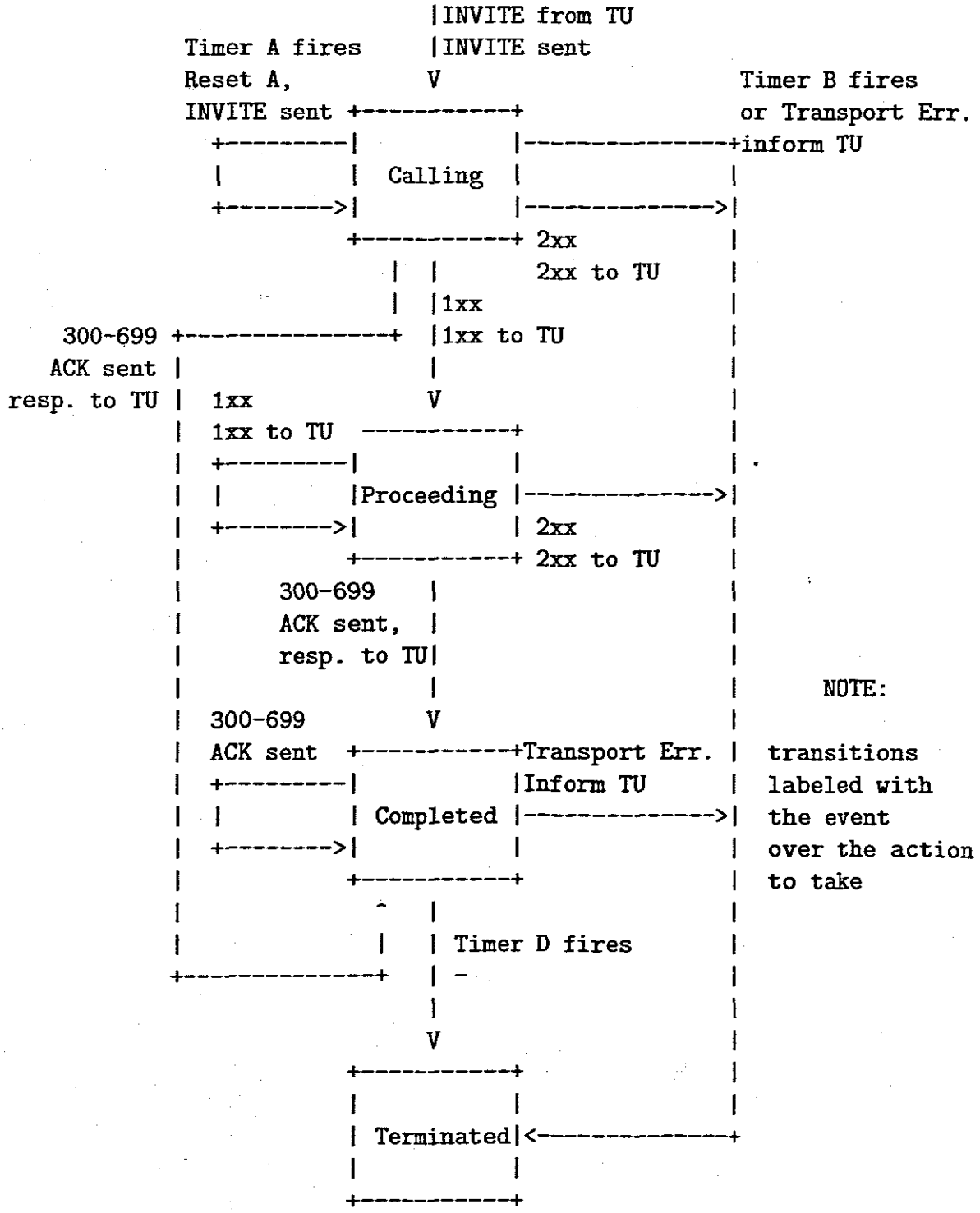


Figure 5: INVITE client transaction

B.2 Non-INVITE client transaction

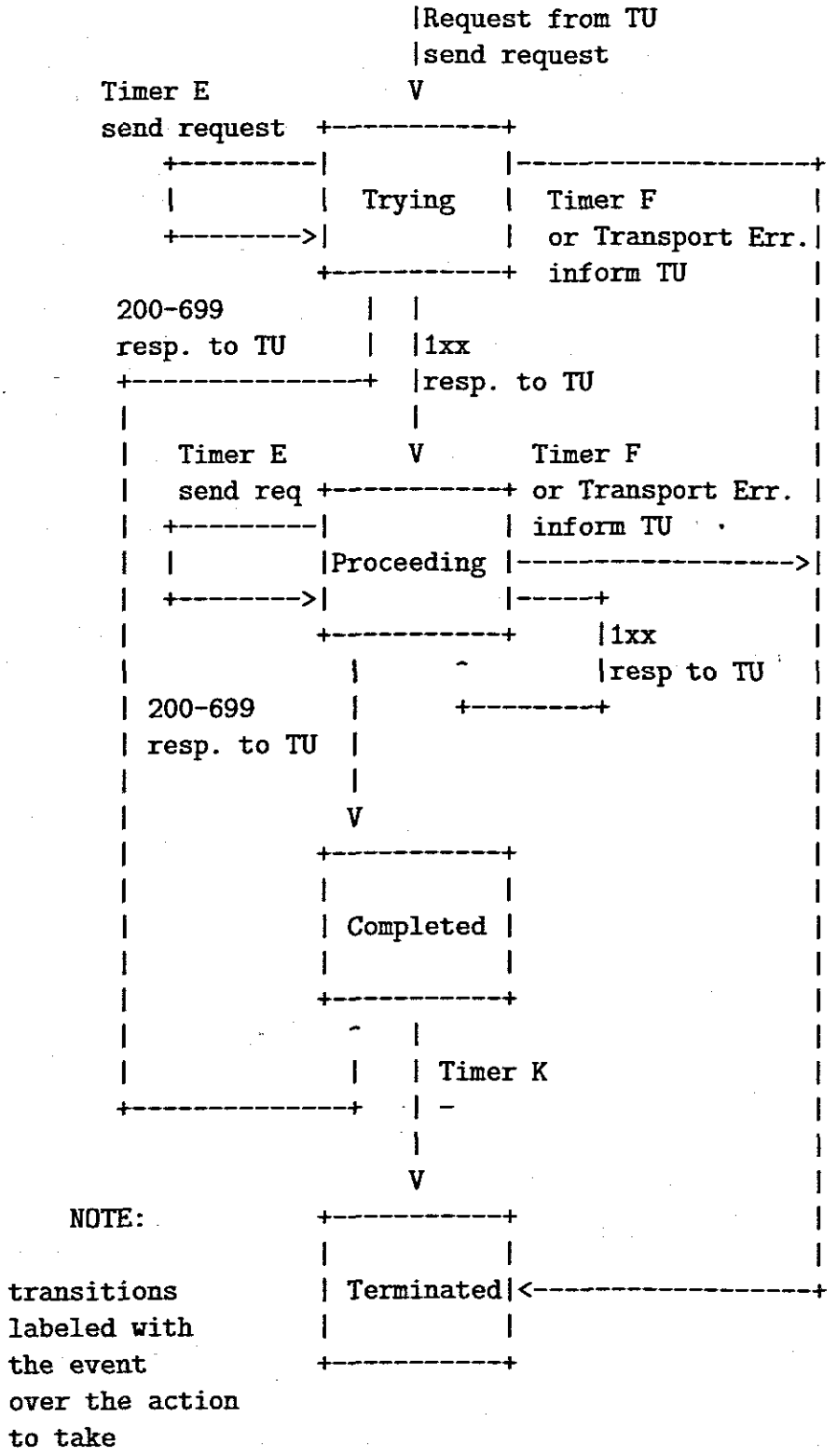


Figure 6: non-INVITE client transaction

B.3 INVITE server transaction

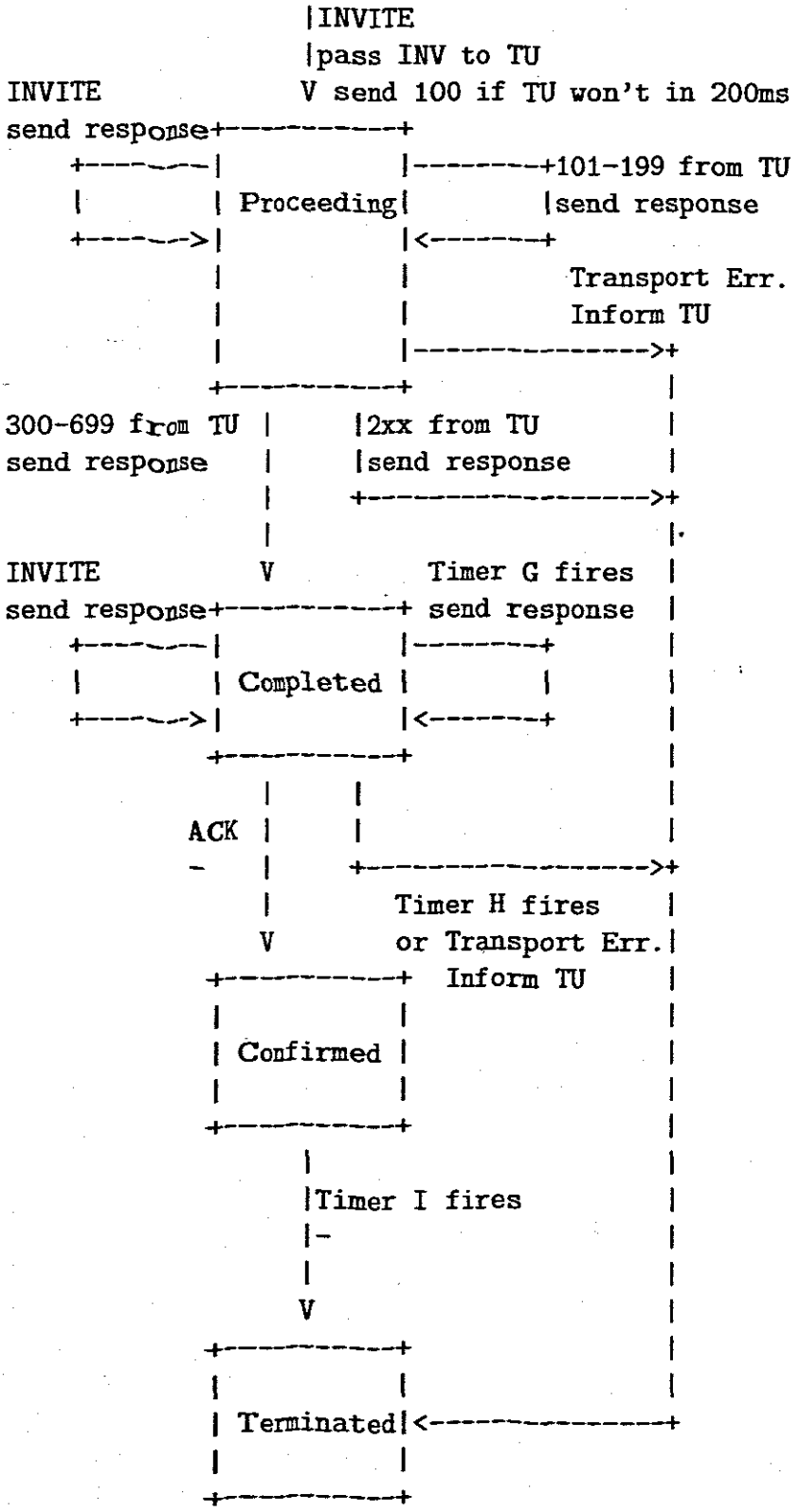


Figure 7: INVITE server transaction

B.4 Non-INVITE server transaction

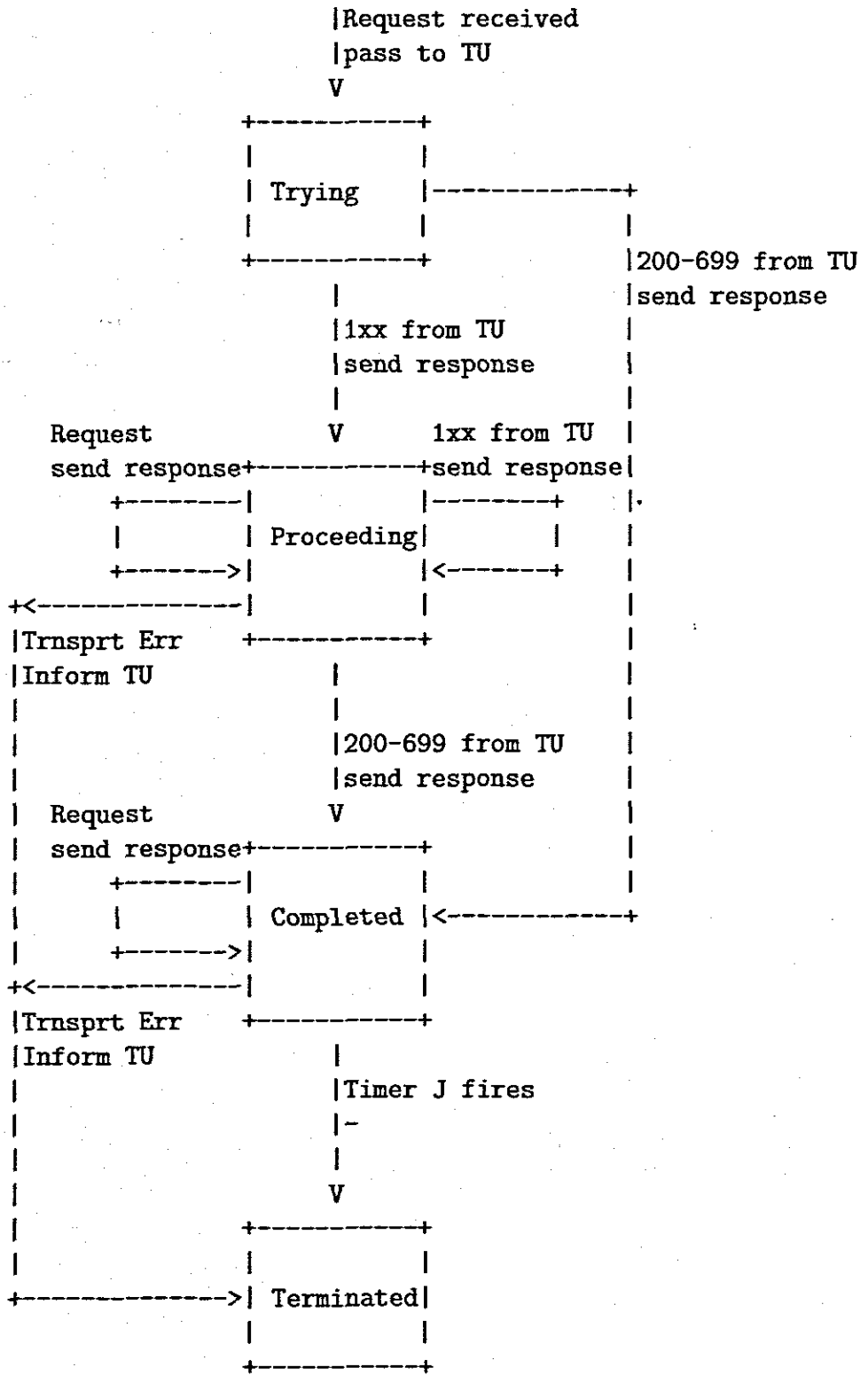


Figure 8: non-INVITE server transaction

B.5 SIP capture 1

```
Frame 11 (505 bytes on wire, 505 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6103 (6103), Dst Port: 5060 (5060),
                               Seq: 4395, Ack: 198096, Len: 451
Session Initiation Protocol
    Request line: INVITE sip:com2@ctech.ac.za SIP/2.0
        Method: INVITE
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK63
        t:<sip:com2@ctech.ac.za>
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        Max-Forwards:70
        i:2663031567com1@ctech.ac.za
        CSeq:11738 INVITE
        m:<sip:com1@ctech.ac.za>
        Content-Disposition:session
        c:application/sdp
        l:152
    Message body
        Session Description Protocol
            Session Description Protocol Version (v): 0
            Owner/Creator, Session Id (o): com1 335094308 335094308 IN
            IP4 155.238.33.245
                Owner Username: com1
                Session ID: 335094308
                Session Version: 335094308
                Owner Network Type: IN
                Owner Address Type: IP4
                Owner Address: 155.238.33.245
            Session Name (s): -
            Connection Information (c): IN IP4 155.238.33.245
                Connection Network Type: IN
                Connection Address Type: IP4
                Connection Address: 155.238.33.245
```

Time De i i i e ime

e i Time

e i Time

e i i e e e

e i De i i me e m i T

e i T e i

e i

e i T

e i m

e i i e m

e i i e ie me m

e i i e e

e i i i e

T i m e

T

i m e

i m e

i m e

e T

m i m e

e Di i i e i

i i

m

e e

m i T

m

D

e ee ee

e T i

e e m e

e

```
0060  2f 32 2e 30 2f 54 43 50 20 31 35 35 2e 32 33 38   /2.0/TCP 155.238
0070  2e 33 33 2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a   .33.245;branch=z
0080  39 68 47 34 62 4b 36 33 0d 0a 74 3a 3c 73 69 70   9hG4bK63..t:<sip
0090  3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a   :com2@ctech.ac.z
00a0  61 3e 0d 0a 66 3a 3c 73 69 70 3a 63 6f 6d 31 40   a>..f:<sip:com1@
00b0  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
00c0  3d 39 34 38 34 31 31 30 37 35 0d 0a 4d 61 78 2d   =948411075..Max-
00d0  46 6f 72 77 61 72 64 73 3a 37 30 0d 0a 69 3a 32   Forwards:70..i:2
00e0  36 36 33 30 33 31 35 36 37 63 6f 6d 31 40 63 74   663031567com1@ct
00f0  65 63 68 2e 61 63 2e 7a 61 0d 0a 43 53 65 71 3a   ech.ac.za..CSeq:
0100  31 31 37 33 38 20 49 4e 56 49 54 45 0d 0a 6d 3a   11738 INVITE..m:
0110  3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e   <sip:com1@ctech.
0120  61 63 2e 7a 61 3e 0d 0a 43 6f 6e 74 65 6e 74 2d   ac.za>..Content-
0130  44 69 73 70 6f 73 69 74 69 6f 6e 3a 73 65 73 73   Disposition:sess
0140  69 6f 6e 0d 0a 63 3a 61 70 70 6c 69 63 61 74 69   ion..c:applicati
0150  6f 6e 2f 73 64 70 0d 0a 6c 3a 31 35 32 0d 0a 0d   on/sdp..l:152...
0160  0a 76 3d 30 0d 0a 6f 3d 63 6f 6d 31 20 33 33 35   .v=0..o=com1 335
0170  30 39 34 33 30 38 20 33 33 35 30 39 34 33 30 38   094308 335094308
0180  20 49 4e 20 49 50 34 20 31 35 35 2e 32 33 38 2e    IN IP4 155.238.
0190  33 33 2e 32 34 35 0d 0a 73 3d 2d 0d 0a 63 3d 49   33.245..s=-..c=I
01a0  4e 20 49 50 34 20 31 35 35 2e 32 33 38 2e 33 33   N IP4 155.238.33
01b0  2e 32 34 35 0d 0a 74 3d 30 20 30 0d 0a 61 3d 73   .245..t=0 0..a=s
01c0  65 6e 64 72 65 63 76 0d 0a 6d 3d 61 75 64 69 6f   endrecv..m=audio
01d0  20 35 30 30 34 20 52 54 50 2f 41 56 50 20 31 38    5004 RTP/AVP 18
01e0  0d 0a 61 3d 72 74 70 6d 61 70 3a 31 38 20 47 37   ..a=rtpmap:18 G7
01f0  32 39 2f 38 30 30 30 0d 0a                       29/8000..
```

```
Frame 13 (274 bytes on wire, 274 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                    Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6103 (6103),
                                Seq: 198096, Ack: 4846, Len: 220
Session Initiation Protocol
    Status line: SIP/2.0 180
        Status-Code: 180
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK63
        t:<sip:com2@ctech.ac.za>;tag=2905616230
            SIP to address: <sip:com2@ctech.ac.za>
```

```
            SIP tag: 2905616230
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        i:2663031567com1@ctech.ac.za
        CSeq:11738 INVITE
        m:<sip:com2@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 180  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK63\r\n
    t:<sip:com2@ctech.ac.za>;tag=2905616230\r\n
    f:<sip:com1@ctech.ac.za>;tag=948411075\r\n
    i:2663031567com1@ctech.ac.za\r\n
    CSeq:11738 INVITE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  01 04 18 00 00 00 80 06 a6 31 9b ee 21 f1 9b ee   .........1..!...
0020  21 f5 13 c4 17 d7 00 03 05 d0 00 00 12 ee 50 18   !.............P.
0030  05 dc 1a 03 00 00 53 49 50 2f 32 2e 30 20 31 38   ......SIP/2.0 18
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  36 33 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32 40   63..t:<sip:com2@
0080  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
0090  3d 32 39 30 35 36 31 36 32 33 30 0d 0a 66 3a 3c   =2905616230..f:<
00a0  73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e 61   sip:com1@ctech.a
00b0  63 2e 7a 61 3e 3b 74 61 67 3d 39 34 38 34 31 31   c.za>;tag=948411
00c0  30 37 35 0d 0a 69 3a 32 36 36 33 30 33 31 35 36   075..i:266303156
00d0  37 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a   7com1@ctech.ac.z
00e0  61 0d 0a 43 53 65 71 3a 31 31 37 33 38 20 49 4e   a..CSeq:11738 IN
00f0  56 49 54 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f 6d   VITE..m:<sip:com
0100  32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d 0a   2@ctech.ac.za>..
0110  0d 0a                                             ..

Frame 23 (481 bytes on wire, 481 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                   Dst Addr: 155.238.33.245 (155.238.33.245)
```

```
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6103 (6103),
                                Seq: 198316, Ack: 4846, Len: 427
Session Initiation Protocol
    Status line: SIP/2.0 200
        Status-Code: 200
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK63
        t:<sip:com2@ctech.ac.za>;tag=2905616230
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2905616230
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        i:2663031567com1@ctech.ac.za
        CSeq:11738 INVITE
        m:<sip:com2@ctech.ac.za>
        Content-Disposition:session
        c:application/sdp
        l:152
    Message body
        Session Description Protocol
            Session Description Protocol Version (v): 0
            Owner/Creator, Session Id (o): com2 211685113 211685113 IN
            IP4 155.238.33.241
                Owner Username: com2
                Session ID: 211685113
                Session Version: 211685113
                Owner Network Type: IN
                Owner Address Type: IP4
                Owner Address: 155.238.33.241
            Session Name (s): -
            Connection Information (c): IN IP4 155.238.33.241
                Connection Network Type: IN
                Connection Address Type: IP4
                Connection Address: 155.238.33.241
            Time Description, active time (t): 0 0
                Session Start Time: 0
                Session Start Time: 0
            Session Attribute (a): sendrecv
            Media Description, name and address (m): audio 5004 RTP/AVP 18
```

```
                Media Type: audio
                Media Port: 5004
                Media Proto: RTP/AVP
                Media Format: 18
            Media Attribute (a): rtpmap:18 G729/8000
                Media Attribute Fieldname: rtpmap
                Media Attribute Value: 18 G729/8000
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 200  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK63\r\n
    t:<sip:com2@ctech.ac.za>;tag=2905616230\r\n
    f:<sip:com1@ctech.ac.za>;tag=948411075\r\n
    i:2663031567com1@ctech.ac.za\r\n
    CSeq:11738 INVITE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    Content-Disposition:session\r\n
    c:application/sdp\r\n
    l:152\r\n
    \r\n
    v=0\r\n
    o=com2 211685113 211685113 IN IP4 155.238.33.241\r\n
    s=-\r\n
    c=IN IP4 155.238.33.241\r\n
    t=0 0\r\n
    a=sendrecv\r\n
    m=audio 5004 RTP/AVP 18\r\n
    a=rtpmap:18 G729/8000\r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  01 d3 1b 00 00 00 80 06 a2 62 9b ee 21 f1 9b ee   .........b..!...
0020  21 f5 13 c4 17 d7 00 03 06 ac 00 00 12 ee 50 18   !.............P.
0030  05 dc 8e 62 00 00 53 49 50 2f 32 2e 30 20 32 30   ...b..SIP/2.0 20
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  36 33 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32 40   63..t:<sip:com2@
0080  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
0090  3d 32 39 30 35 36 31 36 32 33 30 0d 0a 66 3a 3c   =2905616230..f:<
00a0  73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e 61   sip:com1@ctech.a
00b0  63 2e 7a 61 3e 3b 74 61 67 3d 39 34 38 34 31 31   c.za>;tag=948411
```

```
00c0  30 37 35 0d 0a 69 3a 32 36 36 33 30 33 31 35 36   075..i:266303156
00d0  37 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a   7com1@ctech.ac.z
00e0  61 0d 0a 43 53 65 71 3a 31 31 37 33 38 20 49 4e   a..CSeq:11738 IN
00f0  56 49 54 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f 6d   VITE..m:<sip:com
0100  32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d 0a   2@ctech.ac.za>..
0110  43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f 73 69 74   Content-Disposit
0120  69 6f 6e 3a 73 65 73 73 69 6f 6e 0d 0a 63 3a 61   ion:session..c:a
0130  70 70 6c 69 63 61 74 69 6f 6e 2f 73 64 70 0d 0a   pplication/sdp..
0140  6c 3a 31 35 32 0d 0a 0d 0a 76 3d 30 0d 0a 6f 3d   l:152....v=0..o=
0150  63 6f 6d 32 20 32 31 31 36 38 35 31 31 33 20 32   com2 211685113 2
0160  31 31 36 38 35 31 31 33 20 49 4e 20 49 50 34 20   11685113 IN IP4
0170  31 35 35 2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a   155.238.33.241..
0180  73 3d 2d 0d 0a 63 3d 49 4e 20 49 50 34 20 31 35   s=-..c=IN IP4 15
0190  35 2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a 74 3d   5.238.33.241..t=
01a0  30 20 30 0d 0a 61 3d 73 65 6e 64 72 65 63 76 0d   0 0..a=sendrecv.
01b0  0a 6d 3d 61 75 64 69 6f 20 35 30 30 34 20 52 54   .m=audio 5004 RT
01c0  50 2f 41 56 50 20 31 38 0d 0a 61 3d 72 74 70 6d   P/AVP 18..a=rtpm
01d0  61 70 3a 31 38 20 47 37 32 39 2f 38 30 30 30 0d   ap:18 G729/8000.
01e0  0a                                                .
```

```
Frame 25 (308 bytes on wire, 308 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                  Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6103 (6103), Dst Port: 5060 (5060),
                               Seq: 4846, Ack: 198743, Len: 254
Session Initiation Protocol
    Request line: ACK sip:com2@ctech.ac.za SIP/2.0
        Method: ACK
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK155
        t:<sip:com2@ctech.ac.za>;tag=2905616230
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2905616230
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        Max-Forwards:70
        i:2663031567com1@ctech.ac.za
        CSeq:11738 ACK
```

```
      m:<sip:com1@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    ACK sip:com2@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK155\r\n
    t:<sip:com2@ctech.ac.za>;tag=2905616230\r\n
    f:<sip:com1@ctech.ac.za>;tag=948411075\r\n
    Max-Forwards:70\r\n
    i:2663031567com1@ctech.ac.za\r\n
    CSeq:11738 ACK\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  01 26 1c 00 00 00 80 06 a2 0f 9b ee 21 f5 9b ee   .&..........!...
0020  21 f1 17 d7 13 c4 00 00 12 ee 00 03 08 57 50 18   !............WP.
0030  05 dc d3 be 00 00 41 43 4b 20 73 69 70 3a 63 6f   ......ACK sip:co
0040  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 20 53   m2@ctech.ac.za S
0050  49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50 2f 32 2e   IP/2.0..v:SIP/2.
0060  30 2f 54 43 50 20 31 35 35 2e 32 33 38 2e 33 33   0/TCP 155.238.33
0070  2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a 39 68 47   .245;branch=z9hG
0080  34 62 4b 31 35 35 0d 0a 74 3a 3c 73 69 70 3a 63   4bK155..t:<sip:c
0090  6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e   om2@ctech.ac.za>
00a0  3b 74 61 67 3d 32 39 30 35 36 31 36 32 33 30 0d   ;tag=2905616230.
00b0  0a 66 3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65   .f:<sip:com1@cte
00c0  63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67 3d 39 34   ch.ac.za>;tag=94
00d0  38 34 31 31 30 37 35 0d 0a 4d 61 78 2d 46 6f 72   8411075..Max-For
00e0  77 61 72 64 73 3a 37 30 0d 0a 69 3a 32 36 36 33   wards:70..i:2663
00f0  30 33 31 35 36 37 63 6f 6d 31 40 63 74 65 63 68   031567com1@ctech
0100  2e 61 63 2e 7a 61 0d 0a 43 53 65 71 3a 31 31 37   .ac.za..CSeq:117
0110  33 38 20 41 43 4b 0d 0a 6d 3a 3c 73 69 70 3a 63   38 ACK..m:<sip:c
0120  6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e   om1@ctech.ac.za>
0130  0d 0a 0d 0a                                       ....

Frame 57 (307 bytes on wire, 307 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6103 (6103), Dst Port: 5060 (5060),
                               Seq: 5100, Ack: 198743, Len: 253
Session Initiation Protocol
```

```
    Request line: BYE sip:com2@ctech.ac.za SIP/2.0
        Method: BYE
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK53
        t:<sip:com2@ctech.ac.za>;tag=2905616230
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2905616230
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        Max-Forwards:70
        i:2663031567com1@ctech.ac.za
        CSeq:11739 BYE
        m:<sip:com1@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    BYE sip:com2@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK53\r\n
    t:<sip:com2@ctech.ac.za>;tag=2905616230\r\n
    f:<sip:com1@ctech.ac.za>;tag=948411075\r\n
    Max-Forwards:70\r\n
    i:2663031567com1@ctech.ac.za\r\n
    CSeq:11739 BYE\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  01 25 25 00 00 00 80 06 99 10 9b ee 21 f5 9b ee   .%%.........!...
0020  21 f1 17 d7 13 c4 00 00 13 ec 00 03 08 57 50 18   !............WP.
0030  05 dc e3 c0 00 00 42 59 45 20 73 69 70 3a 63 6f   ......BYE sip:co
0040  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 20 53   m2@ctech.ac.za S
0050  49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50 2f 32 2e   IP/2.0..v:SIP/2.
0060  30 2f 54 43 50 20 31 35 35 2e 32 33 38 2e 33 33   0/TCP 155.238.33
0070  2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a 39 68 47   .245;branch=z9hG
0080  34 62 4b 35 33 0d 0a 74 3a 3c 73 69 70 3a 63 6f   4bK53..t:<sip:co
0090  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b   m2@ctech.ac.za>;
00a0  74 61 67 3d 32 39 30 35 36 31 36 32 33 30 0d 0a   tag=2905616230..
00b0  66 3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63   f:<sip:com1@ctec
00c0  68 2e 61 63 2e 7a 61 3e 3b 74 61 67 3d 39 34 38   h.ac.za>;tag=948
00d0  34 31 31 30 37 35 0d 0a 4d 61 78 2d 46 6f 72 77   411075..Max-Forw
00e0  61 72 64 73 3a 37 30 0d 0a 69 3a 32 36 36 33 30   ards:70..i:26630
```

```
00f0  33 31 35 36 37 63 6f 6d 31 40 63 74 65 63 68 2e   31567com1@ctech.
0100  61 63 2e 7a 61 0d 0a 43 53 65 71 3a 31 31 37 33   ac.za..CSeq:1173
0110  39 20 42 59 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f   9 BYE..m:<sip:co
0120  6d 31 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d   m1@ctech.ac.za>.
0130  0a 0d 0a                                          ...

Frame 59 (271 bytes on wire, 271 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                   Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6103 (6103),
                               Seq: 198743, Ack: 5353, Len: 217
Session Initiation Protocol
    Status line: SIP/2.0 200
        Status-Code: 200
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK53
        t:<sip:com2@ctech.ac.za>;tag=2905616230
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2905616230
        f:<sip:com1@ctech.ac.za>;tag=948411075
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 948411075
        i:2663031567com1@ctech.ac.za
        CSeq:11739 BYE
        m:<sip:com2@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 200  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK53\r\n
    t:<sip:com2@ctech.ac.za>;tag=2905616230\r\n
    f:<sip:com1@ctech.ac.za>;tag=948411075\r\n
    i:2663031567com1@ctech.ac.za\r\n
    CSeq:11739 BYE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  01 01 24 00 00 00 80 06 9a 34 9b ee 21 f1 9b ee   ..$......4..!...
0020  21 f5 13 c4 17 d7 00 03 08 57 00 00 14 e9 50 18   !........W....P.
0030  05 dc 4a 46 00 00 53 49 50 2f 32 2e 30 20 32 30   ..JF..SIP/2.0 20
```

```
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  35 33 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32 40   53..t:<sip:com2@
0080  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
0090  3d 32 39 30 35 36 31 36 32 33 30 0d 0a 66 3a 3c   =2905616230..f:<
00a0  73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e 61   sip:com1@ctech.a
00b0  63 2e 7a 61 3e 3b 74 61 67 3d 39 34 38 34 31 31   c.za>;tag=948411
00c0  30 37 35 0d 0a 69 3a 32 36 36 33 30 33 31 35 36   075..i:266303156
00d0  37 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a   7com1@ctech.ac.z
00e0  61 0d 0a 43 53 65 71 3a 31 31 37 33 39 20 42 59   a..CSeq:11739 BY
00f0  45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f 6d 32 40 63   E..m:<sip:com2@c
0100  74 65 63 68 2e 61 63 2e 7a 61 3e 0d 0a 0d 0a      tech.ac.za>....
```

B.6 SIP capture 2

```
Frame 12 (506 bytes on wire, 506 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6257 (6257), Dst Port: 5060 (5060),
                               Seq: 4353, Ack: 198004, Len: 452
Session Initiation Protocol
    Request line: INVITE sip:com2@ctech.ac.za SIP/2.0
        Method: INVITE
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204
        t:<sip:com2@ctech.ac.za>
        f:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        Max-Forwards:70
        i:351774618com1@ctech.ac.za
        CSeq:12824 INVITE
        m:<sip:com1@ctech.ac.za>
        Content-Disposition:session
        c:application/sdp
        l:152
    Message body
```

```
        Session Description Protocol
            Session Description Protocol Version (v): 0
            Owner/Creator, Session Id (o): com1 984950303 984950303 IN
            IP4 155.238.33.245
                Owner Username: com1
                Session ID: 984950303
                Session Version: 984950303
                Owner Network Type: IN
                Owner Address Type: IP4
                Owner Address: 155.238.33.245
            Session Name (s): -
            Connection Information (c): IN IP4 155.238.33.245
                Connection Network Type: IN
                Connection Address Type: IP4
                Connection Address: 155.238.33.245
            Time Description, active time (t): 0 0
                Session Start Time: 0
                Session Start Time: 0
            Session Attribute (a): sendrecv
            Media Description, name and address (m): audio 5004 RTP/AVP 18
                Media Type: audio
                Media Port: 5004
                Media Proto: RTP/AVP
                Media Format: 18
            Media Attribute (a): rtpmap:18 G729/8000
                Media Attribute Fieldname: rtpmap
                Media Attribute Value: 18 G729/8000
Session Initiation Protocol (SIP as raw text)
    INVITE sip:com2@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204\r\n
    t:<sip:com2@ctech.ac.za>\r\n
    f:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    Max-Forwards:70\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:12824 INVITE\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    Content-Disposition:session\r\n
    c:application/sdp\r\n
    l:152\r\n
    \r\n
```

```
v=0\r\n
o=com1 984950303 984950303 IN IP4 155.238.33.245\r\n
s=-\r\n
c=IN IP4 155.238.33.245\r\n
t=0 0\r\n
a=sendrecv\r\n
m=audio 5004 RTP/AVP 18\r\n
a=rtpmap:18 G729/8000\r\n
```

```
0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  01 ec 18 00 00 00 80 06 a5 49 9b ee 21 f5 9b ee   .........I..!...
0020  21 f1 18 71 13 c4 00 00 11 01 00 03 05 74 50 18   !..q.........tP.
0030  05 dc a5 6a 00 00 49 4e 56 49 54 45 20 73 69 70   ...j..INVITE sip
0040  3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a   :com2@ctech.ac.z
0050  61 20 53 49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50   a SIP/2.0..v:SIP
0060  2f 32 2e 30 2f 54 43 50 20 31 35 35 2e 32 33 38   /2.0/TCP 155.238
0070  2e 33 33 2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a   .33.245;branch=z
0080  39 68 47 34 62 4b 32 30 34 0d 0a 74 3a 3c 73 69   9hG4bK204..t:<si
0090  70 3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63 2e   p:com2@ctech.ac.
00a0  7a 61 3e 0d 0a 66 3a 3c 73 69 70 3a 63 6f 6d 31   za>..f:<sip:com1
00b0  40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61   @ctech.ac.za>;ta
00c0  67 3d 32 36 39 35 30 37 37 36 34 34 0d 0a 4d 61   g=2695077644..Ma
00d0  78 2d 46 6f 72 77 61 72 64 73 3a 37 30 0d 0a 69   x-Forwards:70..i
00e0  3a 33 35 31 37 37 34 36 31 38 63 6f 6d 31 40 63   :351774618com1@c
00f0  74 65 63 68 2e 61 63 2e 7a 61 0d 0a 43 53 65 71   tech.ac.za..CSeq
0100  3a 31 32 38 32 34 20 49 4e 56 49 54 45 0d 0a 6d   :12824 INVITE..m
0110  3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68   :<sip:com1@ctech
0120  2e 61 63 2e 7a 61 3e 0d 0a 43 6f 6e 74 65 6e 74   .ac.za>..Content
0130  2d 44 69 73 70 6f 73 69 74 69 6f 6e 3a 73 65 73   -Disposition:ses
0140  73 69 6f 6e 0d 0a 63 3a 61 70 70 6c 69 63 61 74   sion..c:applicat
0150  69 6f 6e 2f 73 64 70 0d 0a 6c 3a 31 35 32 0d 0a   ion/sdp..l:152..
0160  0d 0a 76 3d 30 0d 0a 6f 3d 63 6f 6d 31 20 39 38   ..v=0..o=com1 98
0170  34 39 35 30 33 30 33 20 39 38 34 39 35 30 33 30   4950303 98495030
0180  33 20 49 4e 20 49 50 34 20 31 35 35 2e 32 33 38   3 IN IP4 155.238
0190  2e 33 33 2e 32 34 35 0d 0a 73 3d 2d 0d 0a 63 3d   .33.245..s=-..c=
01a0  49 4e 20 49 50 34 20 31 35 35 2e 32 33 38 2e 33   IN IP4 155.238.3
01b0  33 2e 32 34 35 0d 0a 74 3d 30 20 30 0d 0a 61 3d   3.245..t=0 0..a=
01c0  73 65 6e 64 72 65 63 76 0d 0a 6d 3d 61 75 64 69   sendrecv..m=audi
01d0  6f 20 35 30 30 34 20 52 54 50 2f 41 56 50 20 31   o 5004 RTP/AVP 1
01e0  38 0d 0a 61 3d 72 74 70 6d 61 70 3a 31 38 20 47   8..a=rtpmap:18 G
```

```
01f0  37 32 39 2f 38 30 30 30 0d 0a                     729/8000..

Frame 14 (275 bytes on wire, 275 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                  Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6257 (6257),
                               Seq: 198004, Ack: 4805, Len: 221
Session Initiation Protocol
    Status line: SIP/2.0 180
        Status-Code: 180
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204
        t:<sip:com2@ctech.ac.za>;tag=2397581388
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2397581388
        f:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        i:351774618com1@ctech.ac.za
        CSeq:12824 INVITE
        m:<sip:com2@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 180  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204\r\n
    t:<sip:com2@ctech.ac.za>;tag=2397581388\r\n
    f:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:12824 INVITE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  01 05 18 00 00 00 80 06 a6 30 9b ee 21 f1 9b ee   .........0..!...
0020  21 f5 13 c4 18 71 00 03 05 74 00 00 12 c5 50 18   !....q...t....P.
0030  05 dc cb ee 00 00 53 49 50 2f 32 2e 30 20 31 38   ......SIP/2.0 18
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  32 30 34 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32   204..t:<sip:com2
```

```
0080  40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61   @ctech.ac.za>;ta
0090  67 3d 32 33 39 37 35 38 31 33 38 38 0d 0a 66 3a   g=2397581388..f:
00a0  3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e   <sip:com1@ctech.
00b0  61 63 2e 7a 61 3e 3b 74 61 67 3d 32 36 39 35 30   ac.za>;tag=26950
00c0  37 37 36 34 34 0d 0a 69 3a 33 35 31 37 37 34 36   77644..i:3517746
00d0  31 38 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e   18com1@ctech.ac.
00e0  7a 61 0d 0a 43 53 65 71 3a 31 32 38 32 34 20 49   za..CSeq:12824 I
00f0  4e 56 49 54 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f   NVITE..m:<sip:co
0100  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d   m2@ctech.ac.za>.
0110  0a 0d 0a                                          ...
```

```
Frame 20 (480 bytes on wire, 480 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                  Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6257 (6257),
                               Seq: 198225, Ack: 4805, Len: 426
Session Initiation Protocol
    Status line: SIP/2.0 200
        Status-Code: 200
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204
        t:<sip:com2@ctech.ac.za>;tag=2397581388
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2397581388
        f:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        i:351774618com1@ctech.ac.za
        CSeq:12824 INVITE
        m:<sip:com2@ctech.ac.za>
        Content-Disposition:session
        c:application/sdp
        l:150
    Message body
        Session Description Protocol
            Session Description Protocol Version (v): 0
            Owner/Creator, Session Id (o): com2 78777731 78777731 IN
            IP4 155.238.33.241
                Owner Username: com2
```

```
                Session ID: 78777731
                Session Version: 78777731
                Owner Network Type: IN
                Owner Address Type: IP4
                Owner Address: 155.238.33.241
            Session Name (s): -
            Connection Information (c): IN IP4 155.238.33.241
                Connection Network Type: IN
                Connection Address Type: IP4
                Connection Address: 155.238.33.241
            Time Description, active time (t): 0 0
                Session Start Time: 0
                Session Start Time: 0
            Session Attribute (a): sendrecv
            Media Description, name and address (m): audio 5004 RTP/AVP 18
                Media Type: audio
                Media Port: 5004
                Media Proto: RTP/AVP
                Media Format: 18
            Media Attribute (a): rtpmap:18 G729/8000
                Media Attribute Fieldname: rtpmap
                Media Attribute Value: 18 G729/8000
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 200  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK204\r\n
    t:<sip:com2@ctech.ac.za>;tag=2397581388\r\n
    f:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:12824 INVITE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    Content-Disposition:session\r\n
    c:application/sdp\r\n
    l:150\r\n
    \r\n
    v=0\r\n
    o=com2 78777731 78777731 IN IP4 155.238.33.241\r\n
    s=-\r\n
    c=IN IP4 155.238.33.241\r\n
    t=0 0\r\n
    a=sendrecv\r\n
```

```
    m=audio 5004 RTP/AVP 18\r\n
    a=rtpmap:18 G729/8000\r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   .....D..E.
0010  01 d2 1a 00 00 00 80 06 a3 63 9b ee 21 f1 9b ee   .........c..!...
0020  21 f5 13 c4 18 71 00 03 06 51 00 00 12 c5 50 18   !....q...Q....P.
0030  05 dc fa d0 00 00 53 49 50 2f 32 2e 30 20 32 30   ......SIP/2.0 20
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  32 30 34 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32   204..t:<sip:com2
0080  40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61   @ctech.ac.za>;ta
0090  67 3d 32 33 39 37 35 38 31 33 38 38 0d 0a 66 3a   g=2397581388..f:
00a0  3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e   <sip:com1@ctech.
00b0  61 63 2e 7a 61 3e 3b 74 61 67 3d 32 36 39 35 30   ac.za>;tag=26950
00c0  37 37 36 34 34 0d 0a 69 3a 33 35 31 37 37 34 36   77644..i:3517746
00d0  31 38 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e   18com1@ctech.ac.
00e0  7a 61 0d 0a 43 53 65 71 3a 31 32 38 32 34 20 49   za..CSeq:12824 I
00f0  4e 56 49 54 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f   NVITE..m:<sip:co
0100  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d   m2@ctech.ac.za>.
0110  0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f 73 69   .Content-Disposi
0120  74 69 6f 6e 3a 73 65 73 73 69 6f 6e 0d 0a 63 3a   tion:session..c:
0130  61 70 70 6c 69 63 61 74 69 6f 6e 2f 73 64 70 0d   application/sdp.
0140  0a 6c 3a 31 35 30 0d 0a 0d 0a 76 3d 30 0d 0a 6f   .l:150....v=0..o
0150  3d 63 6f 6d 32 20 37 38 37 37 37 37 33 31 20 37   =com2 78777731 7
0160  38 37 37 37 37 33 31 20 49 4e 20 49 50 34 20 31   8777731 IN IP4 1
0170  35 35 2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a 73   55.238.33.241..s
0180  3d 2d 0d 0a 63 3d 49 4e 20 49 50 34 20 31 35 35   =-..c=IN IP4 155
0190  2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a 74 3d 30   .238.33.241..t=0
01a0  20 30 0d 0a 61 3d 73 65 6e 64 72 65 63 76 0d 0a    0..a=sendrecv..
01b0  6d 3d 61 75 64 69 6f 20 35 30 30 34 20 52 54 50   m=audio 5004 RTP
01c0  2f 41 56 50 20 31 38 0d 0a 61 3d 72 74 70 6d 61   /AVP 18..a=rtpma
01d0  70 3a 31 38 20 47 37 32 39 2f 38 30 30 30 0d 0a   p:18 G729/8000..

Frame 22 (308 bytes on wire, 308 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                    Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6257 (6257), Dst Port: 5060 (5060),
                               Seq: 4805, Ack: 198651, Len: 254
```

```
Session Initiation Protocol
    Request line: ACK sip:com2@ctech.ac.za SIP/2.0
        Method: ACK
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK211
        t:<sip:com2@ctech.ac.za>;tag=2397581388
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 2397581388
        f:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        Max-Forwards:70
        i:351774618com1@ctech.ac.za
        CSeq:12824 ACK
        m:<sip:com1@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    ACK sip:com2@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK211\r\n
    t:<sip:com2@ctech.ac.za>;tag=2397581388\r\n
    f:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    Max-Forwards:70\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:12824 ACK\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  01 26 1b 00 00 00 80 06 a3 0f 9b ee 21 f5 9b ee   .&..........!...
0020  21 f1 18 71 13 c4 00 00 12 c5 00 03 07 fb 50 18   !..q..........P.
0030  05 dc 0d 58 00 00 41 43 4b 20 73 69 70 3a 63 6f   ...X..ACK sip:co
0040  6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 20 53   m2@ctech.ac.za S
0050  49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50 2f 32 2e   IP/2.0..v:SIP/2.
0060  30 2f 54 43 50 20 31 35 35 2e 32 33 38 2e 33 33   0/TCP 155.238.33
0070  2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a 39 68 47   .245;branch=z9hG
0080  34 62 4b 32 31 31 0d 0a 74 3a 3c 73 69 70 3a 63   4bK211..t:<sip:c
0090  6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e   om2@ctech.ac.za>
00a0  3b 74 61 67 3d 32 33 39 37 35 38 31 33 38 38 0d   ;tag=2397581388.
00b0  0a 66 3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65   .f:<sip:com1@cte
00c0  63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67 3d 32 36   ch.ac.za>;tag=26
00d0  39 35 30 37 37 36 34 34 0d 0a 4d 61 78 2d 46 6f   95077644..Max-Fo
```

```
00e0  72 77 61 72 64 73 3a 37 30 0d 0a 69 3a 33 35 31   rwards:70..i:351
00f0  37 37 34 36 31 38 63 6f 6d 31 40 63 74 65 63 68   774618com1@ctech
0100  2e 61 63 2e 7a 61 0d 0a 43 53 65 71 3a 31 32 38   .ac.za..CSeq:128
0110  32 34 20 41 43 4b 0d 0a 6d 3a 3c 73 69 70 3a 63   24 ACK..m:<sip:c
0120  6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e   om1@ctech.ac.za>
0130  0d 0a 0d 0a                                       ....

Frame 68 (305 bytes on wire, 305 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                   Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 6257 (6257),
                               Seq: 198651, Ack: 5059, Len: 251
Session Initiation Protocol
    Request line: BYE sip:com1@ctech.ac.za SIP/2.0
        Method: BYE
    Message Header
        v:SIP/2.0/TCP 155.238.33.241;branch=z9hG4bK5
        t:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP to address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        f:<sip:com2@ctech.ac.za>;tag=2397581388
            SIP from address: <sip:com2@ctech.ac.za>
            SIP tag: 2397581388
        Max-Forwards:70
        i:351774618com1@ctech.ac.za
        CSeq:2984 BYE
        m:<sip:com2@ctech.ac.za>
Session Initiation Protocol (SIP as raw text)
    BYE sip:com1@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.241;branch=z9hG4bK5\r\n
    t:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    f:<sip:com2@ctech.ac.za>;tag=2397581388\r\n
    Max-Forwards:70\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:2984 BYE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
```

```
0010  01 23 22 00 00 00 80 06 9c 12 9b ee 21 f1 9b ee   .#".........!...
0020  21 f5 13 c4 18 71 00 03 07 fb 00 00 13 c3 50 18   !....q........P.
0030  05 dc dc f7 00 00 42 59 45 20 73 69 70 3a 63 6f   ......BYE sip:co
0040  6d 31 40 63 74 65 63 68 2e 61 63 2e 7a 61 20 53   m1@ctech.ac.za S
0050  49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50 2f 32 2e   IP/2.0..v:SIP/2.
0060  30 2f 54 43 50 20 31 35 35 2e 32 33 38 2e 33 33   0/TCP 155.238.33
0070  2e 32 34 31 3b 62 72 61 6e 63 68 3d 7a 39 68 47   .241;branch=z9hG
0080  34 62 4b 35 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d   4bK5..t:<sip:com
0090  31 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74   1@ctech.ac.za>;t
00a0  61 67 3d 32 36 39 35 30 37 37 36 34 34 0d 0a 66   ag=2695077644..f
00b0  3a 3c 73 69 70 3a 63 6f 6d 32 40 63 74 65 63 68   :<sip:com2@ctech
00c0  2e 61 63 2e 7a 61 3e 3b 74 61 67 3d 32 33 39 37   .ac.za>;tag=2397
00d0  35 38 31 33 38 38 0d 0a 4d 61 78 2d 46 6f 72 77   581388..Max-Forw
00e0  61 72 64 73 3a 37 30 0d 0a 69 3a 33 35 31 37 37   ards:70..i:35177
00f0  34 36 31 38 63 6f 6d 31 40 63 74 65 63 68 2e 61   4618com1@ctech.a
0100  63 2e 7a 61 0d 0a 43 53 65 71 3a 32 39 38 34 20   c.za..CSeq:2984
0110  42 59 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f 6d 32   BYE..m:<sip:com2
0120  40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d 0a 0d   @ctech.ac.za>...
0130  0a                                                 .

Frame 72 (269 bytes on wire, 269 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 6257 (6257), Dst Port: 5060 (5060),
                               Seq: 5059, Ack: 198902, Len: 215
Session Initiation Protocol
    Status line: SIP/2.0 200
        Status-Code: 200
    Message Header
        v:SIP/2.0/TCP 155.238.33.241;branch=z9hG4bK5
        t:<sip:com1@ctech.ac.za>;tag=2695077644
            SIP to address: <sip:com1@ctech.ac.za>
            SIP tag: 2695077644
        f:<sip:com2@ctech.ac.za>;tag=2397581388
            SIP from address: <sip:com2@ctech.ac.za>
            SIP tag: 2397581388
        i:351774618com1@ctech.ac.za
        CSeq:2984 BYE
        m:<sip:com1@ctech.ac.za>
```

```
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 200  \r\n
    v:SIP/2.0/TCP 155.238.33.241;branch=z9hG4bK5\r\n
    t:<sip:com1@ctech.ac.za>;tag=2695077644\r\n
    f:<sip:com2@ctech.ac.za>;tag=2397581388\r\n
    i:351774618com1@ctech.ac.za\r\n
    CSeq:2984 BYE\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    \r\n

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  00 ff 26 00 00 00 80 06 98 36 9b ee 21 f5 9b ee   ..&......6..!...
0020  21 f1 18 71 13 c4 00 00 13 c3 00 03 08 f6 50 18   !..q..........P.
0030  05 dc 69 5a 00 00 53 49 50 2f 32 2e 30 20 32 30   ..iZ..SIP/2.0 20
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  31 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   1;branch=z9hG4bK
0070  35 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 31 40 63   5..t:<sip:com1@c
0080  74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67 3d   tech.ac.za>;tag=
0090  32 36 39 35 30 37 37 36 34 34 0d 0a 66 3a 3c 73   2695077644..f:<s
00a0  69 70 3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63   ip:com2@ctech.ac
00b0  2e 7a 61 3e 3b 74 61 67 3d 32 33 39 37 35 38 31   .za>;tag=2397581
00c0  33 38 38 0d 0a 69 3a 33 35 31 37 37 34 36 31 38   388..i:351774618
00d0  63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a 61   com1@ctech.ac.za
00e0  0d 0a 43 53 65 71 3a 32 39 38 34 20 42 59 45 0d   ..CSeq:2984 BYE.
00f0  0a 6d 3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65   .m:<sip:com1@cte
0100  63 68 2e 61 63 2e 7a 61 3e 0d 0a 0d 0a           ch.ac.za>....
```

B.7 SDP offer and answer in SIP messages

```
Frame 8 (506 bytes on wire, 506 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:5f, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.245 (155.238.33.245),
                   Dst Addr: 155.238.33.241 (155.238.33.241)
Transmission Control Protocol, Src Port: 8119 (8119), Dst Port: 5060 (5060),
                               Seq: 4626, Ack: 200271, Len: 452
Session Initiation Protocol
    Request line: INVITE sip:com2@ctech.ac.za SIP/2.0
        Method: INVITE
```

```
Message Header
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK96
    t:<sip:com2@ctech.ac.za>
    f:<sip:com1@ctech.ac.za>;tag=4093811437
        SIP from address: <sip:com1@ctech.ac.za>
        SIP tag: 4093811437
    Max-Forwards:70
    i:1503896443com1@ctech.ac.za
    CSeq:11842 INVITE
    m:<sip:com1@ctech.ac.za>
    Content-Disposition:session
    c:application/sdp
    l:152
Message body
    Session Description Protocol
        Session Description Protocol Version (v): 0
        Owner/Creator, Session Id (o): com1 976560700 976560700 IN IP4
        155.238.33.245
            Owner Username: com1
            Session ID: 976560700
            Session Version: 976560700
            Owner Network Type: IN
            Owner Address Type: IP4
            Owner Address: 155.238.33.245
        Session Name (s): -
        Connection Information (c): IN IP4 155.238.33.245
            Connection Network Type: IN
            Connection Address Type: IP4
            Connection Address: 155.238.33.245
        Time Description, active time (t): 0 0
            Session Start Time: 0
            Session Start Time: 0
        Session Attribute (a): sendonly
        Media Description, name and address (m): audio 5004 RTP/AVP 18
            Media Type: audio
            Media Port: 5004
            Media Proto: RTP/AVP
            Media Format: 18
        Media Attribute (a): rtpmap:18 G729/8000
            Media Attribute Fieldname: rtpmap
```

```
                Media Attribute Value: 18 G729/8000
Session Initiation Protocol (SIP as raw text)
    INVITE sip:com2@ctech.ac.za SIP/2.0\r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK96\r\n
    t:<sip:com2@ctech.ac.za>\r\n
    f:<sip:com1@ctech.ac.za>;tag=4093811437\r\n
    Max-Forwards:70\r\n
    i:1503896443com1@ctech.ac.za\r\n
    CSeq:11842 INVITE\r\n
    m:<sip:com1@ctech.ac.za>\r\n
    Content-Disposition:session\r\n
    c:application/sdp\r\n
    l:152\r\n
    \r\n
    v=0\r\n
    o=com1 976560700 976560700 IN IP4 155.238.33.245\r\n
    s=-\r\n
    c=IN IP4 155.238.33.245\r\n
    t=0 0\r\n
    a=sendonly\r\n
    m=audio 5004 RTP/AVP 18\r\n
    a=rtpmap:18 G729/8000\r\n

0000  00 00 1a 18 fd 44 00 00 1a 18 fd 5f 08 00 45 00   .....D....._..E.
0010  01 ec 36 00 00 00 80 06 87 49 9b ee 21 f5 9b ee   ..6......I..!...
0020  21 f1 1f b7 13 c4 00 00 12 12 00 03 0e 4f 50 18   !............OP.
0030  05 dc f0 cc 00 00 49 4e 56 49 54 45 20 73 69 70   ......INVITE sip
0040  3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a   :com2@ctech.ac.z
0050  61 20 53 49 50 2f 32 2e 30 0d 0a 76 3a 53 49 50   a SIP/2.0..v:SIP
0060  2f 32 2e 30 2f 54 43 50 20 31 35 35 2e 32 33 38   /2.0/TCP 155.238
0070  2e 33 33 2e 32 34 35 3b 62 72 61 6e 63 68 3d 7a   .33.245;branch=z
0080  39 68 47 34 62 4b 39 36 0d 0a 74 3a 3c 73 69 70   9hG4bK96..t:<sip
0090  3a 63 6f 6d 32 40 63 74 65 63 68 2e 61 63 2e 7a   :com2@ctech.ac.z
00a0  61 3e 0d 0a 66 3a 3c 73 69 70 3a 63 6f 6d 31 40   a>..f:<sip:com1@
00b0  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
00c0  3d 34 30 39 33 38 31 31 34 33 37 0d 0a 4d 61 78   =4093811437..Max
00d0  2d 46 6f 72 77 61 72 64 73 3a 37 30 0d 0a 69 3a   -Forwards:70..i:
00e0  31 35 30 33 38 39 36 34 34 33 63 6f 6d 31 40 63   1503896443com1@c
00f0  74 65 63 68 2e 61 63 2e 7a 61 0d 0a 43 53 65 71   tech.ac.za..CSeq
0100  3a 31 31 38 34 32 20 49 4e 56 49 54 45 0d 0a 6d   :11842 INVITE..m
```

```
0110  3a 3c 73 69 70 3a 63 6f 6d 31 40 63 74 65 63 68   :<sip:com1@ctech
0120  2e 61 63 2e 7a 61 3e 0d 0a 43 6f 6e 74 65 6e 74   .ac.za>..Content
0130  2d 44 69 73 70 6f 73 69 74 69 6f 6e 3a 73 65 73   -Disposition:ses
0140  73 69 6f 6e 0d 0a 63 3a 61 70 70 6c 69 63 61 74   sion..c:applicat
0150  69 6f 6e 2f 73 64 70 0d 0a 6c 3a 31 35 32 0d 0a   ion/sdp..l:152..
0160  0d 0a 76 3d 30 0d 0a 6f 3d 63 6f 6d 31 20 39 37   ..v=0..o=com1 97
0170  36 35 36 30 37 30 30 20 39 37 36 35 36 30 37 30   6560700 97656070
0180  30 20 49 4e 20 49 50 34 20 31 35 35 2e 32 33 38   0 IN IP4 155.238
0190  2e 33 33 2e 32 34 35 0d 0a 73 3d 2d 0d 0a 63 3d   .33.245..s=-..c=
01a0  49 4e 20 49 50 34 20 31 35 35 2e 32 33 38 2e 33   IN IP4 155.238.3
01b0  33 2e 32 34 35 0d 0a 74 3d 30 20 30 0d 0a 61 3d   3.245..t=0 0..a=
01c0  73 65 6e 64 6f 6e 6c 79 0d 0a 6d 3d 61 75 64 69   sendonly..m=audi
01d0  6f 20 35 30 30 34 20 52 54 50 2f 41 56 50 20 31   o 5004 RTP/AVP 1
01e0  38 0d 0a 61 3d 72 74 70 6d 61 70 3a 31 38 20 47   8..a=rtpmap:18 G
01f0  37 32 39 2f 38 30 30 30 0d 0a                    729/8000..
```

```
Frame 19 (481 bytes on wire, 481 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:00:1a:18:fd:5f
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                    Dst Addr: 155.238.33.245 (155.238.33.245)
Transmission Control Protocol, Src Port: 5060 (5060), Dst Port: 8119 (8119),
                                Seq: 200491, Ack: 5078, Len: 427
Session Initiation Protocol
    Status line: SIP/2.0 200
        Status-Code: 200
    Message Header
        v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK96
        t:<sip:com2@ctech.ac.za>;tag=976564031
            SIP to address: <sip:com2@ctech.ac.za>
            SIP tag: 976564031
        f:<sip:com1@ctech.ac.za>;tag=4093811437
            SIP from address: <sip:com1@ctech.ac.za>
            SIP tag: 4093811437
        i:1503896443com1@ctech.ac.za
        CSeq:11842 INVITE
        m:<sip:com2@ctech.ac.za>
        Content-Disposition:session
        c:application/sdp
        l:152
    Message body
```

```
        Session Description Protocol
            Session Description Protocol Version (v): 0
            Owner/Creator, Session Id (o): com2 552864949 552864949 IN IP4
            155.238.33.241
                Owner Username: com2
                Session ID: 552864949
                Session Version: 552864949
                Owner Network Type: IN
                Owner Address Type: IP4
                Owner Address: 155.238.33.241
            Session Name (s): -
            Connection Information (c): IN IP4 155.238.33.241
                Connection Network Type: IN
                Connection Address Type: IP4
                Connection Address: 155.238.33.241
            Time Description, active time (t): 0 0
                Session Start Time: 0
                Session Start Time: 0
            Session Attribute (a): recvonly
            Media Description, name and address (m): audio 5004 RTP/AVP 18
                Media Type: audio
                Media Port: 5004
                Media Proto: RTP/AVP
                Media Format: 18
            Media Attribute (a): rtpmap:18 G729/8000
                Media Attribute Fieldname: rtpmap
                Media Attribute Value: 18 G729/8000
Session Initiation Protocol (SIP as raw text)
    SIP/2.0 200  \r\n
    v:SIP/2.0/TCP 155.238.33.245;branch=z9hG4bK96\r\n
    t:<sip:com2@ctech.ac.za>;tag=976564031\r\n
    f:<sip:com1@ctech.ac.za>;tag=4093811437\r\n
    i:1503896443com1@ctech.ac.za\r\n
    CSeq:11842 INVITE\r\n
    m:<sip:com2@ctech.ac.za>\r\n
    Content-Disposition:session\r\n
    c:application/sdp\r\n
    l:152\r\n
    \r\n
    v=0\r\n
```

```
o=com2 552864949 552864949 IN IP4 155.238.33.241\r\n
s=-\r\n
c=IN IP4 155.238.33.241\r\n
t=0 0\r\n
a=recvonly\r\n
m=audio 5004 RTP/AVP 18\r\n
a=rtpmap:18 G729/8000\r\n
```

```
0000  00 00 1a 18 fd 5f 00 00 1a 18 fd 44 08 00 45 00   ....._.....D..E.
0010  01 d3 33 00 00 00 80 06 8a 62 9b ee 21 f1 9b ee   ..3......b..!...
0020  21 f5 13 c4 1f b7 00 03 0f 2b 00 00 13 d6 50 18   !........+....P.
0030  05 dc cd 74 00 00 53 49 50 2f 32 2e 30 20 32 30   ...t..SIP/2.0 20
0040  30 20 20 0d 0a 76 3a 53 49 50 2f 32 2e 30 2f 54   0  ..v:SIP/2.0/T
0050  43 50 20 31 35 35 2e 32 33 38 2e 33 33 2e 32 34   CP 155.238.33.24
0060  35 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b   5;branch=z9hG4bK
0070  39 36 0d 0a 74 3a 3c 73 69 70 3a 63 6f 6d 32 40   96..t:<sip:com2@
0080  63 74 65 63 68 2e 61 63 2e 7a 61 3e 3b 74 61 67   ctech.ac.za>;tag
0090  3d 39 37 36 35 36 34 30 33 31 0d 0a 66 3a 3c 73   =976564031..f:<s
00a0  69 70 3a 63 6f 6d 31 40 63 74 65 63 68 2e 61 63   ip:com1@ctech.ac
00b0  2e 7a 61 3e 3b 74 61 67 3d 34 30 39 33 38 31 31   .za>;tag=4093811
00c0  34 33 37 0d 0a 69 3a 31 35 30 33 38 39 36 34 34   437..i:150389644
00d0  33 63 6f 6d 31 40 63 74 65 63 68 2e 61 63 2e 7a   3com1@ctech.ac.z
00e0  61 0d 0a 43 53 65 71 3a 31 31 38 34 32 20 49 4e   a..CSeq:11842 IN
00f0  56 49 54 45 0d 0a 6d 3a 3c 73 69 70 3a 63 6f 6d   VITE..m:<sip:com
0100  32 40 63 74 65 63 68 2e 61 63 2e 7a 61 3e 0d 0a   2@ctech.ac.za>..
0110  43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f 73 69 74   Content-Disposit
0120  69 6f 6e 3a 73 65 73 73 69 6f 6e 0d 0a 63 3a 61   ion:session..c:a
0130  70 70 6c 69 63 61 74 69 6f 6e 2f 73 64 70 0d 0a   pplication/sdp..
0140  6c 3a 31 35 32 0d 0a 0d 0a 76 3d 30 0d 0a 6f 3d   l:152....v=0..o=
0150  63 6f 6d 32 20 35 35 32 38 36 34 39 34 39 20 35   com2 552864949 5
0160  35 32 38 36 34 39 34 39 20 49 4e 20 49 50 34 20   52864949 IN IP4
0170  31 35 35 2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a   155.238.33.241..
0180  73 3d 2d 0d 0a 63 3d 49 4e 20 49 50 34 20 31 35   s=-..c=IN IP4 15
0190  35 2e 32 33 38 2e 33 33 2e 32 34 31 0d 0a 74 3d   5.238.33.241..t=
01a0  30 20 30 0d 0a 61 3d 72 65 63 76 6f 6e 6c 79 0d   0 0..a=recvonly.
01b0  0a 6d 3d 61 75 64 69 6f 20 35 30 30 34 20 52 54   .m=audio 5004 RT
01c0  50 2f 41 56 50 20 31 38 0d 0a 61 3d 72 74 70 6d   P/AVP 18..a=rtpm
01d0  61 70 3a 31 38 20 47 37 32 39 2f 38 30 30 30 0d   ap:18 G729/8000.
01e0  0a                                                .
```

B.8 ABNF of SDP implementation

The session description protocol grammar as used in this program can be represented in Augmented BNF as follows: Notes: The names in the following Augmented BNF grammar match those in appendix A of the SDP RFC (Handley & Jacobson 1998) for easy comparison. Augmented BNF is defined in RFC 2234 (Crocker & Overell 1997).

> 'A session description consists of a session-level description (details that apply to the whole session and all media streams) and optionally several media-level descriptions (details that apply onto to a single media stream).'(Handley & Jacobson 1998)

```
announcement =          proto-version
                        origin-field
                        session-name-field
                        connection-field
                        time-fields
                        attribute-fields
                        media-descriptions

proto-version =         "v=" 1*DIGIT CRLF
                        ;The current SDP version is 0, therefore 0 is the
                        ;only valid option.

origin-field =          "o=" username space sess-id space sess-version
                        space nettype space addrtype space addr CRLF

session-name-field =    "s=" text CRLF
                        ;RFC 3264 recommends that a single space or dash
                        ;(hyphen) be used for the session name field for
                        ;unicast streams.  Only hyphens are supported.

connection-field =      "c=" nettype space addrtype space
                        connection-address CRLF
                        ;According to the SDP specification, there must
                        ;be a connection field at the session-level or in
                        ;every media description.

time-fields =           "t=" start-time space stop-time CRLF
                        ;The unicast sessions are created and destroyed
```

```
                        ;by SIP.  The time field is therefore set to
                        ;"0 0" as stated in RFC 3264.  The parser only
                        ;checks for "0 0".  If the first 3 characters do
                        ;not equal "0 0" then the field is considered
                        ;invalid.

attribute-fields =      "a=" attribute CRLF
                        ;This implementation requires a session-level
                        ;attribute field (stream type) and one per media
                        ;description (rtpmap).

media-descriptions =    media-field attribute-fields
                        ;Only one media description per announcement

media-field =           "m=" media space port space proto space fmt CRLF

media =                 1*(alpha-numeric)
                        ;"audio" is inserted when building the packet.
                        ;Currently this field is not checked in received
                        ;session descriptions.

fmt =                   1*(alpha-numeric)
                        ;In this implementation, this field contains the
                        ;RTP payload type.

proto =                 1*(alpha-numeric)
                        ;In this implementation, this field contains
                        ;"RTP/AVP" referring to the protocol RTP and its
                        ;audio and visual profile.

port =                  1*(DIGIT)

attribute =             (att-field ":" att-value) | att-field

att-field =             1*(alpha-numeric)

att-value =             byte-string

sess-id =               1*(DIGIT)
                        ;sess-id and sess-version have both been
```

```
                    ;implemented as 32 bit words with an initial
                    ;maximum of (2**30)-1.

sess-version =      1*(DIGIT)

connection-address = addr

start-time =        "0"
                    ;See time-fields note

stop-time =         "0"
                    ;See time-fields note

username =          safe

nettype =           "IN"

addrtype =          "IP4"
                    ;The software does not support IP6.

addr =              unicast-address

unicast-address =   IP4-address

IP4-address =       decimal-uchar "." decimal-uchar "." decimal-uchar
                    "." decimal-uchar
                    ;Unicast addresses are not enforced, though the
                    ;system will not operate in a multicast
                    ;environment.

text =              byte-string
                    ;All text is ASCII

byte-string =       1*(0x01..0x09|0x0b|0x0c|0x0e..0xff)
                    ;any byte except NUL, CR or LF

decimal-uchar =     DIGIT
                    | POS-DIGIT DIGIT
                    | ("1" 2*(DIGIT))
                    | ("2" ("0"|"1"|"2"|"3"|"4") DIGIT)
```

```
                    | ("2" "5" ("0"|"1"|"2"|"3"|"4"|"5"))

alpha-numeric =     ALPHA | DIGIT


DIGIT =             "0" | POS-DIGIT


POS-DIGIT =         "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"


ALPHA =             "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|
                    "l"|"m"|"n"|"o "|"p"|"q"|"r"|"s"|"t"|"u"|"v"|
                    "w"|"x"|"y"|"z"|"A"|"B"|"C "|"D"|"E"|"F"|"G"|
                    "H"|"I"|"J"|"K"|"L"|"M"|"N"|"O"|"P"|" Q"|"R"|
                    "S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"


safe =              alpha-numeric |
                    "'" | "'" | "-" | "." | "/" | ":" | "?" | """ |
                    "#" | "$" | "&" | "*" | ";" | "=" | "@" | "[" |
                    "]" | "^" | "_" | "`" | "{" | "|" | "}" | "+" |
                    "~" | "
                    ;Fields are not currently checked for safe
                    ;characters


space =             %d32
tab =               %d9
CRLF =              %d13.10
```

B.9 SIPSDP.H

The header file from the author's SIP and SDP implementation. Function protoypes are in the following section.

```
/* SIP and SDP header file */
/* Shaun Kaplan */

typedef unsigned char  u_int8;
typedef signed   char  int8;
```

```
typedef unsigned short u_int16;
typedef signed   short int16;
typedef unsigned long  u_int32;
typedef signed   long  int32;

/*typedef union _IPADDR {
        unsigned char   addr[4];
        unsigned long   address;
} IPADDR;*/

/* Task IDs and Priorities */
#define TU_TSK_ID     11
#define TU_TSK_PRI    11
#define TXN_TSK_ID    12
#define TXN_TSK_PRI   12
#define TPORT_TSK_ID  13
#define TPORT_TSK_PRI 13

/* Address of timer2count register */
#define T2CNT       0xFF60
/*
 * Random number generator constants
 * from: Sedgewick, R.  Algorithms in C.  pp 511-514
 * adapted for 16bit machine
 */
#define M 10000
#define m1 100
#define b 5821

/* Minimum 'valid' received packet length */
#define MINLENGTH  120
/* Max length of received packet */
#define MAXLENGTH  520
/* Length of full sip version, i.e. "SIP/x.x" */
#define VERLENGTH 7
/* SIP version 2.0 */
#define SIPVER 0x20

/* Address book constants */
/* Address book not implemented */
```

```
#define ADDR_BK  01
#define SIP_LKUP 02
#define ENTRIES  04

/* User agent type */
#define UAC 0
#define UAS 1
/* Transaction type */
#define INVITET    0
#define NONINVITET 1
/* Transaction states */
#define CALLING    0
#define TRYING     1
#define PROCEEDING 2
#define COMPLETED  3
#define CONFIRMED  4
#define TERMINATED 5

/* frominterface : messages from the user interface to the TU */
#define BUSYO       0
#define ACTIVATE    1
#define PLACE_CALL  2
#define ANSWER_CALL 3
#define END_CALL    4

/* tu2interface : messages from the TU to the user interface */
#define BUSYO         0
#define RCVD1XX       1
//#define ALREADYACTIVE 2
#define CALLINPLACE   3
#define CANTANSWER    4
#define ENDSESSION    5
#define STARTSESSION  6
#define CANTENDCALL   7
//#define ENDCALL       8
#define RINGING       9

/*
 * t2isubmessage : TU to interface sub-messages (only RCVD1XX uses sub-
 * messages)
```

```
 */
#define R1TRYING  0
#define R1RINGING 1
#define R1XX      2

/* tport2tu : Transport to TU messages */
#define BUSY0           0
#define TRANSPORTERROR 1
#define RCVDREQUEST    2
#define SENT           3
#define CLOSED         4
#define NORMAL         5

/* txn2tu : Messages fro the transaction layre to the TU */
#define BUSY0           0
#define TRANSPORTERROR  1
#define TIMEOUT         2
#define RECEIVED1XX     3
#define RECEIVED2XX     4
#define RECEIVED3456XX  5
#define UNEXPECTEDERROR 6
#define RCVDACK         7
#define CLIENTDESTROY   8
#define SERVERDESTROY   9
#define INVALID         10

/* transport_task : Transport task options*/
#define SEND        0
#define RECEIVE     1
#define STOPRECEIVE 2
#define END         3
#define LISTEN0     4

/* Maximum size of a method string, e.g. INVITE */
#define MAXMETHOD 5
/* Maximum number of times a SIP message may be forwarded */
#define MAXFWDS   70
/* Maximum tag size */
#define MAXTAG    11
```

```
/* SIP methods */
typedef enum
{
  ERROR,
  INVITE,
  ACK,
  CANCEL,
  BYE
} method_t;

/* Dialogue ID */
typedef struct
{
  u_int8  call_id[31];
  u_int8  remote_tag[MAXTAG];
  u_int8  local_tag[MAXTAG];
} d_id_t;

/* Dialogue state - see comments below */
typedef struct
{
  u_int8        dialog;
  d_id_t        dialog_id;
  u_int32       local_seq_no;
  u_int32       remote_seq_no;
  u_int8  far  *local_uri;
  u_int8  far  *remote_uri;
  /*u_int8    secure*/
  u_int8        route_set;
} dialog_t;

/*
 * Client state
 *
 * dialog state:
 *  dialog (boolean indicating whether in dialog or not - allows me to
 *    store state in dialog before the dialog has begun.  Don't need to
 *    store info in two locations.  This isn't a problem because this
 *    implementation will only allow for one dialog (and memory is
 *    precious).)
```

```
 * dialog_id.Call_id (from request)
 *         .remote_tag (To tag from response)
 *         .local_tag (From tag from request)
 * local_seq_no (CSeq no. from request)
 * remote_seq_no (init:empty, established when remote ua sends request in
 *   dialog)
 * local_uri (From field)
 * remote_uri (To field)
 * remote_target (uri from Contact in response)
 * [secure] (boolean flag indicating SIPS usage)
 * route_set (set to empty - never traversing servers)
 */

/*
 * Server state
 *
 * dialog state:
 *  dialog (boolean indicating whether in dialog or not - allows me to
 *    store state in dialog before the dialog has begun.  Don't need to
 *    store info in two locations.  This isn't a problem because this
 *    implementation will only allow for one dialog (and memory is
 *    precious).)
 * dialog_id.Call_id (from request)
 *         .remote_tag (From tag from request)
 *         .local_tag (To tag from response to request)
 * local_seq_no (init:empty)
 * remote_seq_no (CSeq no. from request)
 * local_uri (To field)
 * remote_uri (From field)
 * remote_target (uri from Contact in request)
 * [secure] (boolean flag indicating SIPS usage)
 * route_set (set to empty - never traversing servers)
 */


/* SIP structure contains all necessary state for two unit only usage */
typedef struct
{
  u_int8      ua_type;              /* UA core type */
  u_int8      transaction_type;     /* transaction type */
```

```
                                /* Transaction finite state machine info */
u_int8      uac_invite_state;    /* UAC INVITE transaction state */
u_int8      uac_noninvite_state; /* UAC nonINVITE transaction state */
u_int8      uas_invite_state;    /* UAS INVITE transaction state */
u_int8      uas_noninvite_state; /* UAS nonINVITE transaction state */
/* TRANSPORT */
IPADDR      remote_ip;           /* IP address of remote unit*/
u_int16     remote_port;         /* Remote port number */
IPADDR      local_ip;            /* Local IP address */
u_int16     local_port;          /* Local port number */
u_int8      myuri[21];           /* local SIP URI */
u_int8      theiruri[21];        /* remote SIP URI */
u_int8      tporttype[4];        /* Transport protocol used*/
u_int16     rcvd_statuscode;     /* Status code of received response */
method_t    rcvd_method;         /* Method of received request */
u_int8      calling_task;        /* ID of task that resumed the
                                    tranport layer task so that control
                                    can be resumed to that task */
u_int16     rcvdelay;            /* Delay between checking for received
                                    packets*/
int16       id;                  /* socket id */
/* GENERAL*/
u_int8      sipver;              /* SIP version 0xMMmm where MM is
                                    major version and mm is minor
                                    version */
u_int8      maxforwards;         /* Max forwards variable */
/* INTER TASK COMMUNICATION */
u_int8      frominterface;       /* Message from interface */
u_int8      tu2interface;        /* Message for interface */
u_int8      t2isubmessage;       /* Sub-message for interface */
u_int8      tport2tu;            /* Message from transport to TU */
u_int8      transaction2tu;      /* Message from transaction to TU */
u_int8      transport_task;      /* Task for transport to perform */
u_int8      transport_status;    /* Transport layer status message */
                                 /* Previous message from each task */
u_int8  lastfrominterface;
u_int8  lastfromtransaction;
u_int8  lastfromtransport;
u_int8  lastfromtu;
```

```
  u_int16     send_status_code;    /* Status code of response that TU
                                      wants transaction to send to
                                      transport layer */
  method_t    sent_method;         /* Method sent by client */
  /* STATE */
  dialog_t    dialog;              /* Dialogue state */
  u_int8      viabranch[11];       /* Via branch value */

} sip_t;

/* RTP event type used for running SIP with RTP */
typedef enum
{
  START_SESSION,
  END_SESSION,
  SSRC_COLLISION,
  ADJUSTMENT
}event_type;

/* RTP operation modes used for running SIP with RTP */
/* Note these are still test modes */
typedef enum
{
  RTP_TX,
  RTCP_TX,
  RTP_RTCP_TX,
  RTP_RX,
  RTCP_RX,
  RTP_RTCP_RX,
  H_DUPLEX_SEND,   /* send rtp, send and rx rtcp */
  H_DUPLEX_RX,     /* rx rtp, send and rx rtcp */
  FULL_DUPLEX,
  FULL_DUPLEX_TEST /* send and rx rtp, no rtcp */
}rtpmode_type;

/* Informatoin for communicating with RTP tasks */
typedef struct
{
  u_int16      rtp_port_l;
  u_int16      rtp_port_r;
```

```
  u_int8       rx_pt;
  u_int8       tx_pt;
  event_type   event;
  IPADDR       rtp_dest_ip;
  u_int8       flag;
  u_int8  stream[9];
  rtpmode_type rtpmode;
}interface_t;

/* Address book structures - not implemented */
/*
typedef struct
{
  u_int8 type;
  u_int8 subtype;
  int8   name[9];
  int8   sipuri[21];
}address_book_entry;   //max entries: 4


typedef struct
{
  u_int8 type;
  u_int8 subtype;
  int8    sipuri[21];
  IPADDR  ip;
  u_int16 port;
}sip_lookup_entry;     //max entries: 4
*/

/*-------------------------------------------------------------------*/
/*                              SDP.h                                */
/*-------------------------------------------------------------------*/

/* Maximum size of SDP description */
#define MAXSDP 168
/* G.729 clock rate */
#define G729CLK_RATE 8000
/* SDP types */
#define OFFER 0
```

```
#define ANSWER 1

/*
 * SDP structure
 */
/* Informatoin about the media */
typedef struct
{
  u_int32 port;       /* Port number */
  u_int8  pt;         /* RTP payload type */
  //u_int16 clkrate;
} media_t;

/* SDP originator's ID */
typedef struct
{
  u_int8 username[15];    /* max 14 char (excl. NULL) assuming a minimum
                             "@domain" of @x.xxx (@1.com) and max URI of
                             20 char (excl. NULL) */
  u_int8 session_id[11];  /* SDP session ID */
  u_int8 network_type[3]; /* Network type - Internet */
  u_int8 address_type[4]; /* Address type - IP */
  IPADDR address;         /* Originator's IP address */
} id_t;

/* SDP message */
typedef struct
{
  id_t    session;
  u_int8  version[11];     /* SDP session version number */
  u_int8  stream_type[9];  /* sendrecv, sendonly, recvonly, inactive */
  media_t media;
} message_t;

/* Structure to hold the data for an offer and answer */
typedef struct
{
  message_t type[2];
} sdp_t;
```

B.10 SIP and SDP function prototypes

```
/*
 * SIP_initial_invite()
 * Function to build the initial INVITE.
 * Includes generating required state.
 * Argumnents: sip is a pointer to the SIP structure
 *             sdp is a pointer to the SDP body for the INVITE
 *             The INVITE is returned in message
 */
u_int8 far *SIP_initial_invite(u_int8 far *message, u_int8 far *sdp,
                               sip_t far *sip);

/*
 * SIPget_totag()
 * Extracts to tag from response and returns it in tag.
 * Assumes to field has only one parameter - the to tag.
 */
u_int8 far *SIPget_totag(u_int8 far *tag, u_int8 far *response);

/*
 * SIPmatch_response()
 * Function to test whether a response matches a request used to create a
 * transaction.  This is done to determine whether the message is passed
 * to a (the) client transaction.
 * Returns TRUE if response matches request, else returns FALSE
 */
u_int8  SIPmatch_response(u_int8 far *sip, u_int8 far *branch,
                          int8 *method[], method_t mthd);

/*
 * SIPmatch_request()
 * Function to test whether a request matches a response used in a server
 * transaction.  This is done to determine whether the message is passed
 * to a (the) server transaction.
 * Returns TRUE if request belongs to a transaction, else returns FALSE
 */
```

```
u_int8  SIPmatch_request(u_int8 far *sip, u_int8 far *branch,
                         IPADDR sentby, int8 *method[], method_t mthd);
/*
 * SIPget_statuscode()
 * Extracts the status code from the status line in a response and
 * returns the value as a u_int16.  Returns 0 if error (value not an
 * integer).  Function assumes that there are no spaces before the
 * demarcation between SIPVER and STATUSCODE.
 */
u_int16 SIPget_statuscode(u_int8 *message);

/*
 * SIPget_method()
 * Extracts the method from the request line in a request and returns the
 * value as a method_t.  Returns 0 if error.
 */
u_int8 SIPget_method(u_int8 *message);

/* Address book functions - not implemented */
/*void SIP_AB_display_book(void);
void SIP_AB_add_contact(void);
void SIP_AB_get_uri(void);*/

/*
 * Build SIP request
 * Function will build a SIP request using the data passed to it.
 * The message is returned in 'request'
 * NOTE : Only SDP sessions are sent in the message body
 */
int8 far *SIP_build_request(u_int8 far *request,
                            method_t mthd, u_int8 far *requesturi,
                            u_int8 sipver,
                            u_int8 far *transporttype, IPADDR sentby,
                            u_int8 far *viabranch,
                            u_int8 far *touri, u_int8 far *totag,
                            u_int8 far *fromuri, u_int8 far *fromtag,
                            u_int8 maxforwards,
                            u_int8 far *callid,
                            u_int32 cseq,
                            u_int8 far *contacturi,
```

```
                                u_int8 far *body);

/*
 * Build SIP response
 * Function will build a SIP response using the data passed to it.
 * The message is returned in 'response'
 * NOTE : Only SDP sessions are sent in the message body
 */
u_int8 far *SIP_build_response(int8 far *response,
                               u_int8 sipver, u_int16 statuscode,
                               u_int8 far *transporttype, IPADDR sentby,
                               u_int8 far *viabranch,
                               u_int8 far *touri, u_int8 far *totag,
                               u_int8 far *fromuri, u_int8 far *fromtag,
                               u_int8 far *callid,
                               u_int32 cseq, method_t mthd,
                               u_int8 far *contacturi,
                               u_int8 far *body);

/*
 * Create sip uri
 * Input  : [name@]host sip uri
 * Output : Fully formatted SIP URI
 */
int8 far *sip_uri(int8 far *dest, const int8 far *uri);

/*
 * build_request_line
 * Input  : method, request uri, sip version
 * Output : Correctly formatted request line
 */
int8 *build_request_line(int8 *requestline,
                         method_t mthd, int8 far *requesturi,
                         u_int8 sipver);

/*
 * build_status_line
 * Input  : sip version, status code
 * Output : Correctly formatted status line (no reason given - just SPACE)
 */
```

```
int8 *build_status_line(int8 *statusline,
                        u_int8 SIPver, u_int16 statuscode);

/*
 * SIPextractbody()
 * Function to extract sdp offer / answer
 * Will only extract if message contains session body
 */
u_int8 far *SIPextractbody(u_int8 far *sdpa, u_int8 far *p_sip);

/*
 * SIP_analyse_response()
 * Analyses a response and extracts dialog state
 * Returns TRUE else FALSE if any errors.
 */
u_int8 SIP_analyse_response(u_int8 far *response, sip_t far *SIPdata);

/*
 * SIP_analyse_request()
 * Analyses a request and extracts dialog state
 */
u_int8 SIP_analyse_request(u_int8 far *request, sip_t far *SIPdata);

/*
 * SIP_req_match_dialog()
 * Function will check whether a request (usually a BYE) matches a dialog.
 * Inputs are the request and dialog info.
 * Also check if cseq > remote_seq_no (if there is a remote_seq_no)
 * Returns TRUE if match or FALSE.
 */
u_int32 SIP_req_match_dialog(u_int8 far *request, u_int8 dialog,
                             u_int8 far *call_id, u_int8 far *remote_tag,
                             u_int8 far *local_tag, u_int32 remote_seq);
/*
 * SIPgeneratebranch()
 * Function to generate the branch parameter for the via field.
 * As this implementation is based on RFC3261, the first 7 characters
 * will be "z9hG4bK".
 * The branch can be a maximum of 13 chars including the terminating null.
 */
```

```
void  SIPgeneratebranch(u_int8 far *viabranch);

/*
 * SIPgenerate_tag()
 * Tags need to have at least 32 bits of randomness
 * Tags are 11 characters long including the terminating null
 */
void  SIPgeneratetag(u_int8 far *tag);

/*
 * SIPgeneratecallid()
 * Call-id is being implemented as the local sip uri prefixed by a 32 bit
 * random number.  Therefore in this implementation the maximum size of
 * call-id is 31 bytes including the terminating null.
 */
void  SIPgeneratecallid(u_int8 far *call_id, u_int8 far *local_uri);

/*
 * SIPgeneratecseq()
 * The sequence number value MUST be expressible as a 32-bit unsigned
 * integer and MUST be less than 2**31.
 */
u_int32  SIPgeneratecseq(void);

/*
 * SIPrequestvalid()
 * Function that returns TRUE if the SIP version in the request line is
 * SIP/2.0
 */
u_int8  SIPrequestvalid(u_int8 *sip, u_int8 sipver);

/*
 * SIPresponsevalid()
 * Function that returns TRUE if the SIP version in the status line is
 * SIP/2.0
 */
u_int8  SIPresponsevalid(u_int8 *sip, u_int8 sipver);


/*
```

```
 * SDP_initial_offer
 * Builds the initial SDP offer from parameters required to fill fields
 * Outputs SDP session
 */
u_int8 far *SDP_intitial_offer(u_int8 far *offer, sdp_t far *sdp,
                               u_int8 far *sipuri, IPADDR myip,
                               u_int8 far *stream, u_int16 streamport,
                               u_int8 streampt);

/*
 * Build SDP session for answer
 * Takes as input the parameters required to fill fields, received SDP
 * session
 * Outputs SDP session
 */
u_int8 far *SDP_answer(u_int8 far *answer, u_int8 far *offer,
                       sdp_t far *sdp, u_int8 streampt,
                       u_int8 far *sipuri, IPADDR myip);

/*
 * Analyse SDP answer
 * Function will analyse an SDP answer to detemine whether negotiation
 * was successful
 */
u_int8 SDP_analyse_answer(u_int8 far *answer, sdp_t far *sdp);

/*
 * Build SDP session
 * Takes as input the paramters required to fill fields
 * Output : SDP session, code indicating correct completion
 */
u_int8 far *SDP_build(u_int8 far *description, sdp_t far *sdp,
                      u_int8 offans);

/*
 * SDP_parse
 * This function will extract the required data from the SDP fields.
 * SDP_parse will check that the SDP form is valid but not the content of
 * the fields.  This must be checked by the calling function.
 */
```

```
u_int8 SDP_parse(u_int8 far *recv, u_int8 *fields[]);

/*
 * SDPget_username()
 * Function to extract the username from a SIP URI if present.  A hyphen
 * '-' is used if no username is present.
 */
u_int8 far *SDPget_username(u_int8 far *username, u_int8 far *sipuri);

/*
 * SDPget_sess_id()
 * Create a session ID
 * sess-id and sess-version are 32bit with max (2**30)-1
 */
u_int8 far *SDPget_sess_id(u_int8 far *session_id);

/*
 * SDPget_encname()
 * Get encoder name from payload type
 */
u_int8 far *SDPget_encname(u_int8 far *encname, u_int8 pt);

/* Miscellaneous functions */
void  display_title(void);
u_int8 main_menu(void);
u_int32  generate_32bit_value(void);
int16 mult(int16 p, int16 q);
int16 randomint(int16 r);
u_int8 getseed(void);
```

Appendix C

Media transport and QoS

C.1 Codec board schematic

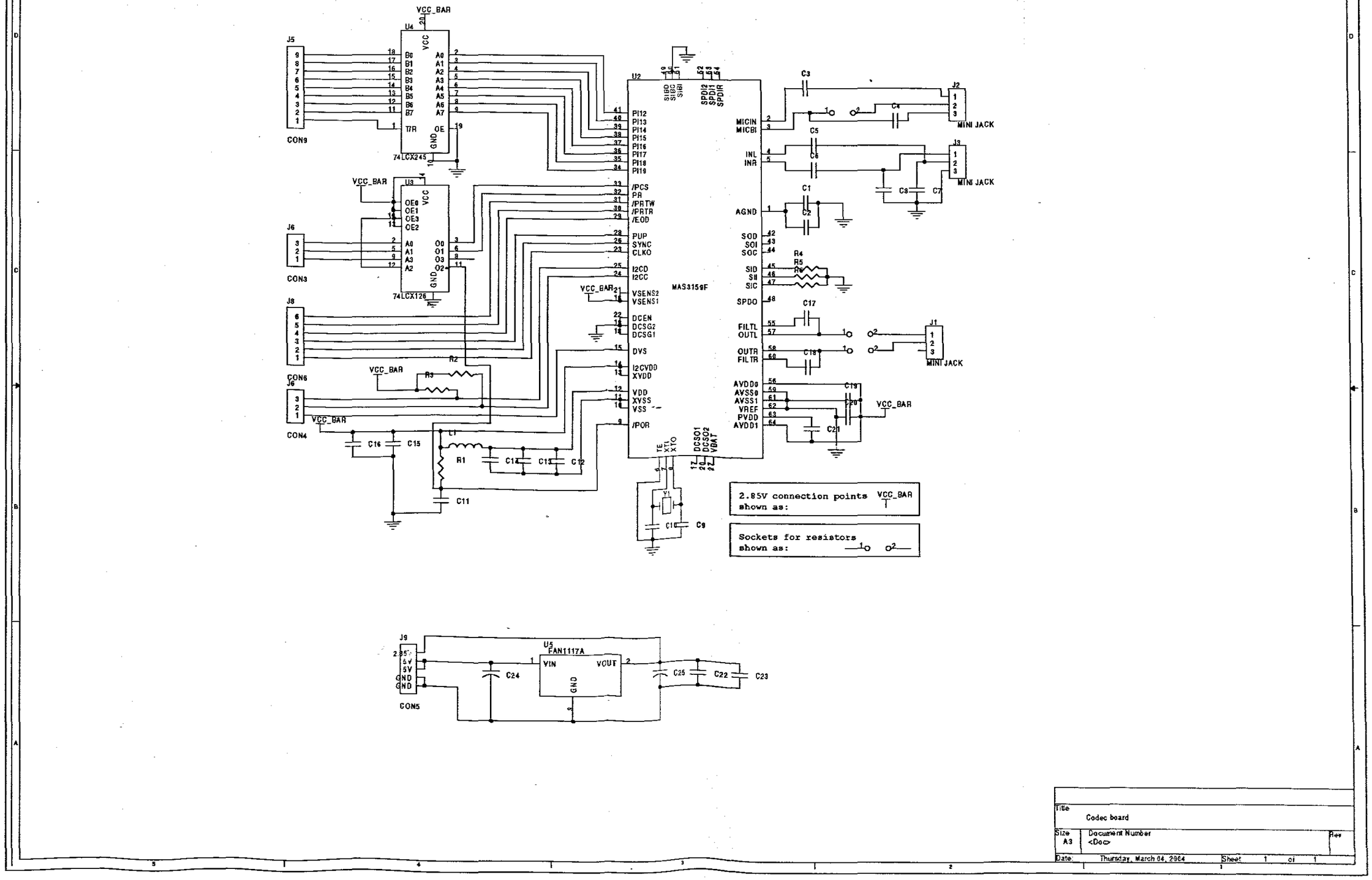


Figure C.1: The schematic for the codec boards.

C.2 MAS3159F oscilloscope captures

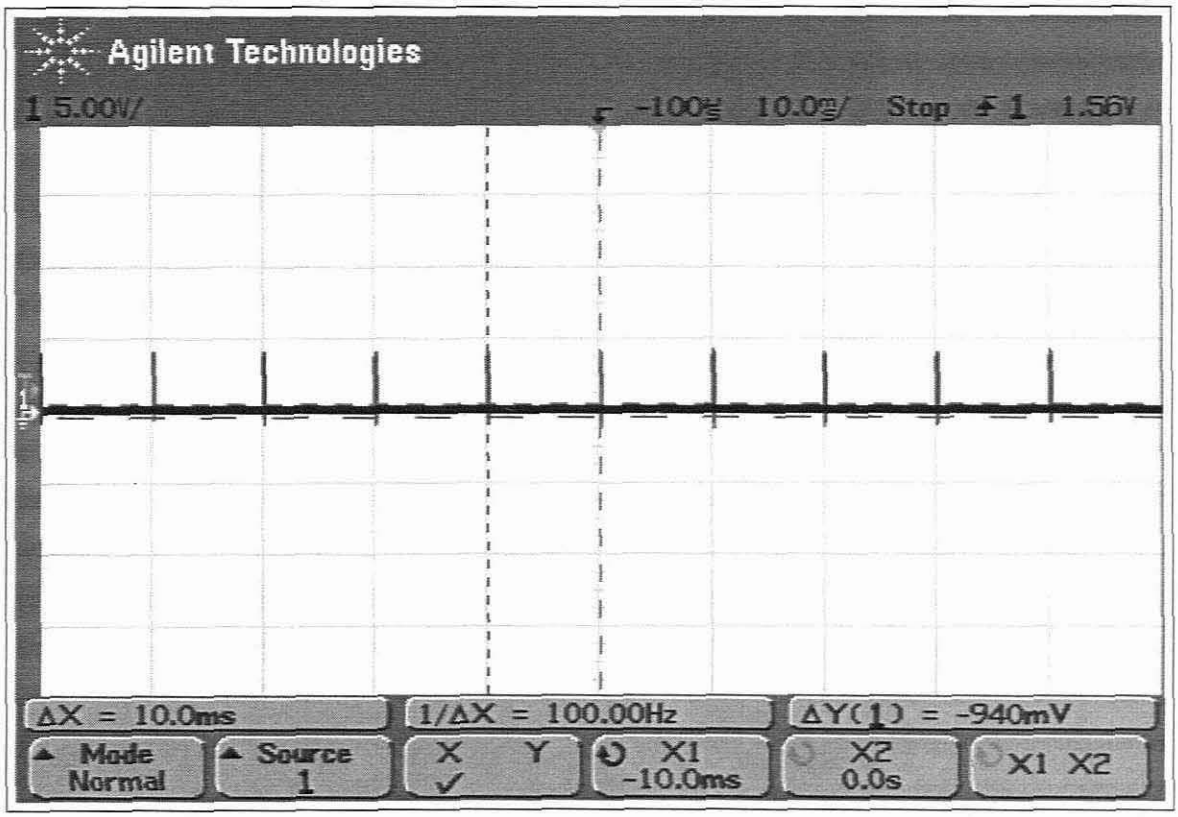


Figure C.2: This capture shows the intervals at which encoded G.729 frames are sent to the controller. The trace is of the $\overline{\text{RTW}}$ line measured after the inverter.

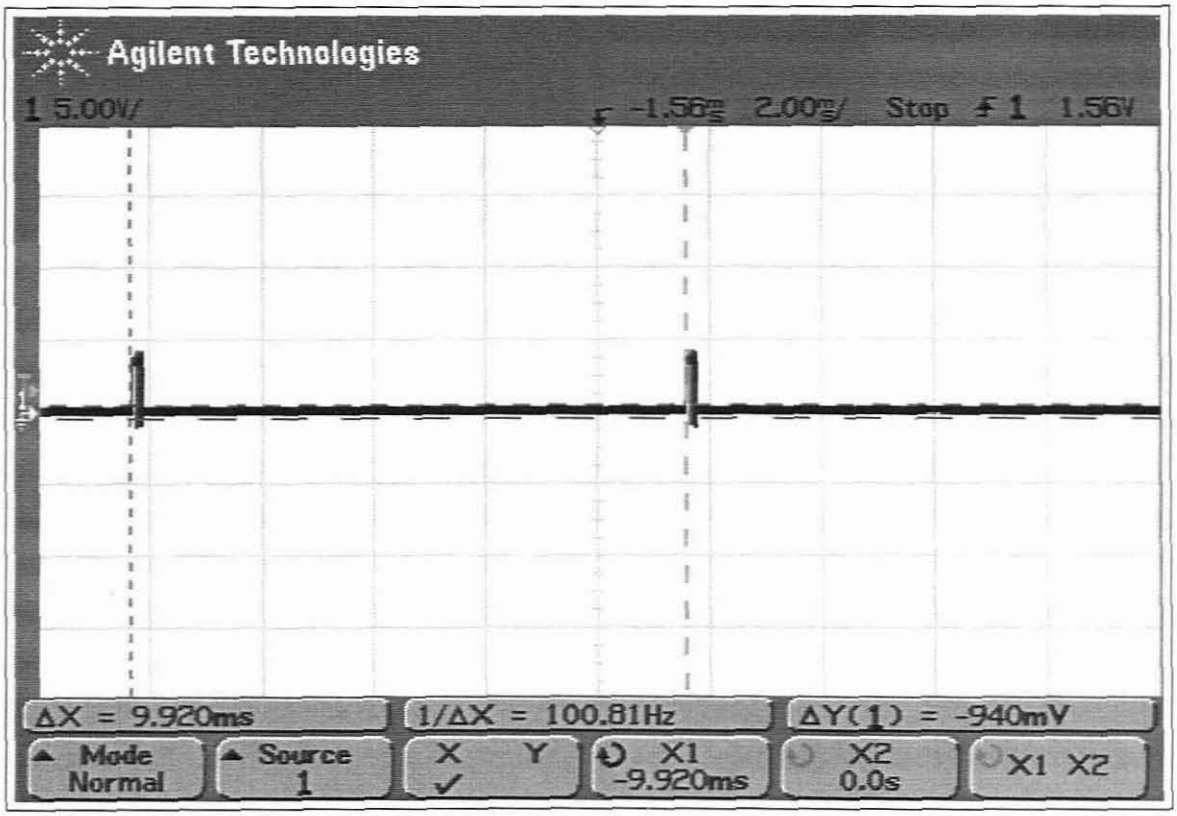


Figure C.3: This capture shows the time between two encoded G.729 frames being sent to the controller. The trace is of the $\overline{\text{RTW}}$ line measured after the inverter.

C.3 RTP SSRC collision detection

```
Frame 132 (78 bytes on wire, 78 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:06:5b:9e:ed:36
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                   Dst Addr: 155.238.33.96 (155.238.33.96)
User Datagram Protocol, Src Port: 5005 (5005), Dst Port: 5005 (5005)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0000 = Reception report count: 0
    Packet type: Receiver Report (201)
    Length: 1
    Sender SSRC: 1116178395
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
```

```
    Length: 6
    Chunk 1, SSRC/CSRC 1116178395
        Identifier: 1116178395
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.241

0000  00 06 5b 9e ed 36 00 00 1a 18 fd 44 08 00 45 00   ..[..6.....D..E.
0010  00 40 1c 00 00 00 80 11 a3 7f 9b ee 21 f1 9b ee   .@..........!...
0020  21 60 13 8d 13 8d 00 2c 5e 84 80 c9 00 01 42 87   !`.....,^.....B.
0030  87 db 81 ca 00 06 42 87 87 db 01 0e 31 35 35 2e   ......B.....155.
0040  32 33 38 2e 33 33 2e 32 34 31 00 00 00 00         238.33.241....

Frame 145 (78 bytes on wire, 78 bytes captured)
Ethernet II, Src: 00:06:5b:9e:ed:36, Dst: 00:00:1a:18:fd:44
Internet Protocol, Src Addr: 155.238.33.96 (155.238.33.96),
                  Dst Addr: 155.238.33.241 (155.238.33.241)
User Datagram Protocol, Src Port: 1221 (1221), Dst Port: 5005 (5005)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0000 = Reception report count: 0
    Packet type: Receiver Report (201)
    Length: 1
    Sender SSRC: 1116178395
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 6
    Chunk 1, SSRC/CSRC 2394602249
        Identifier: 2394602249
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.245

0000  00 00 1a 18 fd 44 00 06 5b 9e ed 36 08 00 45 00   .....D..[..6..E.
```

```
0010  00 40 f9 0a 00 00 80 11 c6 74 9b ee 21 60 9b ee   .@.......t..!`..
0020  21 f1 04 c5 13 8d 00 2c ed e6 80 c9 00 01 42 87   !......,......B.
0030  87 db 81 ca 00 06 8e ba bb 09 01 0e 31 35 35 2e   ............155.
0040  32 33 38 2e 33 33 2e 32 34 35 00 00 00 00         238.33.245....

Frame 146 (86 bytes on wire, 86 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:06:5b:9e:ed:36
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                  Dst Addr: 155.238.33.96 (155.238.33.96)
User Datagram Protocol, Src Port: 5005 (5005), Dst Port: 1221 (1221)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0000 = Reception report count: 0
    Packet type: Receiver Report (201)
    Length: 1
    Sender SSRC: 1116178395
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 6
    Chunk 1, SSRC/CSRC 1116178395
        Identifier: 1116178395
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.241
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Goodbye (203)
    Length: 1
    Identifier: 1116178395

0000  00 06 5b 9e ed 36 00 00 1a 18 fd 44 08 00 45 00   ..[..6.....D..E.
0010  00 48 1d 00 00 00 80 11 a2 77 9b ee 21 f1 9b ee   .H.......w..!...
0020  21 60 13 8d 04 c5 00 34 21 0d 80 c9 00 01 42 87   !`.....4!.....B.
```

```
0030  87 db 81 ca 00 06 42 87 87 db 01 0e 31 35 35 2e   ......B.....155.
0040  32 33 38 2e 33 33 2e 32 34 31 00 00 00 00 81 cb   238.33.241......
0050  00 01 42 87 87 db                                 ..B...

Frame 155 (78 bytes on wire, 78 bytes captured)
Ethernet II, Src: 00:00:1a:18:fd:44, Dst: 00:06:5b:9e:ed:36
Internet Protocol, Src Addr: 155.238.33.241 (155.238.33.241),
                  Dst Addr: 155.238.33.96 (155.238.33.96)
User Datagram Protocol, Src Port: 5005 (5005), Dst Port: 5005 (5005)
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0000 = Reception report count: 0
    Packet type: Receiver Report (201)
    Length: 1
    Sender SSRC: 4097968208
Real-time Transport Control Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 6
    Chunk 1, SSRC/CSRC 4097968208
        Identifier: 4097968208
        SDES items
            Type: CNAME (user and domain) (1)
            Length: 14
            Text: 155.238.33.241

0000  00 06 5b 9e ed 36 00 00 1a 18 fd 44 08 00 45 00   ..[..6.....D..E.
0010  00 40 1e 00 00 00 80 11 a1 7f 9b ee 21 f1 9b ee   .@..........!...
0020  21 60 13 8d 13 8d 00 2c fa 23 80 c9 00 01 f4 42   !`.....,.#.....B
0030  08 50 81 ca 00 06 f4 42 08 50 01 0e 31 35 35 2e   .P.....B.P..155.
0040  32 33 38 2e 33 33 2e 32 34 31 00 00 00 00         238.33.241....
```

C.4 RTP session timing out

The RTCP packets are numbers: 33, 54, 79, 103, 114.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	Cisco_31:7e:cb	Spanning-tree-(for-bridges)_00	STP	Conf. R
2	0.022790	155.238.41.100	224.0.1.22	SRVLOC	Service Request
3	0.023820	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
4	0.024501	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
5	0.025019	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
6	0.025526	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
7	0.025958	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
8	0.026441	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
9	0.026945	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
10	0.027386	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
11	0.028009	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
12	0.028371	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
13	0.033249	155.238.41.100	224.0.1.22	SRVLOC	Attribute Request
14	0.384262	155.238.95.194	224.0.1.22	SRVLOC	Service Request
15	0.603652	155.238.41.167	224.0.1.22	SRVLOC	Service Request
16	1.226626	155.238.52.214	224.0.1.22	SRVLOC	Service Request
17	1.307943	155.238.33.1	Broadcast	ARP	Who has 155.238.33.2
18	1.603801	155.238.41.167	224.0.1.22	SRVLOC	Service Request
19	2.011101	Cisco_31:7e:cb	Spanning-tree-(for-bridges)_00	STP	Conf. R
20	2.068203	155.238.52.214	224.0.1.22	SRVLOC	Service Request
21	2.431868	155.238.41.167	224.0.1.22	SRVLOC	Service Request
22	2.683754	155.238.33.1	224.0.0.13	PIMv2	Hello
23	3.068050	155.238.52.214	224.0.1.22	SRVLOC	Service Request
24	3.072327	155.238.52.214	224.0.1.22	SRVLOC	Service Request
25	3.076456	155.238.52.214	224.0.1.22	SRVLOC	Attribute Request
26	3.447333	155.238.41.167	224.0.1.22	SRVLOC	Service Request
27	3.624085	155.238.40.60	224.0.1.22	SRVLOC	Service Request
28	3.624476	155.238.40.60	224.0.1.22	SRVLOC	Service Request
29	3.962670	155.238.33.1	Broadcast	ARP	Who has 155.238.33.2
30	4.022768	Cisco_31:7e:cb	Spanning-tree-(for-bridges)_00	STP	Conf. R
31	4.473858	155.238.40.60	224.0.1.22	SRVLOC	Service Request
32	4.474048	155.238.40.60	224.0.1.22	SRVLOC	Service Request
33	4.652081	155.238.33.241	155.238.33.245	RTCP	Receiver Report
34	4.988900	155.238.33.171	Broadcast	ARP	Who has 155.238.33.1
35	5.478540	155.238.41.167	224.0.1.22	SRVLOC	Service Request
36	5.489424	155.238.40.60	224.0.1.22	SRVLOC	Service Request
37	5.489620	155.238.40.60	224.0.1.22	SRVLOC	Service Request
38	5.614771	155.238.40.60	224.0.1.22	SRVLOC	Service Request

```
39 5.615083   155.238.40.60      224.0.1.22          SRVLOC   Service Request
40 6.038487   Cisco_31:7e:cb     Spanning-tree-(for-bridges)_00 STP      Conf. R
41 6.504928   155.238.40.60      224.0.1.22          SRVLOC   Service Request
42 6.505097   155.238.40.60      224.0.1.22          SRVLOC   Service Request
43 7.005849   155.238.33.1       224.0.0.1           IGMP     V2 Membership Query
44 7.520551   155.238.40.60      224.0.1.22          SRVLOC   Service Request
45 7.520696   155.238.40.60      224.0.1.22          SRVLOC   Service Request
46 7.886169   155.238.33.97      224.0.1.22          IGMP     V2 Membership Report
47 8.046684   Cisco_31:7e:cb     Spanning-tree-(for-bridges)_00 STP      Conf. R
48 8.595373   155.238.33.162     239.255.255.250     IGMP     V2 Membership Report
49 9.370625   155.238.4.134      155.238.33.190      TCP      524 > 1060 [ACK] Seq
50 9.540946   155.238.41.167     224.0.1.22          SRVLOC   Service Request
51 9.551708   155.238.40.60      224.0.1.22          SRVLOC   Service Request
52 9.551831   155.238.40.60      224.0.1.22          SRVLOC   Service Request
53 10.060069  Cisco_31:7e:cb     Spanning-tree-(for-bridges)_00 STP      Conf. R
54 10.902010  155.238.33.241     155.238.33.245      RTCP
                                                     Receiver Report
55 11.450531  155.238.33.1       Broadcast           ARP      Who has 155.238.33.7
56 11.505673  155.238.6.163      155.238.33.96       TCP      524 > 1037 [ACK] Seq
57 11.505772  DellComp_9e:ed:36  Broadcast           ARP      Who has 155.238.33.1
58 11.506361  155.238.33.1       DellComp_9e:ed:36   ARP      155.238.33.1 is at 0
59 11.506421  155.238.33.96      155.238.6.163       TCP      1037 > 524 [ACK] Seq
60 11.630177  155.238.41.167     224.0.1.22          SRVLOC   Service Request
61 11.631305  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
62 11.631799  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
63 11.632520  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
64 11.632864  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
65 11.633350  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
66 11.633873  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
67 11.634421  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
68 11.634795  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
69 11.635271  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
70 11.635784  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
71 11.636534  155.238.41.167     224.0.1.22          SRVLOC   Attribute Request
72 12.070659  Cisco_31:7e:cb     Spanning-tree-(for-bridges)_00 STP      Conf. R
73 12.657831  155.238.33.174     235.80.68.83        IGMP     V2 Membership Report
74 12.756468  155.238.4.133      155.238.33.238      TCP      822 > printer [SYN]
75 13.614068  155.238.40.60      224.0.1.22          SRVLOC   Service Request
76 13.614243  155.238.40.60      224.0.1.22          SRVLOC   Service Request
77 14.082606  Cisco_31:7e:cb     Spanning-tree-(for-bridges)_00 STP      Conf. R
```

```
 78 14.513287  155.238.4.147     155.238.33.79     UDP      Source port: 1032  D
 79 15.950842  155.238.33.241    155.238.33.245    RTCP
                                                   Receiver Report
 80 16.094577  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
 81 16.321968  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 82 16.322369  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 83 17.232253  155.238.4.133     224.0.1.22        SRVLOC   Attribute Request
 84 17.270248  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 85 17.270460  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 86 18.113151  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
 87 18.285853  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 88 18.286050  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 89 18.317380  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 90 18.317695  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 91 19.301317  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 92 19.301488  155.238.40.60     224.0.1.22        SRVLOC   Service Request
 93 19.513263  155.238.33.1      224.0.0.9         RIPv2    Response
 94 19.513729  155.238.33.1      224.0.0.9         RIPv2    Response
 95 19.513835  155.238.33.1      224.0.0.9         RIPv2    Response
 96 20.118521  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
 97 20.174832  155.238.33.1      Broadcast         ARP      Who has 155.238.33.2
 98 20.261935  155.238.4.234     155.238.33.239    TCP      807 > printer [SYN]
 99 20.316917  155.238.40.60     224.0.1.22        SRVLOC   Service Request
100 20.317069  155.238.40.60     224.0.1.22        SRVLOC   Service Request
101 20.417013  155.238.33.154    155.238.33.255    BROWSER  Host Announcement JA
102 22.130493  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
103 22.302347  155.238.33.241    155.238.33.245    RTCP
                                                   Receiver Report
104 22.348118  155.238.40.60     224.0.1.22        SRVLOC   Service Request
105 22.348284  155.238.40.60     224.0.1.22        SRVLOC   Service Request
106 22.706816  155.238.33.1      Broadcast         ARP      Who has 155.238.33.2
107 23.013371  155.238.94.177    224.0.1.22        SRVLOC   Service Request
108 23.815061  155.238.94.177    224.0.1.22        SRVLOC   Service Request
109 23.815775  155.238.94.177    224.0.1.22        SRVLOC   Service Request
110 23.816367  155.238.94.177    224.0.1.22        SRVLOC   Attribute Request
111 24.142477  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
112 24.932633  155.238.33.1      Broadcast         ARP      Who has 155.238.33.2
113 25.019727  155.238.10.6      155.238.33.236    ICMP     Echo (ping) request
114 25.650827  155.238.33.241    155.238.33.245    RTCP
                                                   Receiver Report
```

```
115 26.155659  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
116 26.410447  155.238.40.60     224.0.1.22          SRVLOC   Service Request
117 26.410633  155.238.40.60     224.0.1.22          SRVLOC   Service Request
118 26.918081  Cisco_fa:e3:d8    CDP/VTP             LLC      U, func = UI; SNAP,
119 28.166467  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
120 28.354764  155.238.40.60     224.0.1.22          SRVLOC   Service Request
121 28.355150  155.238.40.60     224.0.1.22          SRVLOC   Service Request
122 28.452507  155.238.33.1      Broadcast           ARP      Who has 155.238.33.2
123 29.254170  155.238.40.60     224.0.1.22          SRVLOC   Service Request
124 29.254360  155.238.40.60     224.0.1.22          SRVLOC   Service Request
125 29.937834  155.238.33.151    Broadcast           ARP      Who has 155.238.33.1
126 30.178380  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
127 30.269769  155.238.40.60     224.0.1.22          SRVLOC   Service Request
128 30.269972  155.238.40.60     224.0.1.22          SRVLOC   Service Request
129 30.348231  155.238.40.60     224.0.1.22          SRVLOC   Service Request
130 30.348534  155.238.40.60     224.0.1.22          SRVLOC   Service Request
131 30.490808  155.238.33.1      Broadcast           ARP      Who has 155.238.33.2
132 31.153051  155.238.33.2      Broadcast           ARP      Who has 155.238.33.7
133 31.285266  155.238.40.60     224.0.1.22          SRVLOC   Service Request
134 31.285420  155.238.40.60     224.0.1.22          SRVLOC   Service Request
135 32.190340  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
136 32.300851  155.238.40.60     224.0.1.22          SRVLOC   Service Request
137 32.300996  155.238.40.60     224.0.1.22          SRVLOC   Service Request
138 32.585576  155.238.33.1      224.0.0.13          PIMv2    Hello
139 34.202336  Cisco_31:7e:cb    Spanning-tree-(for-bridges)_00 STP      Conf. R
140 34.322638  155.238.33.1      Broadcast           ARP      Who has 155.238.33.2
141 34.332008  155.238.40.60     224.0.1.22          SRVLOC   Service Request
142 34.332194  155.238.40.60     224.0.1.22          SRVLOC   Service Request
143 36.116112  155.238.4.147     155.238.33.165      UDP      Source port: 1032  D
```

C.5 Appendix A from RFC 3550

Appendix A from the RTP specification (Schulzrinne et al. 2003).

```
Appendix A - Algorithms

   We provide examples of C code for aspects of RTP sender and receiver
   algorithms.  There may be other implementation methods that are
   faster in particular operating environments or have other advantages.
```

These implementation notes are for informational purposes only and are meant to clarify the RTP specification.

The following definitions are used for all examples; for clarity and brevity, the structure definitions are only valid for 32-bit big-endian (most significant octet first) architectures. Bit fields are assumed to be packed tightly in big-endian bit order, with no additional padding. Modifications would be required to construct a portable implementation.

```
/*
 * rtp.h  --  RTP header file
 */
#include <sys/types.h>

/*
 * The type definitions below are valid for 32-bit architectures and
 * may have to be adjusted for 16- or 64-bit architectures.
 */
typedef unsigned char  u_int8;
typedef unsigned short u_int16;
typedef unsigned int   u_int32;
typedef          short int16;

/*
 * Current protocol version.
 */
#define RTP_VERSION    2

#define RTP_SEQ_MOD (1<<16)
#define RTP_MAX_SDES 255      /* maximum text length for SDES */

typedef enum {
    RTCP_SR   = 200,
    RTCP_RR   = 201,
    RTCP_SDES = 202,
    RTCP_BYE  = 203,
    RTCP_APP  = 204
} rtcp_type_t;
```

```
typedef enum {
    RTCP_SDES_END   = 0,
    RTCP_SDES_CNAME = 1,
    RTCP_SDES_NAME  = 2,
    RTCP_SDES_EMAIL = 3,
    RTCP_SDES_PHONE = 4,
    RTCP_SDES_LOC   = 5,
    RTCP_SDES_TOOL  = 6,
    RTCP_SDES_NOTE  = 7,
    RTCP_SDES_PRIV  = 8
} rtcp_sdes_type_t;

/*
 * RTP data header
 */
typedef struct {
    unsigned int version:2;   /* protocol version */
    unsigned int p:1;         /* padding flag */
    unsigned int x:1;         /* header extension flag */
    unsigned int cc:4;        /* CSRC count */
    unsigned int m:1;         /* marker bit */
    unsigned int pt:7;        /* payload type */
    unsigned int seq:16;      /* sequence number */
    u_int32 ts;               /* timestamp */
    u_int32 ssrc;             /* synchronization source */
    u_int32 csrc[1];          /* optional CSRC list */
} rtp_hdr_t;

/*
 * RTCP common header word
 */
typedef struct {
    unsigned int version:2;   /* protocol version */
    unsigned int p:1;         /* padding flag */
    unsigned int count:5;     /* varies by packet type */
    unsigned int pt:8;        /* RTCP packet type */
    u_int16 length;           /* pkt len in words, w/o this word */
} rtcp_common_t;

/*
```

```
 * Big-endian mask for version, padding bit and packet type pair
 */
#define RTCP_VALID_MASK (0xc000 | 0x2000 | 0xfe)
#define RTCP_VALID_VALUE ((RTP_VERSION << 14) | RTCP_SR)

/*
 * Reception report block
 */
typedef struct {
    u_int32 ssrc;            /* data source being reported */
    unsigned int fraction:8; /* fraction lost since last SR/RR */
    int lost:24;             /* cumul. no. pkts lost (signed!) */
    u_int32 last_seq;        /* extended last seq. no. received */
    u_int32 jitter;          /* interarrival jitter */
    u_int32 lsr;             /* last SR packet from this source */
    u_int32 dlsr;            /* delay since last SR packet */
} rtcp_rr_t;

/*
 * SDES item
 */
typedef struct {
    u_int8 type;             /* type of item (rtcp_sdes_type_t) */
    u_int8 length;           /* length of item (in octets) */
    char data[1];            /* text, not null-terminated */
} rtcp_sdes_item_t;

/*
 * One RTCP packet
 */
typedef struct {
    rtcp_common_t common;    /* common header */
    union {
        /* sender report (SR) */
        struct {
            u_int32 ssrc;    /* sender generating this report */
            u_int32 ntp_sec; /* NTP timestamp */
            u_int32 ntp_frac;
            u_int32 rtp_ts;  /* RTP timestamp */
            u_int32 psent;   /* packets sent */
```

```
            u_int32 osent;    /* octets sent */
            rtcp_rr_t rr[1];  /* variable-length list */
        } sr;

        /* reception report (RR) */
        struct {
            u_int32 ssrc;     /* receiver generating this report */
            rtcp_rr_t rr[1];  /* variable-length list */
        } rr;

        /* source description (SDES) */
        struct rtcp_sdes {
            u_int32 src;      /* first SSRC/CSRC */
            rtcp_sdes_item_t item[1]; /* list of SDES items */
        } sdes;

        /* BYE */
        struct {
            u_int32 src[1];   /* list of sources */
            /* can't express trailing text for reason */
        } bye;
    } r;
} rtcp_t;

typedef struct rtcp_sdes rtcp_sdes_t;

/*
 * Per-source state information
 */
typedef struct {
    u_int16 max_seq;        /* highest seq. number seen */
    u_int32 cycles;         /* shifted count of seq. number cycles */
    u_int32 base_seq;       /* base seq number */
    u_int32 bad_seq;        /* last 'bad' seq number + 1 */
    u_int32 probation;      /* sequ. packets till source is valid */
    u_int32 received;       /* packets received */
    u_int32 expected_prior; /* packet expected at last interval */
    u_int32 received_prior; /* packet received at last interval */
    u_int32 transit;        /* relative trans time for prev pkt */
    u_int32 jitter;         /* estimated jitter */
```

```
    /* ... */
} source;
```

A.1 RTP Data Header Validity Checks

An RTP receiver should check the validity of the RTP header on incoming packets since they might be encrypted or might be from a different application that happens to be misaddressed. Similarly, if encryption according to the method described in Section 9 is enabled, the header validity check is needed to verify that incoming packets have been correctly decrypted, although a failure of the header validity check (e.g., unknown payload type) may not necessarily indicate decryption failure.

Only weak validity checks are possible on an RTP data packet from a source that has not been heard before:

- o RTP version field must equal 2.
- o The payload type must be known, and in particular it must not be equal to SR or RR.
- o If the P bit is set, then the last octet of the packet must contain a valid octet count, in particular, less than the total packet length minus the header size.
- o The X bit must be zero if the profile does not specify that the header extension mechanism may be used. Otherwise, the extension length field must be less than the total packet size minus the fixed header length and padding.
- o The length of the packet must be consistent with CC and payload type (if payloads have a known length).

The last three checks are somewhat complex and not always possible, leaving only the first two which total just a few bits. If the SSRC identifier in the packet is one that has been received before, then the packet is probably valid and checking if the sequence number is in the expected range provides further validation. If the SSRC identifier has not been seen before, then data packets carrying that

identifier may be considered invalid until a small number of them arrive with consecutive sequence numbers. Those invalid packets MAY be discarded or they MAY be stored and delivered once validation has been achieved if the resulting delay is acceptable.

The routine update_seq shown below ensures that a source is declared valid only after MIN_SEQUENTIAL packets have been received in sequence. It also validates the sequence number seq of a newly received packet and updates the sequence state for the packet's source in the structure to which s points.

When a new source is heard for the first time, that is, its SSRC identifier is not in the table (see Section 8.2), and the per-source state is allocated for it, s->probation is set to the number of sequential packets required before declaring a source valid (parameter MIN_SEQUENTIAL) and other variables are initialized:

```
init_seq(s, seq);
s->max_seq = seq - 1;
s->probation = MIN_SEQUENTIAL;
```

A non-zero s->probation marks the source as not yet valid so the state may be discarded after a short timeout rather than a long one, as discussed in Section 6.2.1.

After a source is considered valid, the sequence number is considered valid if it is no more than MAX_DROPOUT ahead of s->max_seq nor more than MAX_MISORDER behind. If the new sequence number is ahead of max_seq modulo the RTP sequence number range (16 bits), but is smaller than max_seq, it has wrapped around and the (shifted) count of sequence number cycles is incremented. A value of one is returned to indicate a valid sequence number.

Otherwise, the value zero is returned to indicate that the validation failed, and the bad sequence number plus 1 is stored. If the next packet received carries the next higher sequence number, it is considered the valid start of a new packet sequence presumably caused by an extended dropout or a source restart. Since multiple complete sequence number cycles may have been missed, the packet loss statistics are reset.

Typical values for the parameters are shown, based on a maximum misordering time of 2 seconds at 50 packets/second and a maximum dropout of 1 minute. The dropout parameter MAX_DROPOUT should be a small fraction of the 16-bit sequence number space to give a reasonable probability that new sequence numbers after a restart will not fall in the acceptable range for sequence numbers from before the restart.

```
void init_seq(source *s, u_int16 seq)
{
    s->base_seq = seq;
    s->max_seq = seq;
    s->bad_seq = RTP_SEQ_MOD + 1;   /* so seq == bad_seq is false */
    s->cycles = 0;
    s->received = 0;
    s->received_prior = 0;
    s->expected_prior = 0;
    /* other initialization */
}

int update_seq(source *s, u_int16 seq)
{
    u_int16 udelta = seq - s->max_seq;
    const int MAX_DROPOUT = 3000;
    const int MAX_MISORDER = 100;
    const int MIN_SEQUENTIAL = 2;

    /*
     * Source is not valid until MIN_SEQUENTIAL packets with
     * sequential sequence numbers have been received.
     */
    if (s->probation) {
        /* packet is in sequence */
        if (seq == s->max_seq + 1) {
            s->probation--;
            s->max_seq = seq;
            if (s->probation == 0) {
                init_seq(s, seq);
                s->received++;
```

```
                return 1;
            }
        } else {
            s->probation = MIN_SEQUENTIAL - 1;
            s->max_seq = seq;
        }
        return 0;
    } else if (udelta < MAX_DROPOUT) {
        /* in order, with permissible gap */
        if (seq < s->max_seq) {
            /*
             * Sequence number wrapped - count another 64K cycle.
             */
            s->cycles += RTP_SEQ_MOD;
        }
        s->max_seq = seq;
    } else if (udelta <= RTP_SEQ_MOD - MAX_MISORDER) {
        /* the sequence number made a very large jump */
        if (seq == s->bad_seq) {
            /*
             * Two sequential packets -- assume that the other side
             * restarted without telling us so just re-sync
             * (i.e., pretend this was the first packet).
             */
            init_seq(s, seq);
        }
        else {
            s->bad_seq = (seq + 1) & (RTP_SEQ_MOD-1);
            return 0;
        }
    } else {
        /* duplicate or reordered packet */
    }
    s->received++;
    return 1;
}
```

The validity check can be made stronger requiring more than two packets in sequence. The disadvantages are that a larger number of initial packets will be discarded (or delayed in a queue) and that

high packet loss rates could prevent validation. However, because the RTCP header validation is relatively strong, if an RTCP packet is received from a source before the data packets, the count could be adjusted so that only two packets are required in sequence. If initial data loss for a few seconds can be tolerated, an application MAY choose to discard all data packets from a source until a valid RTCP packet has been received from that source.

Depending on the application and encoding, algorithms may exploit additional knowledge about the payload format for further validation. For payload types where the timestamp increment is the same for all packets, the timestamp values can be predicted from the previous packet received from the same source using the sequence number difference (assuming no change in payload type).

A strong "fast-path" check is possible since with high probability the first four octets in the header of a newly received RTP data packet will be just the same as that of the previous packet from the same SSRC except that the sequence number will have increased by one. Similarly, a single-entry cache may be used for faster SSRC lookups in applications where data is typically received from one source at a time.

A.2 RTCP Header Validity Checks

The following checks should be applied to RTCP packets.

- RTP version field must equal 2.

- The payload type field of the first RTCP packet in a compound packet must be equal to SR or RR.

- The padding bit (P) should be zero for the first packet of a compound RTCP packet because padding should only be applied, if it is needed, to the last packet.

- The length fields of the individual RTCP packets must add up to the overall length of the compound RTCP packet as received. This is a fairly strong check.

The code fragment below performs all of these checks. The packet type is not checked for subsequent packets since unknown packet types may be present and should be ignored.

```
u_int32 len;        /* length of compound RTCP packet in words */
rtcp_t *r;          /* RTCP header */
rtcp_t *end;        /* end of compound RTCP packet */

if ((*(u_int16 *)r & RTCP_VALID_MASK) != RTCP_VALID_VALUE) {
    /* something wrong with packet format */
}
end = (rtcp_t *)((u_int32 *)r + len);

do r = (rtcp_t *)((u_int32 *)r + r->common.length + 1);
while (r < end && r->common.version == 2);

if (r != end) {
    /* something wrong with packet format */
}
```

A.3 Determining Number of Packets Expected and Lost

In order to compute packet loss rates, the number of RTP packets expected and actually received from each source needs to be known, using per-source state information defined in struct source referenced via pointer s in the code below. The number of packets received is simply the count of packets as they arrive, including any late or duplicate packets. The number of packets expected can be computed by the receiver as the difference between the highest sequence number received (s->max_seq) and the first sequence number received (s->base_seq). Since the sequence number is only 16 bits and will wrap around, it is necessary to extend the highest sequence number with the (shifted) count of sequence number wraparounds (s->cycles). Both the received packet count and the count of cycles are maintained the RTP header validity check routine in Appendix A.1.

```
extended_max = s->cycles + s->max_seq;
expected = extended_max - s->base_seq + 1;
```

The number of packets lost is defined to be the number of packets

expected less the number of packets actually received:

```
lost = expected - s->received;
```

Since this signed number is carried in 24 bits, it should be clamped at 0x7fffff for positive loss or 0x800000 for negative loss rather than wrapping around.

The fraction of packets lost during the last reporting interval (since the previous SR or RR packet was sent) is calculated from differences in the expected and received packet counts across the interval, where expected_prior and received_prior are the values saved when the previous reception report was generated:

```
expected_interval = expected - s->expected_prior;
s->expected_prior = expected;
received_interval = s->received - s->received_prior;
s->received_prior = s->received;
lost_interval = expected_interval - received_interval;
if (expected_interval == 0 || lost_interval <= 0) fraction = 0;
else fraction = (lost_interval << 8) / expected_interval;
```

The resulting fraction is an 8-bit fixed point number with the binary point at the left edge.

A.4 Generating RTCP SDES Packets

This function builds one SDES chunk into buffer b composed of argc items supplied in arrays type, value and length. It returns a pointer to the next available location within b.

```
char *rtp_write_sdes(char *b, u_int32 src, int argc,
                     rtcp_sdes_type_t type[], char *value[],
                     int length[])
{
    rtcp_sdes_t *s = (rtcp_sdes_t *)b;
    rtcp_sdes_item_t *rsp;
    int i;
    int len;
    int pad;
```

```
        /* SSRC header */
        s->src = src;
        rsp = &s->item[0];

        /* SDES items */
        for (i = 0; i < argc; i++) {
            rsp->type = type[i];
            len = length[i];
            if (len > RTP_MAX_SDES) {
                /* invalid length, may want to take other action */
                len = RTP_MAX_SDES;
            }
            rsp->length = len;
            memcpy(rsp->data, value[i], len);
            rsp = (rtcp_sdes_item_t *)&rsp->data[len];
        }

        /* terminate with end marker and pad to next 4-octet boundary */
        len = ((char *) rsp) - b;
        pad = 4 - (len & 0x3);
        b = (char *) rsp;
        while (pad--) *b++ = RTCP_SDES_END;

        return b;
    }
```

A.5 Parsing RTCP SDES Packets

This function parses an SDES packet, calling functions find_member() to find a pointer to the information for a session member given the SSRC identifier and member_sdes() to store the new SDES information for that member. This function expects a pointer to the header of the RTCP packet.

```
    void rtp_read_sdes(rtcp_t *r)
    {
        int count = r->common.count;
        rtcp_sdes_t *sd = &r->r.sdes;
        rtcp_sdes_item_t *rsp, *rspn;
```

```
        rtcp_sdes_item_t *end = (rtcp_sdes_item_t *)
                               ((u_int32 *)r + r->common.length + 1);
        source *s;

        while (--count >= 0) {
            rsp = &sd->item[0];
            if (rsp >= end) break;
            s = find_member(sd->src);

            for (; rsp->type; rsp = rspn ) {
                rspn = (rtcp_sdes_item_t *)((char*)rsp+rsp->length+2);
                if (rspn >= end) {
                    rsp = rspn;
                    break;
                }
                member_sdes(s, rsp->type, rsp->data, rsp->length);
            }
            sd = (rtcp_sdes_t *)
                 ((u_int32 *)sd + (((char *)rsp - (char *)sd) >> 2)+1);
        }
        if (count >= 0) {
            /* invalid packet format */
        }
    }
```

A.6 Generating a Random 32-bit Identifier

The following subroutine generates a random 32-bit identifier using the MD5 routines published in RFC 1321 [32]. The system routines may not be present on all operating systems, but they should serve as hints as to what kinds of information may be used. Other system calls that may be appropriate include

- getdomainname(),
- getwd(), or
- getrusage().

"Live" video or audio samples are also a good source of random

numbers, but care must be taken to avoid using a turned-off microphone or blinded camera as a source [17].

Use of this or a similar routine is recommended to generate the initial seed for the random number generator producing the RTCP period (as shown in Appendix A.7), to generate the initial values for the sequence number and timestamp, and to generate SSRC values. Since this routine is likely to be CPU-intensive, its direct use to generate RTCP periods is inappropriate because predictability is not an issue. Note that this routine produces the same result on repeated calls until the value of the system clock changes unless different values are supplied for the type argument.

```
/*
 * Generate a random 32-bit quantity.
 */
#include <sys/types.h>    /* u_long */
#include <sys/time.h>     /* gettimeofday() */
#include <unistd.h>       /* get..() */
#include <stdio.h>        /* printf() */
#include <time.h>         /* clock() */
#include <sys/utsname.h> /* uname() */
#include "global.h"       /* from RFC 1321 */
#include "md5.h"          /* from RFC 1321 */

#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final

static u_long md_32(char *string, int length)
{
    MD_CTX context;
    union {
        char   c[16];
        u_long x[4];
    } digest;
    u_long r;
    int i;
```

```
    MDInit (&context);
    MDUpdate (&context, string, length);
    MDFinal ((unsigned char *)&digest, &context);
    r = 0;
    for (i = 0; i < 3; i++) {
        r ^= digest.x[i];
    }
    return r;
}                                   /* md_32 */

/*
 * Return random unsigned 32-bit quantity.  Use 'type' argument if
 * you need to generate several different values in close succession.
 */
u_int32 random32(int type)
{
    struct {
        int     type;
        struct  timeval tv;
        clock_t cpu;
        pid_t   pid;
        u_long  hid;
        uid_t   uid;
        gid_t   gid;
        struct  utsname name;
    } s;

    gettimeofday(&s.tv, 0);
    uname(&s.name);
    s.type = type;
    s.cpu  = clock();
    s.pid  = getpid();
    s.hid  = gethostid();
    s.uid  = getuid();
    s.gid  = getgid();
    /* also: system uptime */

    return md_32((char *)&s, sizeof(s));
}                                   /* random32 */
```

A.7 Computing the RTCP Transmission Interval

The following functions implement the RTCP transmission and reception rules described in Section 6.2. These rules are coded in several functions:

- rtcp_interval() computes the deterministic calculated interval, measured in seconds. The parameters are defined in Section 6.3.

- OnExpire() is called when the RTCP transmission timer expires.

- OnReceive() is called whenever an RTCP packet is received.

Both OnExpire() and OnReceive() have event e as an argument. This is the next scheduled event for that participant, either an RTCP report or a BYE packet. It is assumed that the following functions are available:

- Schedule(time t, event e) schedules an event e to occur at time t. When time t arrives, the function OnExpire is called with e as an argument.

- Reschedule(time t, event e) reschedules a previously scheduled event e for time t.

- SendRTCPReport(event e) sends an RTCP report.

- SendBYEPacket(event e) sends a BYE packet.

- TypeOfEvent(event e) returns EVENT_BYE if the event being processed is for a BYE packet to be sent, else it returns EVENT_REPORT.

- PacketType(p) returns PACKET_RTCP_REPORT if packet p is an RTCP report (not BYE), PACKET_BYE if its a BYE RTCP packet, and PACKET_RTP if its a regular RTP data packet.

- ReceivedPacketSize() and SentPacketSize() return the size of the referenced packet in octets.

- NewMember(p) returns a 1 if the participant who sent packet p is not currently in the member list, 0 otherwise. Note this function is not sufficient for a complete implementation because each CSRC identifier in an RTP packet and each SSRC in a BYE packet should be processed.

- NewSender(p) returns a 1 if the participant who sent packet p is not currently in the sender sublist of the member list, 0 otherwise.

- AddMember() and RemoveMember() to add and remove participants from the member list.

- AddSender() and RemoveSender() to add and remove participants from the sender sublist of the member list.

These functions would have to be extended for an implementation that allows the RTCP bandwidth fractions for senders and non-senders to be specified as explicit parameters rather than fixed values of 25% and 75%. The extended implementation of rtcp_interval() would need to avoid division by zero if one of the parameters was zero.

```
double rtcp_interval(int members,
                     int senders,
                     double rtcp_bw,
                     int we_sent,
                     double avg_rtcp_size,
                     int initial)
{
    /*
     * Minimum average time between RTCP packets from this site (in
     * seconds).  This time prevents the reports from 'clumping' when
     * sessions are small and the law of large numbers isn't helping
     * to smooth out the traffic.  It also keeps the report interval
     * from becoming ridiculously small during transient outages like
     * a network partition.
     */
    double const RTCP_MIN_TIME = 5.;
    /*
     * Fraction of the RTCP bandwidth to be shared among active
```

```
 * senders.  (This fraction was chosen so that in a typical
 * session with one or two active senders, the computed report
 * time would be roughly equal to the minimum report time so that
 * we don't unnecessarily slow down receiver reports.)  The
 * receiver fraction must be 1 - the sender fraction.
 */
double const RTCP_SENDER_BW_FRACTION = 0.25;
double const RTCP_RCVR_BW_FRACTION = (1-RTCP_SENDER_BW_FRACTION);
/*
/* To compensate for "timer reconsideration" converging to a
 * value below the intended average.
 */
double const COMPENSATION = 2.71828 - 1.5;

double t;                       /* interval */
double rtcp_min_time = RTCP_MIN_TIME;
int n;                          /* no. of members for computation */

/*
 * Very first call at application start-up uses half the min
 * delay for quicker notification while still allowing some time
 * before reporting for randomization and to learn about other
 * sources so the report interval will converge to the correct
 * interval more quickly.
 */
if (initial) {
    rtcp_min_time /= 2;
}
/*
 * Dedicate a fraction of the RTCP bandwidth to senders unless
 * the number of senders is large enough that their share is
 * more than that fraction.
 */
n = members;
if (senders <= members * RTCP_SENDER_BW_FRACTION) {
    if (we_sent) {
        rtcp_bw *= RTCP_SENDER_BW_FRACTION;
        n = senders;
    } else {
        rtcp_bw *= RTCP_RCVR_BW_FRACTION;
```

```
            n -= senders;
        }
    }

    /*
     * The effective number of sites times the average packet size is
     * the total number of octets sent when each site sends a report.
     * Dividing this by the effective bandwidth gives the time
     * interval over which those packets must be sent in order to
     * meet the bandwidth target, with a minimum enforced.  In that
     * time interval we send one report so this time is also our
     * average time between reports.
     */
    t = avg_rtcp_size * n / rtcp_bw;
    if (t < rtcp_min_time) t = rtcp_min_time;

    /*
     * To avoid traffic bursts from unintended synchronization with
     * other sites, we then pick our actual next report interval as a
     * random number uniformly distributed between 0.5*t and 1.5*t.
     */
    t = t * (drand48() + 0.5);
    t = t / COMPENSATION;
    return t;
}

void OnExpire(event e,
              int    members,
              int    senders,
              double rtcp_bw,
              int    we_sent,
              double *avg_rtcp_size,
              int    *initial,
              time_tp   tc,
              time_tp   *tp,
              int    *pmembers)
{
    /* This function is responsible for deciding whether to send an
     * RTCP report or BYE packet now, or to reschedule transmission.
     * It is also responsible for updating the pmembers, initial, tp,
```

```
 * and avg_rtcp_size state variables.  This function should be
 * called upon expiration of the event timer used by Schedule().
 */

double t;     /* Interval */
double tn;    /* Next transmit time */

/* In the case of a BYE, we use "timer reconsideration" to
 * reschedule the transmission of the BYE if necessary */

if (TypeOfEvent(e) == EVENT_BYE) {
    t = rtcp_interval(members,
                      senders,
                      rtcp_bw,
                      we_sent,
                      *avg_rtcp_size,
                      *initial);
    tn = *tp + t;
    if (tn <= tc) {
        SendBYEPacket(e);
        exit(1);
    } else {
        Schedule(tn, e);
    }

} else if (TypeOfEvent(e) == EVENT_REPORT) {
    t = rtcp_interval(members,
                      senders,
                      rtcp_bw,
                      we_sent,
                      *avg_rtcp_size,
                      *initial);
    tn = *tp + t;
    if (tn <= tc) {
        SendRTCPReport(e);
        *avg_rtcp_size = (1./16.)*SentPacketSize(e) +
            (15./16.)*(*avg_rtcp_size);
        *tp = tc;

        /* We must redraw the interval.  Don't reuse the
```

```
                one computed above, since its not actually
                distributed the same, as we are conditioned
                on it being small enough to cause a packet to
                be sent */

            t = rtcp_interval(members,
                              senders,
                              rtcp_bw,
                              we_sent,
                              *avg_rtcp_size,
                              *initial);

            Schedule(t+tc,e);
            *initial = 0;
        } else {
            Schedule(tn, e);
        }
        *pmembers = members;
    }
}

void OnReceive(packet p,
               event e,
               int *members,
               int *pmembers,
               int *senders,
               double *avg_rtcp_size,
               double *tp,
               double tc,
               double tn)
{
    /* What we do depends on whether we have left the group, and are
     * waiting to send a BYE (TypeOfEvent(e) == EVENT_BYE) or an RTCP
     * report.  p represents the packet that was just received.  */

    if (PacketType(p) == PACKET_RTCP_REPORT) {
        if (NewMember(p) && (TypeOfEvent(e) == EVENT_REPORT)) {
            AddMember(p);
            *members += 1;
        }
```

```
        *avg_rtcp_size = (1./16.)*ReceivedPacketSize(p) +
            (15./16.)*(*avg_rtcp_size);
    } else if (PacketType(p) == PACKET_RTP) {
        if (NewMember(p) && (TypeOfEvent(e) == EVENT_REPORT)) {
            AddMember(p);
            *members += 1;
        }
        if (NewSender(p) && (TypeOfEvent(e) == EVENT_REPORT)) {
            AddSender(p);
            *senders += 1;
        }
    } else if (PacketType(p) == PACKET_BYE) {
        *avg_rtcp_size = (1./16.)*ReceivedPacketSize(p) +
            (15./16.)*(*avg_rtcp_size);

        if (TypeOfEvent(e) == EVENT_REPORT) {
            if (NewSender(p) == FALSE) {
                RemoveSender(p);
                *senders -= 1;
            }

            if (NewMember(p) == FALSE) {
                RemoveMember(p);
                *members -= 1;
            }

            if (*members < *pmembers) {
                tn = tc +
                    (((double) *members)/(*pmembers))*(tn - tc);
                *tp = tc -
                    (((double) *members)/(*pmembers))*(tc - *tp);

                /* Reschedule the next report for time tn */

                Reschedule(tn, e);
                *pmembers = *members;
            }

        } else if (TypeOfEvent(e) == EVENT_BYE) {
            *members += 1;
```

```
        }
    }
}
```

A.8 Estimating the Interarrival Jitter

The code fragments below implement the algorithm given in Section 6.4.1 for calculating an estimate of the statistical variance of the RTP data interarrival time to be inserted in the interarrival jitter field of reception reports. The inputs are r->ts, the timestamp from the incoming packet, and arrival, the current time in the same units. Here s points to state for the source; s->transit holds the relative transit time for the previous packet, and s->jitter holds the estimated jitter. The jitter field of the reception report is measured in timestamp units and expressed as an unsigned integer, but the jitter estimate is kept in a floating point. As each data packet arrives, the jitter estimate is updated:

```
int transit = arrival - r->ts;
int d = transit - s->transit;
s->transit = transit;
if (d < 0) d = -d;
s->jitter += (1./16.) * ((double)d - s->jitter);
```

When a reception report block (to which rr points) is generated for this member, the current jitter estimate is returned:

```
rr->jitter = (u_int32) s->jitter;
```

Alternatively, the jitter estimate can be kept as an integer, but scaled to reduce round-off error. The calculation is the same except for the last line:

```
s->jitter += d - ((s->jitter + 8) >> 4);
```

In this case, the estimate is sampled for the reception report as:

```
rr->jitter = s->jitter >> 4;
```

C.6 RTP.H

The header file from the author's RTP implementation. Function protoypes are in the following section.

```
/* new-phone01.h */
/* Shaun Kaplan */
/* Includes code (adapted) from RTP specification */

//#define TRUE  1
//#define FALSE 0
//#define NULL  0

typedef unsigned char  u_int8;
typedef          char  int8;
typedef unsigned short u_int16;
typedef          short int16;
typedef unsigned long  u_int32;
typedef signed   long  int32;

/* Task IDs and priorities */
#define CW_ID      12
#define CW_PRI     12
#define CR_ID      13
#define CR_PRI     13

#define RTP_TX_ID   14
#define RTP_TX_PRI  14
#define RTP_RX_ID   15
#define RTP_RX_PRI  15
#define RTP_ID      18
#define RTP_PRI     18

#define RTCP_TX_ID  16
#define RTCP_TX_PRI 16
#define RTCP_RX_ID  17
#define RTCP_RX_PRI 17

/* Transmit and receive buffer sizes */
#define TBMAX 140
```

```
#define RBMAX 100

/*
 * Random number generator constants
 * from: Sedgewick, R.  Algorithms in C.  pp 511-514
 * adapted for 16bit machine
 */
#define M 10000
#define m1 100
#define b 5821

/* Address of time2count register */
#define T2CNT       0xFF60
/* Probation period */
#define PROBATION   2
/* Length of G.729 frame */
#define G729LENGTH  10
/* Timeout period */
#define TIMEOUT     5
/* Maximum size of RTCP packet */
#define RTCP_MAX    95
/* Maximum size of RTCP packet with bye */
#define RTCPBYE_MAX 103
/* Maximum size of sender report */
#define SR_MAX      52
/* Maximum size of SDES packet */
#define SDES_MAX    43
/* Maximum size of report block */
#define RB_MAX      24
/* Real-time clock register */
#define ZONEADDR    0

/*
 * Following 2 macros for endian conversion are from the Spread Toolkit
 * (www.spread.org)
 */
#define Flip_int16(type)  (((type >> 8) & 0x00ff) |
                           ((type << 8) & 0xff00))
#define Flip_int32(type)  (((type >>24) & 0x000000ff) |
                           ((type >> 8) & 0x0000ff00) |
```

```
                        ((type << 8) & 0x00ff0000) |
                        ((type <<24) & 0xff000000) )

#define RTP_VERSION    2
/* maximum text length for SDES */
#define RTP_MAX_SDES 255

/* RTCP types */
#define RTCP_SR   200
#define RTCP_RR   201
#define RTCP_SDES 202
#define RTCP_BYE  203
#define RTCP_APP  204

/* SDES item types */
#define RTCP_SDES_END   0
#define RTCP_SDES_CNAME 1
#define RTCP_SDES_NAME  2
#define RTCP_SDES_EMAIL 3
#define RTCP_SDES_PHONE 4
#define RTCP_SDES_LOC   5
#define RTCP_SDES_TOOL  6
#define RTCP_SDES_NOTE  7
#define RTCP_SDES_PRIV  8

typedef enum
{
  ENCODE,
  DECODE
}codec_operation;

/* RTP modes of operation for testing purposes */
typedef enum
{
  RTP_TX,
  RTCP_TX,
  RTP_RTCP_TX,
  RTP_RX,
  RTCP_RX,
  RTP_RTCP_RX,
```

```
  H_DUPLEX_SEND,    /* send rtp, send and rx rtcp */
  H_DUPLEX_RX,      /* rx rtp, send and rx rtcp */
  FULL_DUPLEX,
  FULL_DUPLEX_TEST /* send and rx rtp, no rtcp */
}rtpmode_type;

/* RTP session events */
typedef enum
{
  START_SESSION,
  END_SESSION,
  SSRC_COLLISION,
  ADJUSTMENT
}event_type;

/* Structure for information from the user inerface */
typedef struct
{
  u_int16      rtp_port_l;   /* local RTP port */
  u_int16      rtp_port_r;   /* remote RTP port */
  u_int8       rx_pt;        /* receiving payload type */
  u_int8       tx_pt;        /* transmitting payload type */
  event_type   event;        /* RTP events */
  IPADDR       rtp_dest_ip;  /* destination IP address */
  u_int8       flag;         /* if flag 0, read event from this struct */
  u_int8  stream[9];         /* stream type - sendrecv, etc. */
  rtpmode_type rtpmode;      /* mode of operation */
}interface_t;

/* RTCP events */
typedef enum
{
  FIRST,
  NORMAL,
  LAST
}rtcp_event_type;

/* Data related to received packets (remote data)*/
typedef struct
{
```

```
  u_int8  pt;              /* payload type */
  u_int32 last_report_seq; /* sequence number of last received RTCP */
  u_int32 frame_count;     /* no. of media frames received */
  u_int32 seq_cycles;      /* no. of sequence no. cycles << 16 */
  u_int16 base_seq;        /* first received sequence number */
  u_int16 last_seq;        /* most recently received sequence number */
  u_int16 bad_seq;
  u_int32 received;        /* no. of packets received */
  u_int32 expected_prior;
  u_int32 received_prior;
  u_int8  probation;       /* remaiing probation period */
  u_int32 rtcp_received;
  u_int8  timeout;         /* if == 0 then ssrc timed out */
  int8    cname[33];       /* max 32 char + terminating NULL */
  u_int8  cname_length;    /* including terminating NULL */
//  u_int8  initial;        /* flag indicating if we've received an RTCP
//                           packet yet */
  u_int32 lsr;             /* last sr timestamp */
  u_int32 ntp;             /* middle 32bits of 'ntp' timestamp */
  u_int32 SR_time;         /* SR time in rtp timestamp units */
  u_int32 jitter;          /* Jitter value */
  int32   transit;         /* trasnit value for jitter calculation */
  int32   tsdiff;     /* tsdiff is the difference between the ts of the
  first received packet and our ts at the time the packet was received
  less 20ms.  This value will allow us to calculate jitter using common
  units. */
}rx_t;

/* Data related to transmitted packets (local data)*/
typedef struct
{
  u_int8  pt;              /* payload type */
  u_int16 seq;             /* seq no. of last packet sent */
  u_int32 seq_cycles;      /* no. of times sequence no. has cycled << 16 */
  u_int32 base_seq;        /* first seq number */
  u_int32 last_report_seq;      /* previous sent RTCP's sequence no. */
  u_int32 two_reports_back_seq; /* sequence no from sent RTCP, two RTCP
                                   packet ago */
  u_int32 frame_count;     /* no. of media frames sent */
  int8    cname[33];       /* max 32 char + terminating NULL */
```

```
  u_int8  cname_length;  /* including terminating NULL */
  volatile u_int32 ts;   /* current timestamp value */
  u_int32 base_ts;       /* original timestamp value */
}tx_t;

/* RTP session data structure */
typedef struct
{
  event_type  event;       /* RTP session state */
  u_int8  end_task;        /* value of TRUE ends tasks */
  u_int16 rtp_port_l;      /* local rtp port */
  u_int16 rtp_port_r;      /* remote rtp port */
  int16   rtp_id;          /* rtp udp id */
  int16   rtcp_id;         /* rtcp udp id */
  IPADDR  rtp_dest_ip;     /* destination ip address for rtp (and rtcp) */
  u_int32 local_ssrc;      /* local SSRC identifier */
  u_int32 remote_ssrc;     /* remote SSRC identifier */
  rx_t    rx;
  tx_t    tx;
  rtcp_event_type  rtcp_event;  /* RTCP event */
}RTP_session_t;

typedef enum
{
  HOURS,
  MINUTES,
  SECONDS,
  DAY,
  MONTH,
  YEAR,
  ZONE
}time_types;

/* RTP data header */
typedef struct
{
   unsigned int cc:4;          /* CSRC count */
   unsigned int x:1;           /* header extension flag */
   unsigned int p:1;           /* padding flag */
   unsigned int version:2;     /* protocol version */
```

```
   unsigned int pt:7;          /* payload type */
   unsigned int m:1;           /* marker bit */
   u_int16 seq;                /* sequence number */
   u_int32 ts;                 /* timestamp */
   u_int32 ssrc;               /* synchronization source */
//   u_int32 csrc[1];            /* optional CSRC list */
} rtp_hdr_t;

/* RTCP common header word */
typedef struct
{
   unsigned int count:5;       /* varies by packet type */
   unsigned int p:1;           /* padding flag */
   unsigned int version:2;     /* protocol version */
   unsigned int pt:8;          /* RTCP packet type */
   u_int16 length;             /* pkt len in words, w/o this word */
} rtcp_common_t;

/* sender report(SR) header and sender info */
typedef struct
{
    u_int32 sender_ssrc;       /* sender generating this report */
    u_int32 ntp_sec;           /* NTP timestamp */
    u_int32 ntp_frac;
    u_int32 rtp_ts;            /* RTP timestamp */
    u_int32 psent;             /* packets sent */
    u_int32 osent;             /* octets sent */
} rtcp_sr_t;

/* Reception report block */
typedef struct
{
   u_int32 ssrc;               /* data source being reported */
   u_int8 fraction;            /* fraction lost since last SR/RR */
   u_int8 lost_up;             /* cumul. no. pkts lost (signed!) */
   unsigned int lost_lo;
   u_int32 last_seq;           /* extended last seq. no. received */
   u_int32 jitter;             /* interarrival jitter */
   u_int32 lsr;                /* last SR packet from this source */
   u_int32 dlsr;               /* delay since last SR packet */
```

```
} rtcp_rr_t;

/* SDES item */
typedef struct
{
  u_int8 type;              /* type of item (rtcp_sdes_type_t) */
  u_int8 length;            /* length of item (in octets) */
  int8 data[33];            /* text, not null-terminated */
}sdes_item_t;

/* RTCP structures */
/* Basic sender report : Common header + SR */
typedef struct
{
  rtcp_common_t common;
  rtcp_sr_t sr;
}rtcp_sr_0_t;

/* Basic receiver report : Common header + SSRC */
typedef struct
{
  rtcp_common_t common;
  u_int32 sender_ssrc;
}rtcp_rr_0_t;
/*---*/

/* BYE */
typedef struct
{
  rtcp_common_t common;
  u_int32 ssrc;   /* list of sources */
  /* can't express trailing text for reason */
}rtcp_bye_t;
```

C.7 RTP function prototypes

```
/*************************************************************************
* Function : to generate a 16 bit initial sequence number
* Output : (u_int32) random initial sequence number
```

```
*****************************************************************************/
u_int16  generate_seq_number(void);

/*****************************************************************************
* Function : to generate a 32 bit initial timestamp value
* Output : (u_int32) random initial timestamp value
*****************************************************************************/
u_int32  generate_timestamp(void);

/*****************************************************************************
* Function : to generate a 32 bit SSRC ID
* Output : (u_int32) random, SSRC value
*****************************************************************************/
u_int32  generate_ssrc(void);

/*****************************************************************************
* Function : Generate cname from local IP address
* Output : length of cname (in bytes including terminating null character)
*****************************************************************************/
u_int8   generate_cname(void);

/*****************************************************************************
* Function : to build a compound rtcp packet (ex. BYE)
* Input  : pointer to rtcp_compound_packet
* Output : length of rtcp compound packet (and packet)
*****************************************************************************/
u_int8   build_compound_rtcp(u_int8 far *rtcp_pac,
                             RTP_session_t far *pRTCPmsg);

/*****************************************************************************
* Function : Build SR packet
* Input  : Pointer to sender report packet
* Output : Length of packet
*****************************************************************************/
u_int8   build_SR(u_int8 far *s_r_p, RTP_session_t far *SRdata);

/*****************************************************************************
* Function : Build RR packet
* Input  : Pointer to receiver report packet
* Output : Length of RR packet
```

```
*****************************************************************************/
u_int8   build_RR(u_int8 far *s_r_p, RTP_session_t far *RRdata);

/*****************************************************************************
* Function : Build report block
* Input  : Pointer to report_block
* Output : Length of report block
*****************************************************************************/
u_int8   build_report_block(u_int8 far *r_b, RTP_session_t far *RBdata);

/*****************************************************************************
* Function : Build SDES (header + 1 chunk (CNAME))
* Input  : pointer to sdes_packet
* Output : sdes packet_length (returns 0 if error)
*****************************************************************************/
u_int8   build_SDES(u_int8 far *s_p, RTP_session_t far *SDESdata);

/*****************************************************************************
* Function : Build Bye packet (without a reason)
* Input  : Pointer to bye packet
* Output : Length of bye packet
*****************************************************************************/
u_int8   build_bye(u_int8 far *r_p, u_int8 p_length,
                   RTP_session_t far *pRTCPmsg);

/*****************************************************************************
* Function : Check sequence numbers to determine whether a source is valid
* Input  : sequence number of current received rtp packet
* Output : u_int8 packet valid/invalid
*****************************************************************************/
u_int8   sequence_check(u_int16 seq, RTP_session_t far *pRTPmsg);

/*****************************************************************************
* Function : Convert elapsed time to NTP time format
* Output  :  elapsed time in seconds and milliseconds
*****************************************************************************/
void     ntp_time(u_int32 *p_sec, u_int32 *p_ms, u_int32 current_ts,
                  u_int32 base_ts);

/*****************************************************************************
```

```
* Function : rtcp validity test
* Input  : compound rtcp packet as u_int8 array, length of data
* Output : TRUE or FALSE code
*****************************************************************************/
u_int16  rtcp_validify(u_int8 *buf, u_int16 length);

/*****************************************************************************
* Function :  parse RTCP compound packets
* Input  :  buffer of received data, length of received data
* Output :  u_int16 return code
* Also sets rx.cname and rx.cname_length
*****************************************************************************/
u_int16  compound_parser(u_int8 buf[], u_int16 buf_length,
                         RTP_session_t far *pRTCPmsg);


/*****************************************************************************
* Function : Set up RTP session
* Output : Code indicating if setup successful or not
*****************************************************************************/
u_int8   rtp_start_session(interface_t far *pImsg);

/*****************************************************************************
* Function : End RTP session
* Output : Code indicating if successful or not
*****************************************************************************/
u_int8   rtp_end_session(rtpmode_type mode);

/*****************************************************************************
* Function : initialize new source
* Input  : void
* Output : various rx. variables set to initial values
*****************************************************************************/
void     initialize_new_source(RTP_session_t far *pRTPdata);

/*  FIFO buffer functions*/
/*****************************************************************************
* Function :  to initialize a FIFO buffer
* Input  :  code signifying which buffer is to be accessed
*****************************************************************************/
```

```
void     buffer_init(u_int8 b2);

/*************************************************************************
* Function :  to put data into a FIFO buffer
* Input    :  u_int8 data, code signifying which buffer is to be accessed
*************************************************************************/
void far put(u_int8 v, u_int8 b0);

/*************************************************************************
* Function :  to get data from a FIFO buffer
* Input    :  code signifying which buffer is to be accessed
* Output   :  u_int8 data from buffer
*************************************************************************/
u_int8   get(u_int8 b1);

/* I2C and codec related functions */
/*************************************************************************
* Function : To setup codec for use
* Input    : codec operation
* Output   : Code indicating if setup successful or not
*************************************************************************/
u_int8   codec_setup(codec_operation code);
void     codec_write(void);
u_int8   i2c_encode(void);
u_int8   i2c_decode(void);
void     start(void);
void     stop(void);
void     ack(void);
void     nack(void);
void     sendbyte(u_int8 dbyte);
u_int8   receivebyte(void);
void     outhigh(void);
void     outlow(void);
void     wait_ack(void);
u_int8   getbit(void);

/* Miscellaneous functions */
void     display_title(void);
u_int8   main_menu(void);
u_int8   rtp_menu(void);
```

```
void     timeset(void);
u_int8   change_time(u_int8 value, time_types type);
u_int8   getseed(void);
int16    randomint(int16 r);
int16    mult(int16 p, int16 q);
```