Voice Communication Across the Internet: A Network Voice Terminal[*]

Henning Schulzrinne
Department of Electrical and Computer Engineering
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
hgschulz@cs.umass.edu

July 29, 1992

Abstract

Voice conferencing has attracted interest as a useful and viable first real-time application on the Internet. This report describes NEVOT a network voice terminal meant to support multiple concurrent both two-party and multi-party conferences on top of a variety of transport protocols and using audio encodings offering from vocoder to multi-channel CD quality. As it is to be used as an experimental tool, it offers extensive configuration, trace and statistics options. The design is kept modular so that additional audio encodings, transport and real-time protocols as well as user interfaces can be added readily. In the first part, the report describes the X-based graphical user interface, the configuration and operation. The second part describes the individual components of NEVOT and compares alternate implementations. An appendix covers the installation of NEVOT.

1 Introduction

Increased bandwidth and computational resources have made interactive voice and video communication between workstations across packet communication facilities feasible. Cooperative work, teleconferencing [1] and simple one-to-one "videotelephones" [2, 3] are applications that have attracted a large amount of implementation and research interest.

Transmitting voice and video across a packet-switched network offers a number of advantages other the circuit-switched approach. First, we obtain all the well-known benefits of service integration, particularly important in a multi-media setting. Secondly, we may be able to achieve a higher bandwidth utilization since voice and video do not always use their peak bandwidth (due to silence periods and variable rate coding). Finally, because interleaving several associations tends to be easier in a packet-switched network, control (signaling, to use the telephony term) can be more sophisticated[1].

Research in transmitting voice across a packet network dates back to the early ARPAnet days. Cohen [4] refers to cross-continental packet voice experiments in 1974. According to [5], low-bit

[*]This work is supported in part by the Office of Naval Research under contract N00014-90-J-1293, the Defense Advanced Research Projects Agency under contract NAG2-578 and a National Science Foundation equipment grant, CERDCR 8500332.

[1]Even narrowband ISDN uses the packet-switched D channel for signaling.

rate voice conferences very carried out in 1976. The early '80s saw experiments of transmitting low-bitrate voice across mobile radio [6, 7] and satellite [8] packet channels. The first Internet packet voice protocol was specified formally in 1977 [9], and a packet video standard followed in 1981 [10]. The CCITT standard G.PVNP [11] was published in 1989. Packet audio/video should be set apart from the approach to voice/data integration that provides fixed-bandwidth circuits on multiple access networks [12, 13].

Interest in packet audio has increased recently as more and more workstations now come equipped with built-in toll-quality (Sun SPARCstations, DEC workstations) or CD-quality (NeXt) audio hardware support. There exist a fair number of simple programs that utilize the SPARCstation audio hardware to communicate between two workstation on a local net, for example vtalk (Miron Cuperman, OKI) or PhoneTalk (Patrik Nises and Joakim Wettby, Royal Institute of Technology, Stockholm). Programs designed for multiple-party connections across wide-area networks include VT [1] and `vat` (Van Jacobsen and Steve McCanne, LBL). A number of commercial products use medium-bitrate packet voice to more effectively utilize leased private lines, extending the concept of the traditional data-only multiplexer [14]. System implementations of packet voice terminals are described in [5, 18, 19]. Packet radio experiments are featured in [20]. Surveys on packet voice performance are presented in [18].

Numerous other voice/data integration schemes have been studied, usually combining a circuit-switched path for voice and a packet-switched path for data, possibly with bandwidth traded between the two. Examples include [21]. Economic studies comparing alternative network strategies were performed by Gitman and Frank [22].

This report describes NEVOT and is divided into three major parts. The first part, Section 2, describes the facilities of NEVOT and how to use them, principally through the graphical user interface. The second part then delves into the internals, laying out the methods used and comparing some implementation choices. Finally, an appendix provides some hints on installing NEVOT.

2 The Network Voice Terminal (NEVOT) – User's Guide

NEVOT ("NEtwork VOice Terminal") is a tool to support audio conferences across local and wide area networks, including the Internet. It supports multiple simultaneous conferences and a variety of standard and experimental network protocols, including ST-II [23], IP multicast [24, 25, 26] [27, p. 281f] and TCP. It is meant to serve several purposes:

- as a demonstration tool for Internet audio conferences,
- as a measurement tool to investigate traffic patterns and losses in packet voice applications across wide-area networks,
- as a demonstration implementation of real-time services in a distinctly non-real-time operating system (Unix)
- as a traffic source to validate and evaluate resource allocation protocols and algorithms
- as a platform for implementing conference control mechanisms

Extensive tracing and parameterization facilities as well as a modular architecture support experiments in packet voice. The major features are summarized below.

2.1 Features of Version 0.95

Features anticipated for versions released shortly are also listed, but so indicated. Due to operating system or hardware support, a few features are platform-specific. A symbol is used to mark the corresponding platform.

- platforms:
  - Sun SPARCstation¶
  - Silicon Graphics 4D/30 and 4D/35 (Indigo)§
  - Personal DECstation†[in preparation]
- audio protocols:
  - NVP-II (network voice protocol) as used by `vat` (Lawrence Berkeley Laboratory) and vt (ISI)
  - `vat` audio packet format
- transport protocols:
  - unicast UDP
  - multicast UDP
  - TCP
  - ST-II¶
- operation as gateway or end system
- compatible with `vat` session protocol
- user interfaces:
  - XView (OpenLook)
  - Motif GUI
  - curses (for terminals with cursor positioning)
  - dumb terminal
- control:
  - initialization file
  - command line arguments
  - interactive
- several independent concurrent conferences, each with different encoding and compression
- DES-based voice encryption
- current audio encodings supported:
  - 16 bit linear encoding, with all hardware-supported sample rates§
  - 64 kb/s G.711 μ-law PCM

  - 32 kb/s G.721 ADPCM¶
  - 32 kb/s Intel/DVI ADPCM
  - 24 kb/s G.723 ADPCM¶
  - 4.8 kb/s LPC (linear-predictive coding) with setable vocoder interval

- dynamic change in audio encoding, with each site having different encodings (but the same sample rate)
- one or multiple audio channels (i.e., mono or stereo)
- playback and recording of audio files (.au and AIFF/AIFC formats), with encoding translation
- extensive statistics and tracing facilities
- arbitrary voice packet length, which may differ for each site
- lost packet substitution
- setable audio buffer occupancy
- configurable adjustment mechanisms for playout delay, VU meter, silence detector and automatic gain control
- redefinable session identifier string with variable substitution

Most commonly, NEVOT interacts with the user through the Open Look^{TM} or Motif^{TM} graphical user interface on X11-capable workstations. Another version with identical functionality, but a more limited user interface, requires only cursor-addressable ASCII terminals supported by the `curses` library. A fourth version is meant mostly for remote use and uses only terse sequential terminal output to stdio. The command interface used to control the text versions is also available for the XView and Motif versions.

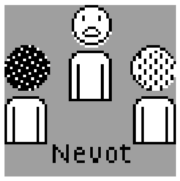


Figure 1: The NEVOT icon

NEVOT may utilize the network services of unicast UDP, multicast UDP, TCP and ST-II [28, 23]. The *source address option* allows operation behind a UDP-level packet reflector, e.g., the simple version written by the author. The packet reflector is used to allow a site running kernels without multicast support to participate in multicast audio conferences. The packet reflector is executed on a multicast-capable site, declares itself part of the multicast group and simply forwards every packet arriving from the multicast group on a unicast UDP socket. The address where the voice packet originated from (i.e., the source address) is prepended by the packet reflector as the first four user data bytes of the packet. This is necessary for proper operation of the voice terminal, since the IP source address of the UDP packet reaching the final destination contains the IP address of the packet reflector rather than that of the speaker, while the actual source address is needed to distinguish several audio streams coming from the same reflector.

NEVOT can maintain several concurrent conferences. The user can participate in all conferences simultaneously, but the conferences remain separate for the other participants. A conference with participation from remote sites could be set-up using the following scenario[2]. Conference audio from the meeting room is distributed via multicast to all sites, who are only listening to that conference. Each remote site also maintains a second "call-in" conference which it uses to pose questions through a moderator at the conference location. Before a remote participant raises a question, he or she listens to the "call-in" channel to reduce conflicts.

NEVOT can act as an application-level gateway between different conferences. Each conference can employ different audio encodings and transport protocols. The audio from all sites within a stream is mixed and distributed to streams, except the originating one.[3]

NEVOT can play and record sound files in AIFF/AIFC and Sun/DEC `.au` format. Recording may be useful to document speech quality or to take notes of important announcements. The usefulness is somewhat limited by the disk space requirement of 480,000 bytes per minute. Optional voice compression independent of the packet audio compression is planned. It is also planned to extend the drag-and-drop mechanism of the windowing system to playing sound files.

The distribution and reception of audio can be controlled at different levels. Check boxes at the top of the base window mute the microphone or speaker, effective for all conferences. Similarly, check boxes within each conference control area control talking or listening to the participants of that conference only. Finally, the site control area has a button that toggles listening to that site. If a site is talking, a one eighth note symbol appears in the listen button, highlighting the current speaker. A muted site is indicated by a crossed-out listen button, as shown in the figure. Note that the muting can also be controlled by keyboard control (see section 2.5); in particular, the microphone mute is toggled by the space bar.

The long propagation and queueing delays in packet networks combined with the use of loudspeakers and omnidirectional microphones may create severe acoustical echo problems. The echo suppressor check box enables a simple echo suppression mechanism that mutes the speaker when sound is received through the local microphone. While reducing echo effects, echo suppression may also cause speech break-ups during local noise peaks.

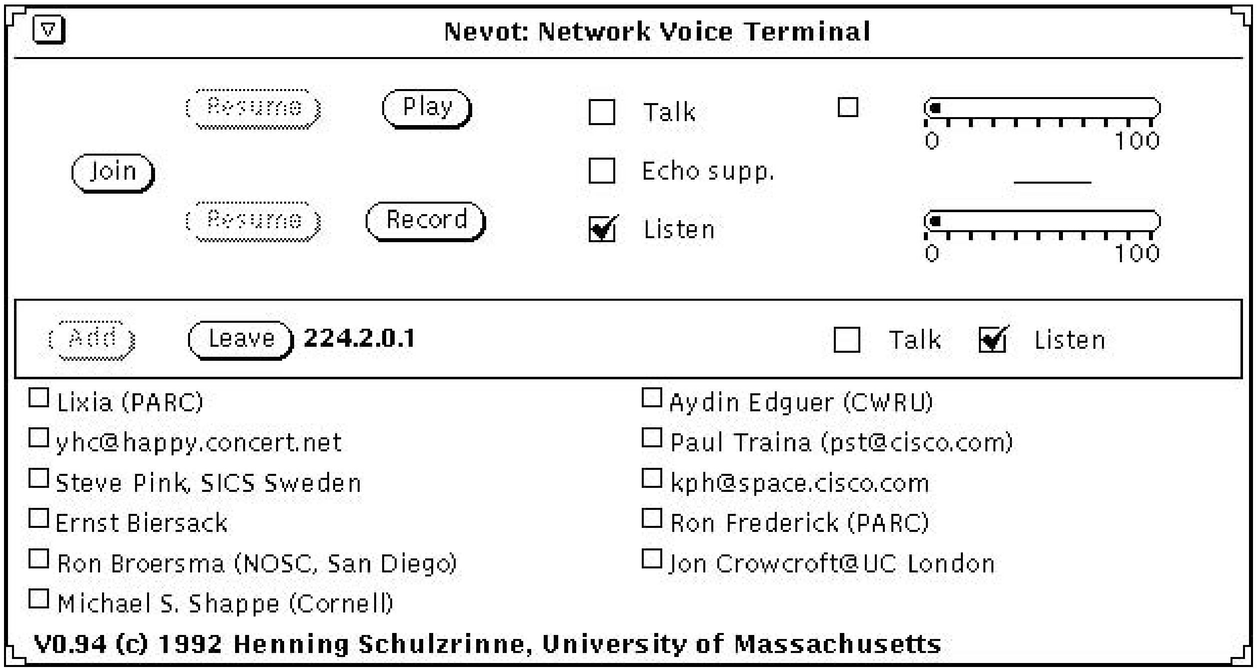
2.2 Setting up a Conference

To set up a new conference (stream), press the join button. The panel shown in Fig. 3 requests the necessary information for the conference: The **host** field contains a list of hosts and their source routing, if necessary, as described in the next paragraph. Instead of specifying hosts directly, `@file` retrieves the information from a text file. Nesting is currently not supported. Hosts can be specified in either dotted decimal notation or as a name. Any name that can be resolved through the network information services (NIS, aka YP), the resolver (`bind`), the `/etc/hosts` file or other locally available means is permissible. Multicast addresses can be added to `/etc/hosts` file or its YP distributed version. Multicast addresses are automatically recognized and thus require no further identification. Multicast addresses are currently only supported using UDP transport. To wait for connections initiated by other sites, leave the field blank.

The host field contains a white-space separated list structured by the (case-sensitive) keywords `TARGET`, `STRICT_ST` (strict ST source route), `LOOSE_ST` (loose ST source route), `STRICT_IP` (strict source route with IP encapsulation), `LOOSE_IP` (loose source route with IP encapsulation) and `PORT`.

[2]This is actually being planned for the Internet engineering task force (IETF) working group sessions.

[3]It is planned to allow gateway operation even on machines without audio support so that a superminicomputer could serve as a transcoder for high-quality low-bitrate audio conferences, once a mechanism for predictable task scheduling on 20 ms intervals can be determined.



Nevot: Network Voice Terminal

☐ Talk
☐ Echo supp.
✓ Listen

224.2.0.1 ☐ Talk ✓ Listen

☐ Lixia (PARC)
☐ yhc@happy.concert.net
☐ Steve Pink, SICS Sweden
☐ Ernst Biersack
☐ Ron Broersma (NOSC, San Diego)
☐ Michael S. Shappe (Cornell)
☐ Aydin Edguer (CWRU)
☐ Paul Traina (pst@cisco.com)
☐ kph@space.cisco.com
☐ Ron Frederick (PARC)
☐ Jon Crowcroft@UC London

V0.94 (c) 1992 Henning Schulzrinne, University of Massachusetts

Figure 2: Nevot display when connected other sites

TARGET is followed by a single host name, while the routing options are followed by lists of zero or more hosts. The current port number, valid until the next occurrence of the PORT specifier, is given by a positive integer following PORT. Currently, only the ST-II API understands about source routes; other transport protocols simple ignore these specifications. As an example, consider:

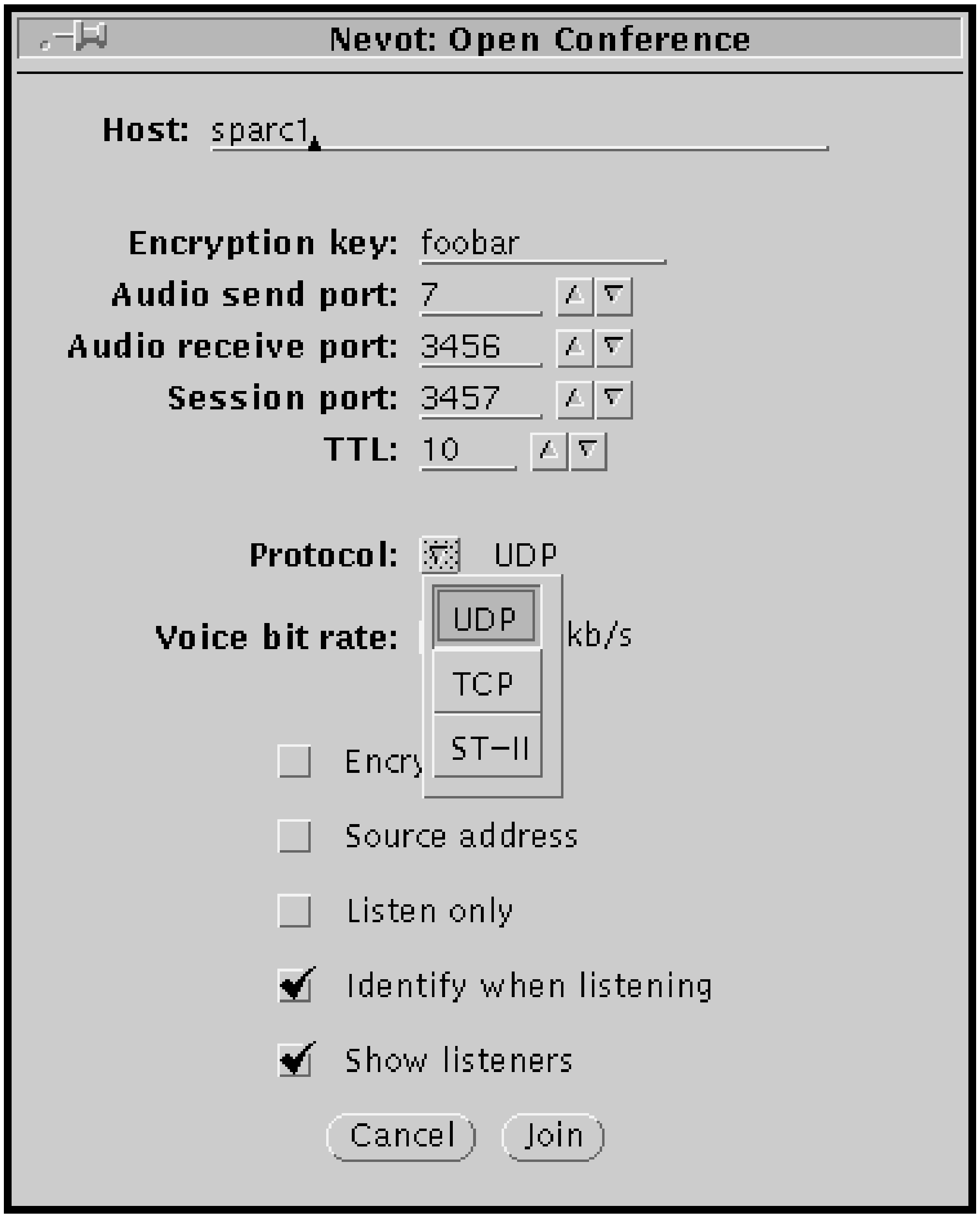
```
TARGET desperado.ecs.umass.edu
  LOOSE_ST sparc1.ecs.umass.edu
PORT 3458
TARGET despot.ecs.umass.edu
```

The **encryption key** is used when encryption mode setting is chosen[4].

The **conference identifier** distinguishes several conferences using the same protocol and port. The fields labeled **audio send port** and **audio receive port** are filled in with the ports for sending and receiving audio data. In normal operation, both ports will have the same value, agreed upon by all conference participants. However, transmission quality over a link can be tested by specifying a send port number of 7, the echo port. The remote host will simply reverse sender and receiver address and return the audio packet. Currently, echo facilities are not available for ST-II. The **session port** is used to send and receive session control messages. The **ttl** field specifies the time-to-live of multicast packets. The value currently is interpreted only by multicast UDP conferences. The choice menu beneath the time-to-live field determines the protocol to be used, namely UDP (or multicast UDP), TCP or ST-II.

The audio encoding is set by the next choice menu. The supported encodings and their rates are listed in Table 1. Note that incoming audio data from different sites within the same conference can use different encodings, all outgoing voice data for a conference, however, uses the same encoding.

[4]The key is limited to 8 characters, where only the least significant 7 bits of each character are used. Control characters can be specified using the customary C language notation, i.e., \n for newline, \t for horizontal tab, etc., or \ooo as a three-digit octal number.



Nevot: Open Conference

Host: sparc1

Encryption key: foobar
Audio send port: 7
Audio receive port: 3456
Session port: 3457
TTL: 10

Protocol: UDP

Voice bit rate: kb/s

[ ] Encry
[ ] Source address
[ ] Listen only
[✓] Identify when listening
[✓] Show listeners

Figure 3: The conference set-up panel

The third column in the table indicates the increment in the packetization interval so that all packets have the same integer number of bytes. The LPC codec can be configured to compute the filter coefficients over different intervals, given by the **vocoder** parameter. Also, if the vocoder period is a submultiple of the packetization interval, several predictor sets are packed into a single network packet, amortizing the header overhead over a larger number of audio bytes. The vat LPC1 codec is equivalent to setting the vocoder period and packetization interval to 22.5 ms, while the LPC4 codec uses a packetization interval of 90 ms and a vocoder interval of 22.5 ms. If the vocoder intervals differ between sender and receiver, speech will appear to be pitch-shifted and slowed down or speeded up. Vocoder intervals above 25 ms degrade voice performance below the already barely acceptable communication quality achieved by LPC. The LPC codec allows to run packet voice over a 9.6 kbps SLIP (modem) connection.

name	kb/s	voice coding method	packetization increment (ms)
G.711	64	8-bit μ-law PCM	0.125
G.721	32	4-bit CCITT G.721 ADPCM	0.25
DVI	32	4-bit Intel/DVI ADPCM	0.25
G.723	24	3-bit CCITT G.723 ADPCM	0.33333
LPC	4.8	LPC codec	vocoder

Table 1: Voice encoding

The check boxes determine the characteristics of the conference. Checking off the encrypt box enables DES voice data encryption, with the key entered above. The listen only box disables the automatic creation of reciprocal ST-II connections. If the identify when listening box is checked, NEVOT sends out periodic messages containing the user and host name in a format compatible with vat even if we have muted our site. The feature may be useful to turn off that feature for large conferences with mostly passive audiences where it is undesirable to flood the network and the displays of participants with the names of all listeners. Conversely, the show listeners option indicates that sites that have not sent audio data are to be displayed. If not checked, only those sites that have talked recently are shown. The source address field was discussed earlier and is only used in conjunction with the packet reflector. Reverse name lookup, i.e., the mapping from Internet addresses to host names, can be enabled by checking reverse name lookup. Since name lookup may take an indeterminate amount of time, during which NEVOT is otherwise blocked and since most sites transmit site identification strings, the use of this option is generally not recommended. Checking the exclusive option ensures that if talking for this conference is enabled, talking to all other conferences is automatically disabled. This is useful for side chats, making it a bit less likely that what was considered a confidential remark gets distributed to the conference at large. — The join button establishes the conference.

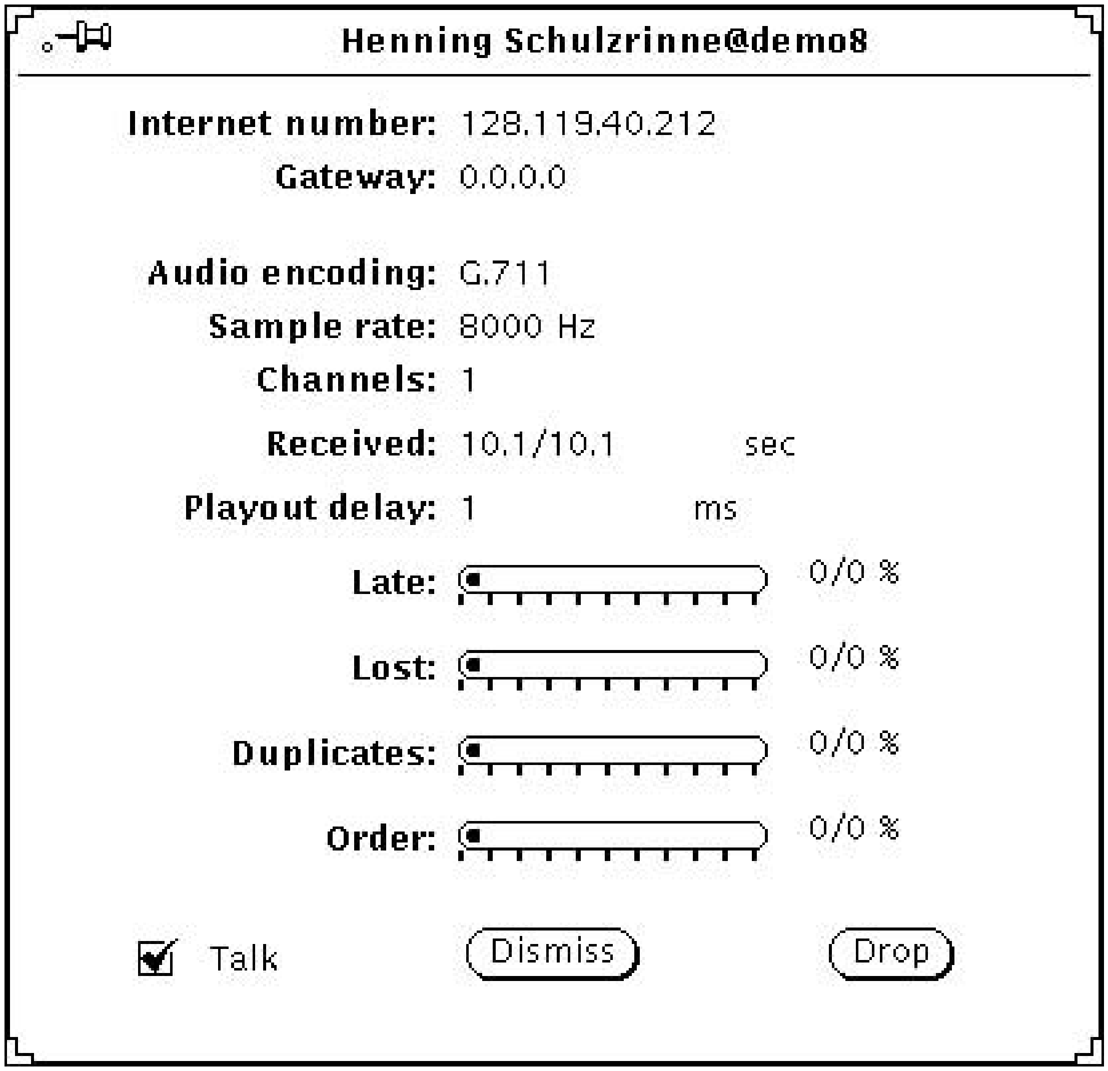
To leave a conference, press the leave button in the conference control panel. To add a new site to the conference, pressing the add button adds popup panel, where the site name can be filled in. Sites that send audio or control information are automatically added. Adding sites is unnecessary for multicast UDP conferences and thus the button is inactive and shown dimmed. The properties of an existing conference can be modified by modifying the conference set-up pop up, which is invoked by pressing the mouse menu button[5] while the pointer is within the conference control panel.

[5]typically, the right-most button

Each conference site has its own control panel. On the right, it displays the site identifier, which may be an Internet number (if the Internet number could not be translated into a host name), a host name or, if the other site is sending its identifier, the remote user name and host name. Note that it is very easy to spoof this identifier, so it should not be relied upon for authentication.

If there are a large number of sites in a conference, the site panels will overlap. The number of rows displayed per conference is given by the `max_height` configuration parameter. During a conference, you can resize the NEVOT window to allow more or less space for each site entry.

Clicking with the right (menu) mouse button on a site panel brings up the *status display* for that site (Fig. 4). It shows site statistics and features a button to drop the site and a check mark that enables talking to this site. Talking is by default enabled for multicast UDP, but must be enabled explicitly for ST-II and unicast UDP, establishing a connection in the outgoing direction. The status pop-up panel is dismissed by selecting the "dismiss" button or unpegging the pushpin.



Henning Schulzrinne@demo8

Internet number:	128.119.40.212
Gateway:	0.0.0.0
Audio encoding:	G.711
Sample rate:	8000 Hz
Channels:	1
Received:	10.1/10.1 sec
Playout delay:	1 ms
Late:	0/0 %
Lost:	0/0 %
Duplicates:	0/0 %
Order:	0/0 %

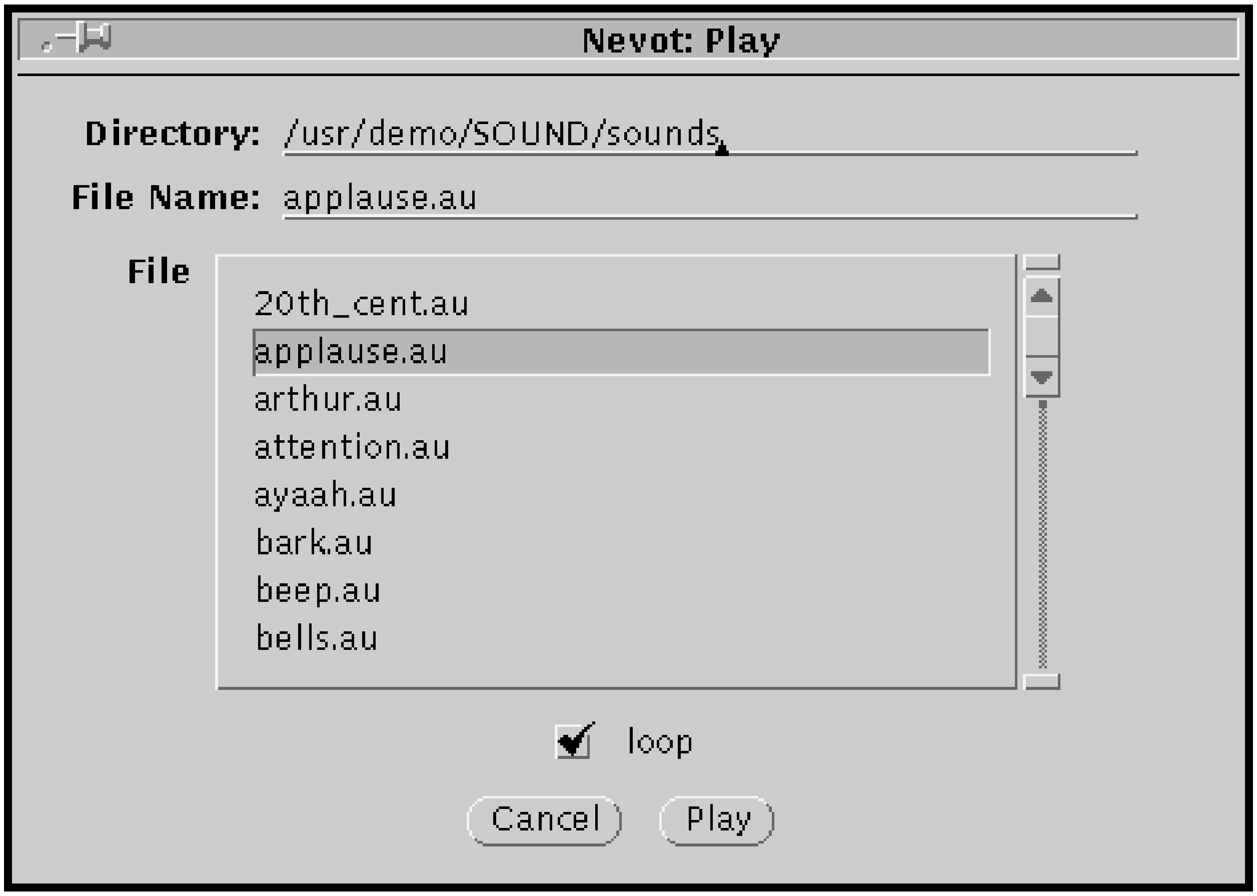
Figure 4: The site status pop-up

A single site can be dropped from the conference by selecting the Drop button in the site status panel. Sites that have been inactive for long periods of time are dropped automatically (see description of time outs in the properties menu).

Clicking on a site with the middle mouse button provides a simplified way to start a second conference. A join panel as in Fig. 3 is displayed, with the current conference characteristics such as encoding, port numbers, etc. The conference identifier is incremented by one and the exclusive option is checked. Naturally, any of the parameters can be changed just as when creating a conference using the Join button. However, either the port number, protocol or conference identifier must differ from all other conferences.

2.3 Recording, Playback and Volume Adjustment

NEVOT can play back audio files. The play button invokes a file pop-up menu (see Fig. 5[6]). Pressing play within the pop-up starts the playback. For standard Sun audio files (extension .au or .snd), the description is shown in the base window footer. The normal conference and site talk controls also apply while playing audio files. Naturally, the microphone is disabled during playback, but reception is unaffected. Putting a check mark in the looping check box plays the same sound file again and again. While playing, the play button becomes a stop play button. Selecting the stop or pause cancels or pauses playback. Playing resumes when selecting the resume button. While playing sound, the descriptive header information and length is displayed in the footer of the base window.


Nevot: Play

Directory: /usr/demo/SOUND/sounds

File Name: applause.au

File
20th_cent.au
applause.au
arthur.au
attention.au
ayaah.au
bark.au
beep.au
bells.au

Figure 5: The play file pop-up menu

Recording audio works in a similar fashion. In addition to directory and file name, the descrip-

[6]The second item in the list shows another possibility for this feature.

tion text field may be used to add a brief descriptive title to the sound recording. The recording can also be appended to an existing file if the appropriate check box is marked. Both outgoing and incoming audio is recorded, even if the global listen box is not checked.

The volume adjustment was intentionally left out of NEVOT. The Sun `gaintool` provides this functionality, in addition to side tone adjustment and switching between speaker and headphone jack output.

2.4 Configuration

NEVOT is heavily parameterized. Parameters can be set on start-up through an initialization file, by keyboard commands or through a pop-up menu. We will describe both methods in turn. The file `.nevotinit` in the path specified through the environment variable `NEVOT_PATH` is read on startup to initialize parameters. A path is a list of directories, with elements separated by a colon. The tilde notation is expanded to the respective home directory. If the environment variable `NEVOT_PATH` is undefined, the compiled default value is used (typically ".: "). The keyboard command `s` sets parameters while NEVOT is running. The parameters setable through the initialization file or keyboard commands are shown in tables 2, 3 and 4, below. In the tables, AGC refers to the automatic gain control, VU to the voice volume gauge, SD to the silence detector and DEL to the delay adjustment. The abbreviations VE denotes voice energy, ranging from zero to 127. String values can be enclosed in quotation marks; strings containing white space must be enclosed in quotation marks. Bits within flags are specified by concatenating the listed symbols, separated by vertical bars, `|`. White space is not allowed. A flag is negated by prefixing it with an exclamation mark or tilde. Example:

```
s mode del|!sd
```

sets the "del" bit and resets the "sd" bit in the "mode" flag. An example of an initialization file is shown below:

```
s trace_length 5000
s trace_events !audio_in|!audio_out|!transmit|!receive|!packet_loss|!silence|!AGC|!delay_adj
s agc_tc 1024
s agc_hyst 20
s agc_nom 4294967295
s agc_interval 0.5
s vu_tc 100
s vu_hyst 2
s vu_nom 0
s vu_interval 0.1
s sd_tc 0
s sd_hyst 8
s sd_nom 50
s sd_interval 0
s davg_tc 1000
s davg_hyst 2
s davg_nom 4
s davg_interval 2.5
s dvar_tc 1000
s dvar_hyst 2
s dvar_nom 4
s dvar_interval 2.5
s host "224.2.0.1"
s st_recv_port 3456
s st_send_port 3456
s st_session_port 3457
```

```
s st_ttl 127
s st_proto UDP
s st_conference_id 0
s st_mode !encrypt|!source_addr|!listen_only|id_listen_only|show_listeners|!resolve|!exclusive
s voice_coding G.711
s play_dir "/usr/demo/SOUND/sounds"
s play_file (null)
s play_loop 0
s play_gain -1
s rec_gain -1
s mon_gain 0
s play_port jack
s rec_port default
s play_channels 1
s rec_channels 1
s play_sample_rate 8000
s rec_sample_rate 8000
s rec_lowater 22.5
s play_lowater 22.5
s play_hiwater 90
s rec_hiwater 90
s packetization 22.5
s vocoder 22.5
s soft_to 1
s hard_to 10
s check_interval 6
s key ""
s repeat_th 40
s echo_th 30
s before_spurt 40
s after_spurt 100
s silence_sub 0
s echo_supp 0
s verbosity 2
s mode !mike_loop|!af_loop|!af_only|sd|!agc|vu|del
s user "%n@%h"
s protocol vat
s role end_node
s max_height 2
```

The properties pop-up menu is invoked by pressing the mouse menu button while in the global control panel (i.e., the panel at the top, not associated with a conference or site).

The parameters have the following meaning:

Silence, lost packet: Determines what happens if a packet is lost or there is silence (i.e., no packet to be played out). NEVOT either repeats the last packet or inserts actual silence. Repeating the last packet reducing the push-to-talk effect where the speaker background noise of the speaker is cut off abruptly during silence periods.

Audio before/after talkspurt: Determines the time (in milliseconds) of "silence" sent before and after a talk spurt, reducing front and end clipping.

Audio low/high water mark: To compensate for non-periodic scheduling, the operating system buffers a number of bytes before playout. This number determines the range of acceptable buffering. Too low a value will lead to clicks, particularly if the system is busy. A high value incurs additional play out delay.

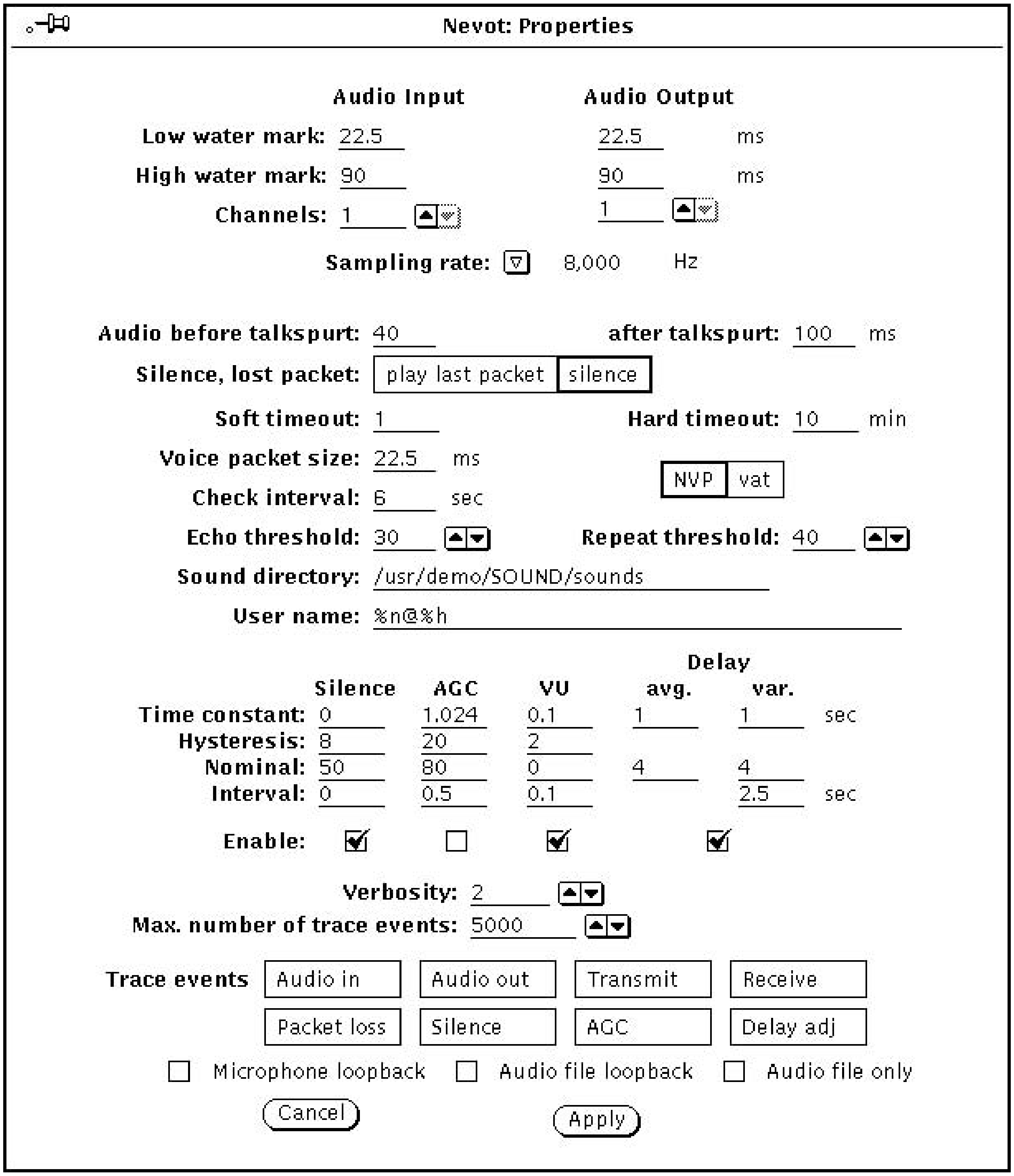
`agc_tc`	AGC: time constant (ms)
`agc_hyst`	AGC: hysteresis (VE)
`agc_nom`	AGC: set point; desired energy level
`agc_interval`	AGC: adjustment interval
`davg_tc`	delay average: time constant (ms)
`davg_hyst`	delay average: hysteresis (VE)
`davg_nom`	delay average: variance multiplier
`davg_interval`	delay average: adjustment interval
`dvar_tc`	delay variation: time constant (ms)
`dvar_hyst`	delay variation: hysteresis (VE)
`dvar_nom`	delay variation: variance multiplier
`dvar_interval`	delay variation: adjustment interval
`vu_tc`	VU meter: time constant (ms)
`vu_hyst`	VU meter: hysteresis (VE)
`vu_nom`	VU meter: currently not used
`vu_interval`	VU meter: update interval (sec)
`sd_tc`	silence detection: time constant (ms)
`sd_hyst`	silence detection: hysteresis (VE)
`sd_nom`	silence detection: max. threshold
`sd_interval`	silence detection: update interval for minimum (sec)

Table 2: Filter parameters

`play_file`	default playback file name
`play_dir`	default directory for sound files
`play_loop`	play same audio file again and again
`rec_gain`	audio recording gain; −1: leave as is
`play_gain`	audio playback gain; −1: leave as is
`mon_gain`	audio monitor gain; −1: leave as is
`rec_port`	audio input port: `default`, `line`, `mic` or `digital`
`play_port`	audio output port: `default`, `speaker` or `jack`
`rec_channels`	audio record channels
`play_channels`	audio output channels
`rec_sample_rate`	audio input sample rate
`play_sample_rate`	audio output sample rate
`before_spurt`	packets before talk spurt
`after_spurt`	packets after talk spurt
`play_lowater`	the minimum occupancy of the audio output buffer buffer (ms)
`play_hiwater`	the maximum occupancy of the audio output buffer buffer (ms)
`rec_lowater`	the minimum occupancy of the audio input buffer buffer (ms)
`rec_hiwater`	the maximum occupancy of the audio input buffer buffer (ms)
`packetization`	packetization interval (ms)
`vocoder`	vocoder interval (ms)
`silence_sub`	silence substitution algorithm
`echo_supp`	echo suppressor flag
`repeat_th`	don't repeat if packet energy is above this threshold
`echo_th`	threshold below which microphone audio is treated as echo
`voice_coding`	audio encoding[a]

[a]`G.711`, `G.721`, `G.723`, `CELP`, `LPC`, `DVI`, `linear 8`, `linear 16`

Table 3: Audio and audio file parameters


Nevot: Properties

	Audio Input	Audio Output	
Low water mark:	22.5	22.5	ms
High water mark:	90	90	ms
Channels:	1	1	

Sampling rate: 8,000 Hz

Audio before talkspurt: 40 after talkspurt: 100 ms

Silence, lost packet: play last packet [✓ silence]

Soft timeout: 1 Hard timeout: 10 min

Voice packet size: 22.5 ms [✓ NVP] vat

Check interval: 6 sec

Echo threshold: 30 Repeat threshold: 40

Sound directory: /usr/demo/SOUND/sounds

User name: %n@%h

	Silence	AGC	VU	Delay avg.	Delay var.	
Time constant:	0	1.024	0.1	1	1	sec
Hysteresis:	8	20	2			
Nominal:	50	80	0	4	4	
Interval:	0	0.5	0.1		2.5	sec
Enable:	✓	☐	✓	✓		

Verbosity: 2

Max. number of trace events: 5000

Trace events: Audio in, Audio out, Transmit, Receive, Packet loss, Silence, AGC, Delay adj

☐ Microphone loopback ☐ Audio file loopback ☐ Audio file only

Cancel Apply

Figure 6: The properties pop-up

`trace_length`	number of trace events
`trace_events`	events to be traced[a]
`st_send_port`	default audio send port
`st_recv_port`	default audio receive port
`st_session_port`	default session control port
`st_ttl`	time-to-live for multicast packets
`st_proto`	default protocol: `UDP`, `TCP`, `ST-II`
`st_conference_id`	default conference identifier
`st_mode`	stream mode[b]
`soft_to`	soft time-out (min)
`hard_to`	hard time-out (min)
`check_interval`	interval to send identifier and check time-out (sec)
`verbosity`	0: no output; 1: minimal messages on stdout
`mode`	flags: `mike_loop`, `af_loop`, sd, agc, vu, del
`user`	user name format, as described on p. 15
`protocol`	audio packet format: `vat` or `nvp`
`role`	role: `end_node` or `gateway`
`max_height`	maximum number of rows per conference

[a] `audio_in`, `audio_out`, `transmit`, `receive`, `packet_loss`, `silence`, `AGC`, `delay_adj`

[b] values: `encrypt`, `source_addr`, `listen_only`, `id_listen_only`, `show_listeners`

Table 4: Network parameters

Soft/hard time out: The waiting time, in minutes, after the last audio/control packet is received before a site is timed out.

Voice packet size: The voice packetization interval, in milliseconds. Sites within a conference may use different packetization intervals. The voice packetization interval must yield a integer packet size after encoding. See the voice encoding table above for acceptable increments. The standard packetization interval that ensures interoperability with `vat` is 22.5 ms.

NVP/vat: The audio packet format, either NVP or the `vat` private header format. All conferences must use the same format; thus, you can switch between the two only prior to opening the first conference.

Check interval: The check interval is the time after which the program sends out an identifying message to all other conference participants. This is also the granularity with which time-outs are checked.

Echo threshold: If voice packets with average energy below this threshold are encountered while other users are talking, it is assumed that the microphone is picking up the audio of these other users. This mechanism is enabled only when the echo suppressor box is checked.

Repeat threshold: Packets with energy above this threshold are not repeated during silence periods.

Sound directory: The default directory for sound files.

User name: The message to be send with the session protocol. The string may contain format characters which are replaced by the current value :

%e	telephone extension (e.g., `x3179`)
%h	host name (e.g., `gaia.cs.umass.edu`)
%i	host Internet number (e.g., `128.119.40.186`)
%n	real user name (e.g., `Henning Schulzrinne`)
%o	office room number (e.g., `A203`)
%p	home phone number (e.g., `555-1212`)
%t	terminal device name (e.g., `/dev/tty01`)
%u	the user login name (e.g., `hgschulz`)
%%	the percent sign itself

typically, user name and host, but can be anything. This can be used for a crude messaging protocol.

Silence: This column determines the parameters for the silence detector. Currently, the time constant is not used. The hysteresis determines the amount by which the energy of a packet must exceed the current minimum average. The maximum silence threshold is given by the sum of the nominal value and the hysteresis. The interval determines how long the silence detector waits during a talk spurt before raising the minimum average.

AGC: This column contains the parameters for the automatic gain control, namely time constant, hysteresis (i.e., the band around the nominal energy value where no gain adjustment is made), the desired average energy value and the adjustment interval.

VU: This column contains the parameters for the VU meter, both for incoming and outgoing audio. The VU display is updated with a period given by the **interval** entry. The nominal value is ignored.

Delay avg.: This column contains the parameters for the delay adjustment. The nominal value is used as the initial delay (in bytes).

Delay var.: The fields in this column describe the adjustment filter for the delay variation. The nominal value denotes the factor that is used to multiply the delay variance estimate to arrive at a new play out delay.

Enable: Filters are enabled if the box underneath the parameter column is checked.

Verbosity: The amount of information printed to `stderr`. Values above 2 are useful only for debugging.

Number of trace events: This limits the number of events that are written to the tracefile.

Trace events: The buttons determine the events that will be recorded in the trace file (see Section 2.6).

Microphone/Audio file loopback: If checked, audio from the microphone or the currently playing audio file is mixed in with the remote audio. Microphone loopback is primarily useful for testing, while audio file loopback allows monitoring of the playback progress. It is also useful when using the echo port, as it provides immediate acoustic feedback of the length of the playback delay.

Audio file only: Only send audio data from play files, not from the A/D converter. This option is particularly useful to create reproducible runs for debugging or performance measurements.

Pressing Apply accepts the current parameters and dismisses the pop-up window. The parameters are automatically saved when leaving the program.

2.5 Keyboard Control

Most functions described above can also be invoked through the keyboard. For the `curses` and `stdio` version, this is the only form of interaction with NEVOT. For the XView version, the typed commands are displayed in the status line. To apply a command to a specific conference, type its position, where the first conference (displayed top most) is numbered one. The conference number only applies to the next command. The commands and their arguments are:

`j` *host*$_1$ *host*$_2$ ...
: If no current conference, create new conference with hosts listed otherwise, add host to existing conference. Host entries follow the format described in Section 2.2.

`l`
: Leave specified conference.

(space bar)
: Toggle microphone muting, globally or for numbered conference.

`s` *parameter value*
: Set parameter *parameter* to value *value.*

`s` *parameter*
: Show value of parameter *parameter.*

`p` *file*
: Play audio file *file*; if already playing an audio file; this command toggles between pausing and resuming playback.

`r` *file*
: Start recording into file *file*; if already recording, this command toggles between pausing and resuming recording.

`?`
: Show current short statistics. The statistics are shown per site in the form "total/since last statistics". Shown are: the total number of packets received, percentage of packets that were late, lost, duplicated and out-of-order as well as the current smoothed average playout delay estimate in ms, measured from the time of arrival to the time of submission to the audio stream device (i.e., not counting operating system audio buffering).

`q`, `x`, `Q`
: Quit NEVOT.

2.6 Traces

As an aid in troubleshooting and performance analysis, a number of events can be traced. The events are dumped to disk in binary format.[7] Each event record has the following format:

[7]A special version of the trace routines accumulates events in a main memory area and dumps them to disk as the program terminates.

```
typedef struct trace_t {
  union {
    u_long l;
    u_char b[4];   /* byte components */
  } addr;          /* applicable address */
  struct timeval tv;
  long v[4];       /* values */
  char event;      /* event code */
} trace_t;
```

The event codes and the interpretation of the event values are summarized in the tables below.

trace category	events
AGC	V
audio_in	A, a
audio_out	P, p
delay_adj	d
packet_loss	=, *, #
receive	R, !
silence	S, s
transmit	T

Table 5: The trace categories

In addition, a file named as *s*.sta, where *s* denotes the program starting time, contains summary statistics, including resource utilization. An example is shown below:

```
Program started: Wed Jul 22 16:49:54 1992
Elapsed time:                  57.487 sec
User time used:                 4.950 sec
System time used:               5.389 sec
Messages sent:                     68
Messages received:                164
Signals received:                   1
Voluntary context switches:      1655
Involuntary context switches:    1646
Process swapped out:                0
Block input operations:             6
Block output operations:            5
Maximum resident set size:        456 pages
Integral resident set size:    442935 pages * clock ticks
Current time: Wed Jul 22 16:50:51 1992
Audio samples from A/D:    263880
Audio underflows:               1
Silent packets:                 0
STREAM: UDP to (224.2.0.1,3456).
Packets sent:                   0
Jeff Bailey (Kent State Univ) [131.123.2.60/0.0.0.0, S=3456 R=3456]:
```

variable	meaning
t	packet time stamp
t_1	packet time stamp of first packet in talkspurt
t_∞	packet time stamp of latest packet
s	packet sequence number
s_1	packet sequence number of first packet in talkspurt
a	packet insertion location in ring buffer
a_p	next play out location
D	delay for beginning of talkspurt (bytes)
d	actual delay (bytes)
$\hat{d}$	smoothed average delay estimate (bytes)
$\widehat{\lvert d-\hat{d}\rvert}$	smoothed delay variance estimate (bytes)
$\hat{e}$	smoothed energy estimate
V	length of silence period, in packetization intervals
e	power estimate for packet
m	maximum absolute value
$\bar{x}$	average sample value
s_r	audio samples read from audio buffer
s_R	total audio samples read from audio buffer
s_q	samples in queue
s_W	audio samples written to audio device
p_R	audio packets read from audio buffer
S	stream sequence number

Table 6: Meaning of the trace values

code	function	value 1	value 2	value 3	value 4
a	audio in	p_R	s_R	s_q	s_r
A	audio in error	p_R	s_R	s_q	s_r
T	transmit audio p.	p_R	s_R	S	0
P	play	s_q	a_p	m	e
p	play error	s_q	a_p	m	e
d	delay update	$\hat{d}$	$\widehat{\lvert d-\hat{d}\rvert}$	D	0
R	received	t	s	a	d
H	corrupted header	header	0	0	0
*	late packet	t	s	a	d
=	duplicate packet	t	t_∞	a_p	0
#	packet reordering	t	t_∞	a_p	0
!	talk spurt	t_1	s_1	a	D
S	silence period	p_R	V	0	0
s	silence	p_R	m	e	$\bar{x}$
V	agc	$\hat{e}$	gain	0	0

Table 7: Trace events and their associated values

```
davidc@sirius.net.Hawaii.Edu [132.160.3.9/0.0.0.0, S=3456 R=3456]:
```

3 Nevot Implementation

NEVOT comes in four flavors. The first is built using the Open Look GUI and was implemented in XView, the Sun X toolkit. The basic window structure was generated by DevGuide. The second uses the Motif widget set. The third and fourth versions are command-driven and display state through the `curses` terminal-independent screen library or simple sequential terminal output. The four versions share most of the network and signal processing code. Some of the ideas and code are drawn from VT, the USC/ISI voice terminal, and `vat`, Van Jacobson's conferencing tool. NEVOT is compatible with both of these tools.

NEVOT allows the participation in several concurrent conferences by maintaining a separate *stream* descriptor structure for each. Each stream description in turn contains a list of site descriptors. Individual streams may use different network protocols, but all streams have to use the same audio packetization duration for outgoing audio data. Each stream can use its own audio encoding method for all outgoing audio packets. (This limitation is unavoidable for ST-II and UDP multicast transport since the voice terminal submits only one audio packet to the network for all conference participants at each packetization interval.) NEVOT allows each site to use its own audio encoding method for incoming audio data. The audio encoding method used is carried in the second octet of the `vat` session protocol, and thus can be easily changed dynamically.

The general program structure is shown in Fig. 7. Dashed lines indicate that modules of that type are accessed through tables of function pointers, easing the integration of additional protocol or data types.

3.1 Network I/O

The structure of NEVOT is complicated by the desire to support three different transport protocols, namely UDP (and multicast UDP), TCP and ST-II. The transport-dependent parts have been largely isolated in the modules `udp.c`, `tcp.c` and `st2.c`, with function pointer references in the general-purpose code. Each stream maintains a separate timestamp reflecting the number of packetization intervals while this stream was active. Regardless of the audio transport protocol used, session data is always transmitted and received by a separate unicast or multicast socket. Dynamic buffers are allocated from an mbuf-like pool of fixed-size buffers occupying a contiguous memory area.

Unicast and multicast UDP require one socket per stream, used for both outgoing and incoming audio data. The mapping from incoming packet to site has to be done by searching the site list for a matching address, port and conference identifier. The search is speeded up by maintaining a one-deep address translation cache. Outgoing unicast UDP data has to be sent to each site separately, while a single send operation suffices for multicast streams.

TCP uses a bidirectional socket for each site, plus one socket to listen for new connection requests. The socket uniquely identifies the site, so that address matching is not required. The TCP stream knows no record boundaries; the packet length is determined from a two-byte packet length prefixed to the actual data packet. Unlike the datagram protocols, several read operations may be required to acquire a single packet from the network.

ST-II sends all outgoing audio data through a single socket; through the same socket, control messages such as notification of connection acceptance or closings are received. Each incoming stream uses its own socket, making address lookup for incoming packets unnecessary. As for TCP, a listen socket accepts new connections.

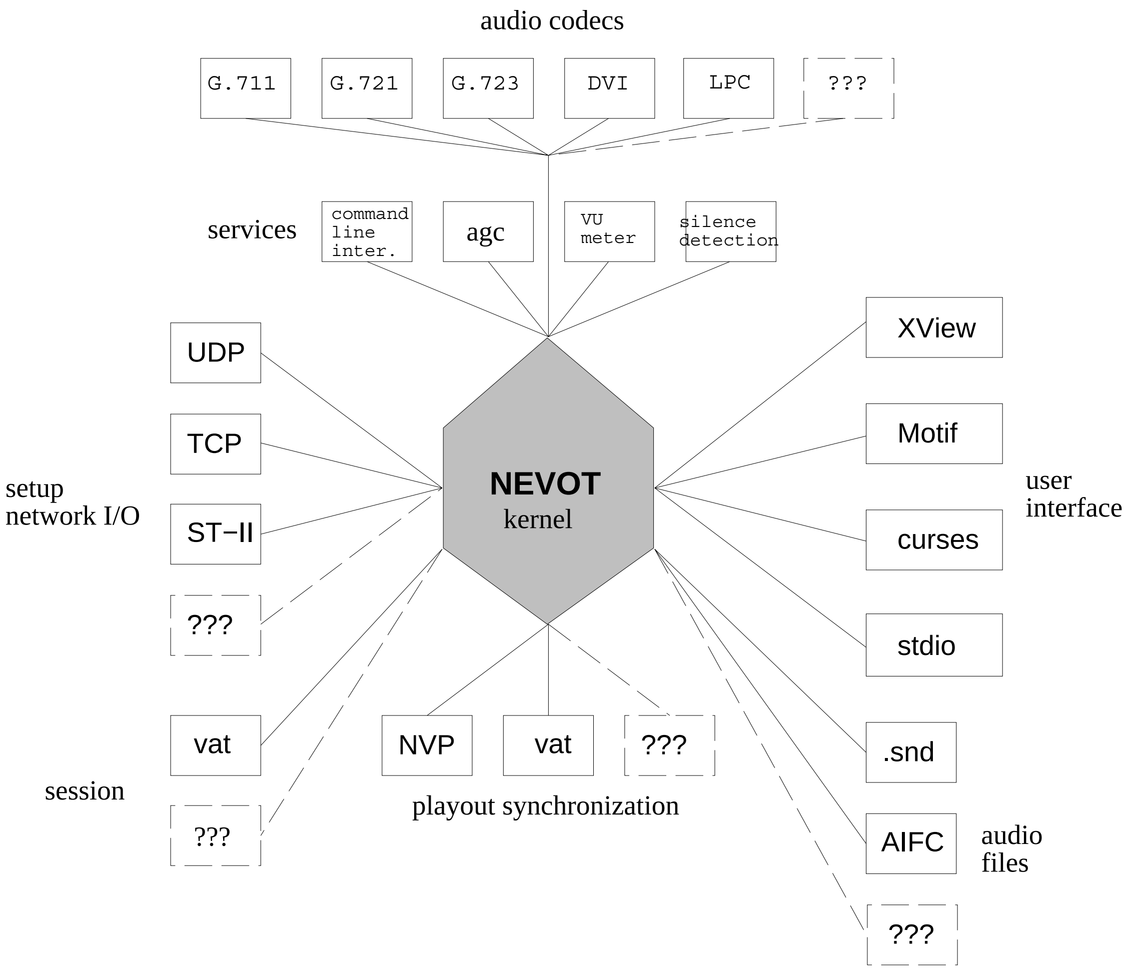


Figure 7: NEVOT Structure Overview

Application-level multiplexing requires additional machinery. Streams with the same network-level port and protocol are considered part of the same family, tied together as a linked list. The network sockets are not closed until the last member of the family has ceased to exist. This allows a large number of concurrent side chats, even without the ability at the socket API to distinguish whether a packet originated from a multicast or unicast address.

3.2 Audio Synchronization

3.2.1 Audio Timing

Audio must be played out, that is, submitted to the digital-to-analog converter, synchronously (at fixed intervals), despite network impairments and operating system restrictions. The network over which we wish to communicate may loose, corrupt, reorder and duplicate packets. Also, the digitization clocks at different stations will not be exactly the same. The operating system cannot guarantee that the process running the network voice terminal is scheduled at predictable time instances. The operating system issue is discussed in section 3.6.

The playout process is clocked by the analog-to-digital converter: a packet is played out for every complete sample buffer received from the A/D converter. This clocking is also used while playing audio files, even though the audio data from the A/D converter itself is not used. This scheme has the disadvantage that a single audio buffer may be read in several increments, depending on the granularity of the underlying stream buffering[8]. As an alternative, the system interval timer could be used or we could simply block on audio input as soon as the `select()` return indicates that at least some data is waiting to be read.

3.2.2 Playout Buffer

The initial design had each site maintain a separate circular list of buffer pointers. The circular list makes insertion very efficient, consisting of a single modulo operation and pointer copy, without actually touching the data returned from reading the socket. Also, detection of missing packets or silence at playout time is easy: the buffer pointer in the circular buffer is nil if no packet has arrived for that slot. In that design, strict synchronization was maintained through talkspurts and silence periods, i.e., the silence period duration at the transmitter was replicated at the receiver, as long as the playout delay estimate did not change. The delay variability between packet arrival and playout time was measured and used to compute a desirable playout delay. If this delay differed from the one currently in use more than a setable hysteresis value, silence periods were used to bring the two in alignment, by either skipping a silent buffer to decrease playout delay or replicating a silent buffer to increase playout delay. This scheme was seen to occasionally lose synchronization, especially after long silences where the time stamp value had wrapped around several times.

If a received packet time stamp fell outside the range covered by the circular buffer or if too many packets were lost in a row, it was taken as an indication that transmitter and receiver have completely lost synchronization and the circular buffer was reset. This loss of synchronization may occur, for example, if a transmitting site is restarted or after a failed network link recovers.

For a number of reasons, the individual circular buffers were abandoned in favor of a single contiguous playout buffer[9] shared by all sites and streams. With the single buffer, packet sizes for different sites can differ and are not forced to be the same as the packetization interval. Secondly,

[8]It would be helpful if the `select()` call allowed specification of a minimum amount of data required before considering a file descriptor as ready. The SGI audio library allows this specification indirectly, while a kernel variable has to be patched for SunOS.

[9]Currently, generously sized at 90000 bytes.

it becomes possible to vary the playout delay in byte rather than packetization interval increments. Also, each site does not have to maintain a separate buffer point array of approximately 1000 bytes each (for 5.7 second maximum reconstitution delay). The single buffer has the disadvantage that every insertion implies mixing since we cannot easily determine whether a region within the circular buffer is empty, partially empty or already filled (recall that packet sizes are allowed to differ between sites). Also, for this reason, we have to clear the ring buffer after it has been copied to the audio stream[10].

Ring buffer wrap-around needs to be handled since the site packet sizes are not constrained to fit integrally into the ring buffer size. The mixing routines check whether the request reaches beyond the physical buffer end and split the request in two, the first reaching to the end of the physical buffer, the second covering the first few bytes. The logical buffer size is chosen as an integral multiple of the audio buffer size so that the playback routine does not have to worry about buffer wrap-around.

The details of the playout synchronization depend on the network audio protocol used. NVP uses a combination of a timestamp and sequence number, while the `vat` protocol features a longer timestamp and a talkspurt bit. The details are described below.

3.2.3 Synchronization for NVP Audio Protocol

NVP packets carry timestamps that are incremented every packetization interval, modulo `TS_MOD` (1024), regardless of whether voice is transmitted or not. The sequence number is incremented (modulo $\texttt{SEQ\_MOD} = 64$) for every packet transmitted and should thus be received without gap. We define a to be less than or equal than b modulo m if $a \leq b$ or $a > b \cap a - b > m/2$.

To compensate for reordering, each site maintains the time stamp and sequence number for the latest packet. A packet is declared to be the latest if its sequence number is greater than any other previously received, modulo `SEQ_MOD`. This will fail if more than $\texttt{SEQ\_MOD}/2$ packets are lost in a row.

Counting missing packets is also made more complicated by the single buffer, variable packet sizes, reordering and losses. We determine the number of packets sent by the counting the number of sequence number wrap-arounds. A wrap-around is detected if the new packet is determined to be the latest packet, as defined above, and its sequence number is smaller in value than the previous latest packet. The total number of packets that should have been received is then given by the number of wrap-arounds r and the earliest and latest sequence number seen, s_0 and s, as

$$\begin{array}{ll} (r-1)\texttt{SEQ\_MOD} + \texttt{SEQ\_MOD} - s_0 + (s+1) & \text{for} \quad r > 0 \\ s - s_0 & \text{for} \quad r = 0 \end{array}$$

Duplicate packets cannot simply be mixed in as the audio volume of that particular segment would increase, possibly resulting in clipping. Currently, a packet is declared a duplicate and ignored if its sequence number and time stamp matches that of the latest packet. This fails in the unlikely event that exactly `SEQ_MOD` packets were lost and the next arriving packet has the timestamp of the latest packet seen previously. A packet duplicating other than that with the currently highest sequence number is not detected. (Indeed, this would be difficult to accomplish, short of keeping a list of received packets.)

Packet reordering is detected if the time stamp of the arriving packet is smaller than that of the latest packet. If sequence numbers have not wrapped around yet, we need to adjust our initial sequence number s_0.

[10]The clearing of 180 bytes takes approximately 420 μs or 2% of available time on a SPARCstation II.

The sender begins a new talkspurt when the time stamp difference between two packets with consecutive sequence numbers is greater than one. Due to packet losses and reorderings, the receiver definition has to be slightly more general. The receiver declares the beginning of a new talkspurt if a packet is the latest packet and the difference in timestamp, Δt, exceeds the difference in sequence number, Δs. Each site remembers sequence number, s_1, time stamp, t_1, and ring buffer position, a_1, of the packet initiating the current talkspurt.

Since the beginning of each talkspurt is placed D bytes ahead of the current playout pointer a_p, a drastic change in D may cause the new talkspurt to overlap the end of the previous talkspurt. If the new a_1 is less than or equal to (modulo ring size) the insertion point of the latest packet, a_1 is set to that point plus one site packetization interval. Due to ring buffer wrap-around, this algorithm fails and thus needs to be disabled after very long silence periods. We simply skip the check for this case when the playout pointer has passed by the last packet of the previous talkspurt. This has occurred if the delay measured for that last packet is less than the number of packetization intervals that have occurred since that time. (Every site tracks the packetization sequence number seen by the latest arrival.)

Packet reordering within the network may cause the second packet of a talkspurt to arrive before the first. (Unfortunately, the close succession of packets at the beginning of a talkspurt makes reordering there particularly likely.) We simply insert this reordered packet before the packet triggering the new talkspurt. During talkspurts exceeding `ts_mod/2` packets (roughly 11.5 seconds for the canonic packetization interval of 22.5 ms), the difference in timestamp between the arriving packet and the beginning of the talkspurt would lead us to the erroneous conclusion that the packet predates the talkspurt beginning. The problem is avoided by adding `ts_mod/2` to t_1 and the corresponding byte offset to a_1 if $(t - t_1)$ exceeds `ts_mod/2`.

The insertion point a of a packet is given by

$$\mathsf{a} = (a_1 + p(t - t_1)) \bmod R$$

where R is the ring size, p the packet size, a_1 is the insertion point of the first packet in a talkspurt. At the beginning of a talkspurt, a_1 is set to a_p, the ring location of the next buffer to be played out, plus the current playout delay D, as estimated below, modulo R.

A packet is late if its playout time has passed. Instead of discarding it, we declare it the beginning of a new talkspurt. It remains to be seen whether this approach is more robust.

3.2.4 Synchronization for `vat` Audio Protocol

The `vat` "native" audio header contains 32-bit timestamp, incremented for every audio sample rather than for every frame, and a one-bit flag indicating the beginning of a talkspurt. This scheme has the advantage that packet reordering does not affect the playout delay, but the absence of sequence numbers makes it difficult for the receiver to determine the amount of packet loss. If the first packet in a talkspurt is lost, two talkspurts are merged and one opportunity to adjust delays is missed, but unless talkspurts are extremely long, this should have no dire consequences.

Other aspects, such as late packets and the insertion point computation, are handled as for NVP.

3.2.5 Playout Delay Estimation

The playout delay D is set to some constant, say, three or four, times a delay variance estimate described below. The factor can be intuitively justified by treating the delay distribution as normal and postulating that less than 0.1% of the packets should be late. This estimate would appear to be more robust then counting the rare events of late packets, as done in VT. The delay variance is estimated as the absolute difference between the current estimated mean delay and the delay sample. The first absolute moment is considered a more robust estimator and not as sensitive to outliers as the standard deviation. For normal random variables, it is known [29, p. 111] that

$$E[|x|] = \sigma\sqrt{2/\pi} = 0.798\sigma.$$

Note that this differs from the conclusion drawn in [30, p. 325]. The reconstruction delay sample d is defined as $a - a_p$, where a is the value before the late correction and thus d may be negative.

At the beginning of a talkspurt, several packets are sent in close succession, namely the packet whose energy level triggered the silence detector plus a fixed number of packets stored to limit front clipping. Because of this mechanism, the actual average playout delay will typically be larger than D, as illustrated in Fig. 8. Also, the delay variance is increased.

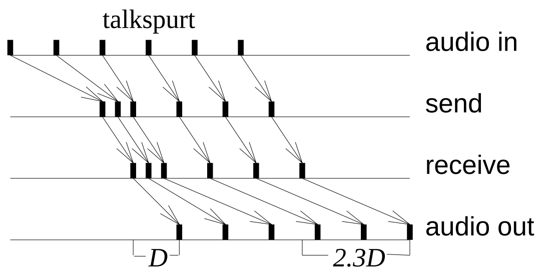


Figure 8: Effect of packet caching on playout delay

3.3 Audio Encoding

The μ-law transfer characteristic is given by [31, p.]

$$c(x) = x_m \frac{\log(1 + \mu x/x_m)}{1 + \mu}$$

where $c(x)$ is the coded value corresponding to input x, with the maximum absolute value of x given by x_m. μ has a value of 255. The companding gain is given by $\mu/\log(1 + \mu)$. This characteristic is approximated by a piecewise-linear function to ease translation between linear and μ-law encodings. In the G.711 encoding, bit 1 (the most significant bit) is used for the sign, bit 2 to 4 for the segment and 5 through 8 for the level within the segment.

$\mu - law$ encoding yields a maximum signal-to-noise ratio of 38 dB, with a dynamic range of ± 4096 or 13 bits linear. Other ADPCM-type encoders (CCITT G.721 and G.723) provide almost the same quality at 32 and 24 kbps, but their processing requirements make them unsuitable for less powerful workstations.

3.4 Low-Pass Filters

VU meter, AGC and delay adjustment employ a first-order recursive low pass (i.e., infinite impulse response) filter with unity zero frequency gain:

$$y_{i+1} = \alpha y_i + (1 - \alpha)x_i = y_i - (1 - \alpha)(y_i - x_i)$$

The time constant of this filter is $\tau = e^{-T/\tau} \approx T/(1-\alpha)$, where T is the sampling interval. The approximation holds if the time constant is at least, say, ten sampling intervals long. The filter can be implemented in fixed-point arithmetic using only two shifts and two additions per sample if we are willing to limit the set of achievable time constants. With y in fixed-point binary, we can write:

$$y_{i+1} = y_i + 2^a x_i - 2^{-b} y_i$$

The multiplications by 2^a and 2^{-b} are implemented as shifts. The values of a and b depend on the sampling interval T and the time constant τ:

$$\begin{aligned} a &= \Delta - b && (1) \\ b &= -\left\lfloor \log_2(1 - e^{-T/\tau}) + 0.5 \right\rfloor && (2) \end{aligned}$$

The offset Δ determines the number of fractional binary digits. This fixed-point arithmetic limits the effect of round-off errors. The same basic filter is used by TCP in the round trip time estimator [27, p. 188,192], [32, p. 278] and was first suggested by Van Jacobson [30].

For VU and AGC, a hysteresis value prevents control reactions to minor excursions around the set point. Also, the adjustment interval limits updates to a submultiple of the sampling period; this is helpful as the adjustment may be expensive in terms of CPU time, for example, updating the VU meter.

3.5 Talker Indication

Since the delays between packet arrival and playout can be substantial, it is unsatisfying to indicate the talker at the time of packet arrival. On the other hand, mixing at the time of arrival makes it difficult to determine the speaker at playout time without maintaining additional state. First, a ring of bit masks was used, with the ring location a submultiple of the audio insertion location. (There is no real advantage in indicating talkers to 20 ms resolution.) However, this limited the total number of sites to some fixed number. Instead, a four-entry array of site pointers is used, under the assumption that the occurence more than four simultaneous talkers is unlikely. A counter determines the insertion location. At playout, it is checked whether the pointer within the talker ring buffer has left the current location. (As pointed out, one talker ring buffer usually covers several packetization intervals.) If so, all sites active in the last period but not listed in the array for the current period are marked as no longer talking. A timestamp as the site talk status field speeds the comparison between current and previous array, as we cannot rely on the fact that the site entries are ordered in the same manner. The talking status set routine also has to deal with the case that the interval "covered" by a single voice packet may be larger than the time represented by a single slot, as tends to occur with low-rate codecs.

3.6 Audio Buffer Occupancy Control

The variability introduced by the non-deterministic scheduling of the voice terminal process is compensated for by the stream buffer. The program checks the buffer occupancy on playback and, during silent periods, inserts additional silence buffers or skips a silent buffer if the buffer content falls below a low water mark or rises above the high water mark, respectively. The two adjustment mechanisms also counteract clock skew between transmitter and receiver.

3.7 The Session Protocol

A separate datagram socket is used to transmit and receive session messages. Until some agreement on a full session protocol can be reached, NEVOT uses the set of multicast datagrams employed by `vat`. These offer only minimal conference control, indicating participants and encodings, but without parameter negotiation, connection setup, discovery, floor control, etc.

All messages consist of a flag byte, a type byte and a conference identifier. The most commonly used message (type 1, `S_id`) simply contains the alias (identifier) of the remote site. `S_bye` messages contain no further data and signal that the source has disconnected. Message of type `S_idlist` are used by gateways and contain a list of (address, alias) pairs, plus the audio format used[11]

A time-out mechanism removes the site display and marks the site as closed if no audio data or identifying packet has been received from the remote site for a specified amount of time. This prevents sites from being displayed as active even though they have just left their voice application running. The site entry itself, however, is *not* removed, thus maintaining the site statistics. It is anticipated that the set of conference participants is sufficiently static that memory usage for maintaining the site entries is not a problem. The audio file descriptor is closed if there are no sites, eliminating the audio processing overhead and allowing use of the audio device by other applications.

Session messages are generated based on a packetization interval count as long as the audio device is open and through the system interval timer when the audio device is closed. The interval is randomized between 0.5 and 1.5 the nominal value to avoid synchronization between sites.

3.8 Automatic Gain Control and VU Meter

The microphone gain is controlled either manually or by an automatic gain control (AGC), which derives its control signal from low-pass filtered per-packet samples reflecting the signal energy within the packet. AGC adjustment takes place only during silent periods.

VU meters measure the short-term energy with a well-described frequency response [33]: the display should reach 99% of full scale 300 ms after applying the corresponding full-scale level, with an overshoot of between 1 and 1.5 %. This response can be achieved by a damped two-pole low-pass filter with a Q of 0.62 and a cutoff frequency of 2.1 Hz.[12].

However, traditional VU meters do not work particularly well for digital audio systems. While analog audio components typically have transfer characteristics that diverge more and more from the desired linear shape as the level exceeds the 0 dB point, a digital system clips hard, i.e., all input levels that exceed the level represented by the largest digital code word are clipped to that value. Thus, a digital audio system is much more sensitive to level overload than an analog one. For this reason, a peak indicator was chosen as a loudness indicator instead of the traditional damped VU meter. The case for the use of a peak indicator is strengthened by their use in DAT (digital audio tape) recorders[13].

Since the peak indicator is supposed to indicate clipping, the DC offset is not removed, even though this does distort the relationship between loudness and meter display. (DC offset is found mostly with 16-bit A/D converters; telephone-quality A/D converters cut off frequencies below 20 Hz or so.)

In NEVOT, the interval over which the input peak is measured can be set, with a resolution of one packetization interval. (In the SGI audio control panel, a rate of 18 to 20 interval per second

[11]The purpose of the latter is not quite clear.

[12]Malcolm Slaney, private communication

[13]Gints Klimanis, private communication

was found to be satisfactory.) To prevent display flicker and save CPU cycles, the VU meter is updated only when the new value differs by more than the specified hysteresis from the currently displayed value.

3.9 Silence Detector

The adaptive silence detector is borrowed from VTand described in [34, p. 6,7]. It computes a measure reflecting the average sample energy of the packet and declares it silent if this energy measure is below the current threshold. For μ-law audio, the energy measure is the average of the sign-stripped sample values and is really a geometric mean since the sum of logarithms is equivalent to the product of the sample values). The geometric average is known to be no greater than the mean [35, p. 293]. For linearly encoded audio, we apply a μ-law transformation to the sample average so that we again arrive at scaled energy values of between 0 and 127.

The threshold is the minimum running average, a quantity described below, plus a hysteresis. During talkspurts, the minimum running average is increased by one every `sd.interval` packets, as long as it remains below `sd.nom` plus the hysteresis value[14]. During silent periods, the minimum average is updated after every packet if the measured energy falls below the current minimum average. An adjustable amount of hangover packets are transmitted after a silent period has been declared. Also, a setable number of packets are stored in a cyclic buffer during the silent period and transmitted in rapid succession at the beginning of a talk spurt, reducing front clipping at the expense of increased and more irregular traffic.

3.10 Lost-Packet Reconstruction

Reconstruction of lost packets and silence fill-in is handled by the same mechanism, namely simple repetition of the last received frame. Lost packets and silence are handled in the same manner since there is no reliable way of distinguishing silence from lost packets until the end of the silence period. (Naturally, in many cases we do know that a packet has been lost, so that more sophisticated reconstitution algorithms are possible.)

3.11 Planned Enhancements

It would be desirable to decouple the audio recording and playback functions, including AGC and volume (VU) display, and create a separate "tape recorder" tool. However, it is not clear whether the additional overhead incurred by interprocess communication (probably through a stream socket) is tolerable. Clearly, operating system support for connecting stream sources through various processes with low overhead is called for. Adding the audio processing as stream heads and creating an "audio bus" similar to that found in professional mixers is probably the best long-term design alternative, albeit limited to System V based operating systems.

- echo suppression and/or cancellation.
- encoding of recorded audio files.
- distributed (prioritized FCFS) or centralized (token) floor control using session control packets (probably should be separate from voice module, controlling conference talk switch on NEVOT).
- porting to Personal DECstation or DECstation with multimedia board.

[14]This change from the original design avoids drop outs during sustained audio material, e.g., orchestral music.

- LPC-10 encoding.
- drag-and-drop for audio output files.
- integration with enhanced "talk"-like program for call setup
- "personal phone book" and user locator

4 Acknowledgements and Copyrights

The DES encryption module was developed by Steve Kent and John Linn of BBN Communications Corporation, Cambridge, MA and provided by Karen Seo of BBN. The audio library incorporating G.721 and G.723 audio compression was provided by Daniel Steinberg of Sun Microsystems. It may at some point be integrated into the regular Sun OS. The Intel/DVI ADPCM codec was slightly modified from sources by Jack Kansen (CWI) and is copyrighted 1992 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands (used by permission). Ron Frederick (frederic@parc.xerox.com) or Xerox PARC, Palo Alto, CA, contributed the LPC codec which is based on an implementation done by Ron Zuckerman (ronzu@isu.comm.mot.com) of Motorola which was posted to the Usenet group comp.dsp on June 26, 1992.

The ST-II API and kernel support was developed by Charlie Lynn at BBN. The ST-II API (`st2_api.h`) is copyrighted (c) 1991 by BBN Systems and Technologies, a division of Bolt Beranek and Newman, Inc. and used by permission. The UDP multicast kernel support was written by Steve Deering, Xerox Parc. Charlie Lynn (BBN) was helpful with some of the fine points of the ST-II API.

Advice on porting NEVOT to the Silicon Graphics platform was provided by Andrew Cherenson (SGI). Michael Halle (MIT) figured out how to get XView applications to display fonts at the design sizes. The VU meter is based on discussions with Gints Klimanis (SGI).

The audio mixing (mix.c) and checksum code (checksum.c) was taken from the ISI voice terminal (VT), copyright June 1991 by the University of Southern California, by permission. The silence detector and the ST-II code are modified versions of the respective parts of VT.

The `vat` session and audio protocol were implemented based on descriptions provided by Van Jacobsen.

The I/O flags interpreter (flags.c) is a modified version of software contributed to Berkeley by Chris Torek. Copyright (c) 1990 by the Regents of the University of California; used by permission.

A NEVOT Installation

A.1 General Installation

NEVOT is available for anonymous ftp from `gaia.cs.umass.edu`, file `pub/nevot/nevot-0.95.tar.Z`. Executables are found in the same directory.

In the `Makefile`, the following symbols can be set to 1 to enable non-standard features:

symbol	enables
MULTICAST	IP multicasting
STII	ST-II transport support
DES	DES (data encryption standard) encryption

For export control reasons, DES encryption is not available outside the United States. The master `Makefiles` are located in the `nevot/xview`, `nevot/stdio` and `nevot/curses` subdirectories. To allow installation for those without superuser privileges, the Sun `libaudio.a` library enhanced with ADPCM encoding functions is located in the `lib.sun` directory. For SGI, the XView libraries are included in `lib.sgi`. Execute one of the shell scripts `sun`, `sgi`, or `dec` in the appropriate directory to create the desired version for your platform.

Sun only: It is strongly recommended to reduce the audio buffer size to the packetization interval. A high number of audio output errors ('p' events in traces) indicate that the buffer size is probably too large. To set the buffer size, you have to become super-user and execute the following Bourne shell script:

```
adb -k -w /vmunix /dev/mem <<EOD
audio_79C30_bsize/X
./W 0xb4
EOD
```

A.2 XView

To enable on-line help for the OpenWindows version, the environment variable `HELPPATH` should be set to include the source directory where the `.info` files are located (here, assumed to be /usr/local/nevot/xview):

```
setenv HELPPATH ${HELPPATH}:/usr/local/nevot/xview
```

If the XView fonts appear too large, the reason is most likely a mismatch between the screen resolution expected by XView and the actual one. Problems occur when running XView applications on DEC and SGI systems. The simple remedy is to make the X server see the 75 dpi Lucida fonts before seeing the 100 dpi fonts. For example, hiding the 100 dpi Lucida fonts and then recreating the font list with `mkfontdir` should fix the size problem.

If you want to use the current XView version rather than the one included with the distribution, the XView libraries and fonts have to be installed before compiling and using NEVOT. The directories used below are typical, but not mandatory. Different directory assignments may have to be reflected in the NEVOT `Makefile`.

1. Obtain and unpack the binary XView distribution for Ultrix from `media-lab.media.mit.edu`, file `xview3-ultrix.4.2-mips.tar.Z`.

2. Obtain the standard XView distribution from wherever the X11 distribution is archived, for example, `prep.ai.mit.edu`. You only need the fonts.

3. Create two font directories, say for a 100 dpi monitor:

   ```
   mkdir /usr/lib/X11/fonts/xview
   mkdir /usr/lib/X11/fonts/xview/100dpi
   mkdir /usr/lib/X11/fonts/xview/misc
   ```

4. Install the fonts from the XView distribution, directories

   ```
   xview3/fonts/bdf/100dpi/*.bdf
   xview3/fonts/bdf/misc/*.bdf
   ```

in the appropriate destination directories, as created in the previous step.

5. Convert the fonts in both directories from `.bdf` to Ultrix `.pcf` format:

```
foreach f (*.bdf)
dxfc -o $f
end
```

6. In both directories, create the necessary `fonts.dir` file by running `dxmkfontdir` in each of the two directories.

7. Notify the X server of the additional font directories:

```
xset fp+ /usr/lib/X11/fonts/xview/100dpi,/usr/lib/X11/fonts/xview/misc
```

   You can check which font directories are used by `xset q`. This setting has to be redone each time you start the server.

8. At this point, you should be able to run an XView application on a system running Open-Windows (e.g., a SPARCstation) and redirect the display to the DECstation. You should also be able to run a pre-compiled Ultrix application. Thus, this step concludes the necessary work if you are not building NEVOT from sources.

9. Install the include files from the XView Ultrix (!) distribution in the appropriate places:

```
mkdir /usr/include/xview
cp xview3/include/xview/*.h /usr/include/xview
mkdir /usr/include/pixrect
cp xview3/include/pixrect/*.h /usr/include/pixrect
```

10. Install the XView libraries and some support files from the XView Ultrix distribution:

```
cp xview3/lib/* /usr/lib
mkdir /usr/lib/help
cp xview3/lib/help/* /usr/lib/help
cp xview3/lib/.[a-z]* /usr/lib
```

   The two important libraries are `libxview.a` and `libolgx.a`.

A.3 Common Problems

Frequent break-ups even on local connections: In the `.sta` file, check the audio underflow count. It should be close to zero; if not and you are using SunOS, make sure that the kernel buffer size has been set properly¶, as described earlier.

G.721/G.723 distorted: Slower machines such as the Sun IPC or ILC may not be able to keep up with G.721 and G.723 encoding or decoding. Monitoring CPU utilization with the perfmeter program should give you a good indication of the resource utilization. Another indication of insufficient CPU cycles is a high audio underflow count despite having set the audio buffer size correctly¶.

References

[1] E. M. Schooler and S. L. Casner, "A packet-switched multimedia conferencing system," *SIGOIS (ACM Special Interest Group on Office Information Systems) Bulletin*, vol. 10, pp. 12–22, Jan. 1989.

[2] H. M. Vin, P. T. Zellweger, D. C. Swinehart, and P. V. Rangan, "Multimedia conferencing in the Etherphone environment," *IEEE Computer*, vol. 24, pp. 69–79, Aug. 1991.

[3] J. DeTreville and D. W. Sincoskie, "A distributed experimental communications system," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1070–1075, Dec. 1983.

[4] D. Cohen, "On packet speech communication," in *Proceedings of the Fifth International Conference on Computer Communications*, (Atlanta, Georgia), pp. 271–274, IEEE, Oct. 1980.

[5] J. W. Forgie, "Voice conferencing in packet networks," in *Conference Record of the International Conference on Communications (ICC)*, (Seatle, WA), pp. 21.3.1–21.3.4, IEEE, June 1980.

[6] S. A. Mahmoud, W.-Y. Chan, J. S. Riordon, and S. E. Aidarous, "An integrated voice/data system for VHF/UHF mobile radio," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1098–1111, Dec. 1983.

[7] N. Shacham, E. J. Craighill, and A. A. Poggio, "Speech transport in packet-radio networks with mobile nodes," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1084–1097, Dec. 1983.

[8] G. Falk, S. J. Groff, W. C. Milliken, M. Nodine, S. Blumenthal, and W. Edmond, "Integration of voice and data in the wideband packet satellite network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1076–1083, Dec. 1983.

[9] D. Cohen, "Specification for the network voice protocol (nvp)," Network Working Group Request for Comment RFC 741, ISI, Jan. 1976.

[10] R. Cole, "Pvp - a packet video protocol," W-Note 28, Information Sciences Institute, University of Southern California, Los Angeles, CA, Aug. 1981.

[11] CCITT, "Draft recommendation G.PVNP: Packetized voice networking protocol," 1989. Appendix 2 to Annex 1 of Question 24/XV (COM XV-1-E).

[12] J. R. Brandsma, A. A. M. L. Bruekers, and J. L. W. Kessels, "Philan: a fiber-optic ring for voice and data," *IEEE Communications Magazine*, vol. 24, pp. 16–22, Dec. 1986.

[13] L. M. Casey, R. C. Dittburner, and N. D. Gamage, "Fxnet: a backbone ring for voice and data," *IEEE Communications Magazine*, vol. 24, pp. 23–28, Dec. 1986.

[14] L. T. Corley, "Bellsouth trial of wideband packet technology," in *Conference Record of the International Conference on Communications (ICC)*, vol. 3, (Atlanta, GA), pp. 1000–1002 (324.2), IEEE, Apr. 1990.

[15] E. M. Schooler, S. L. Casner, and J. Postel, "Multimedia conferencing: Has it come of age?," in *Proceedings of the 24th Hawaii International Conference on System Science*, vol. 3, (Hawaii), pp. 707–716, IEEE, Jan. 1991.

[16] E. M. Schooler, "The connection control protocol: Specification (version 1.1)," technical report, USC/Information Sciences Institute, Marina del Ray, CA, Jan. 1992.

[17] E. M. Schooler, "The connection control protocol: Architecture overview (version 1.0)," technical report, USC/Information Sciences Institute, Marina del Ray, CA, Jan. 1992.

[18] G. Barberis, M. Calabrese, L. Lambarelli, and D. Roffinella, "Coded speech in packet-switched networks: Models and experiments," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1028–1038, Dec. 1983.

[19] A. A. Kapaun, W.-H. F. Leung, G. W. R. Luderer, M. J. Morgan, and A. K. Vaidya, "Wide-band packet access for workstations: integrated voice/data/image services on the Unix PC," in *Proceedings of the Conference on Global Communications (GLOBECOM)*, vol. 3, (Houston, TX), pp. 1439–1441 (40.6), IEEE, Dec. 1986.

[20] P. Spilling and E. Craighill, "Digital voice communications in the packet radio network," in *Conference Record of the International Conference on Communications (ICC)*, (Seattle, WA), pp. 21.4.1–21.4.7, IEEE, June 1980.

[21] H. Miyahara and T. Hasegawa, "Integrated switching with variable frame and packet," in *Conference Record of the International Conference on Communications (ICC)*, vol. 2, (Toronto, Canada), pp. 20.3.1–20.3.5, IEEE, June 1978.

[22] I. Gitman and H. Frank, "Economic analysis of integrated voice and data networks: A case study," *Proceedings of the IEEE*, vol. 66, pp. 1549–1570, Nov. 1978.

[23] C. Topolcic, "ST II," in *First International Workshop on Network and Operating System Support for Digital Audio and Video*, no. TR-90-062 in ICSI Technical Reports, (Berkeley, CA), 1990.

[24] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Trans. Computer Systems*, vol. 8, pp. 85–110, May 1990.

[25] S. Deering, "Host extensions for IP multicasting," Network Working Group Request for Comments RFC 1054, Stanford University, May 1988.

[26] S. Deering, "Host extensions for IP multicasting," Network Working Group Request for Comments RFC 1112, Stanford University, Aug. 1989.

[27] D. E. Comer, *Internetworking with TCP/IP*, vol. 1. Englewood Cliffs, NJ: Prentice Hall, 1991.

[28] S. Casner, J. Lynn, Charles, P. Park, K. Schroder, and C. Topolcic, "Experimental internet stream protocol, version 2 (ST-II)," Tech. Rep. RFC 1190, Network Working Group, Oct. 1990.

[29] A. Papoulis, *Probability, Random Variables, and Stochastic Processes.* New York, NY: McGraw-Hill Book Company, 2nd ed., 1984.

[30] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, pp. 314–329, Aug. 1988. Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[31] N. S. Nayant and P. Noll, *Digital Coding of Waveforms.* Englewood Cliffs, NJ: Prentice Hall, 1984.

[32] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP*, vol. 2. Englewood Cliffs, NJ: Prentice Hall, 1991.

[33] H. A. Chinn, D. K. Gannett, and R. M. Morris, "A new standard volume indicator and reference level," *Bell System Technical Journal*, vol. 19, pp. 94–137, Jan. 1940.

[34] I. H. Merritt, "Providing telephone line access to a packet voice network," Research Report ISI/RR-83-107, Information Sciences Institute (ISI), Marina del Ray, CA, Feb. 1983.

[35] I. N. Bronstein and K. A. Semendjajew, *Taschenbuch der Mathematik.* Thun und Frankfurt/Main: Verlag Harri Deutsch, 19th ed., 1981.