

Sicsophone: A Low-Delay Internet Telephony Tool

Olof Hagsand
LCN Laboratory, IMIT Department
KTH, Royal Institute of Technology
Sweden
olof@kth.se

Ian Marsh
SICS AB
Stockholm
Sweden
ianm@sics.se

Kjell Hanson
Prosilient Software AB
Stockholm
Sweden
kjell@prosilient.com

Abstract

The end to end delay is a critical factor in the perceived quality of service for Voice over IP applications. The described solution is a complete system-level platform called Sicsophone. We describe a VoIP system that couples the low level features of audio hardware with a standard jitter buffer playout algorithm. Using the sound card directly eliminates intermediate buffering as well as providing fine control over timers needed by a soft real-time application such as VoIP. A statistical based approach for inserting packets into audio buffers is used in conjunction with a scheme for inhibiting unnecessary fluctuations in the system. We also present mouth to ear delay measurements for selected VoIP applications and show that several hundreds of milliseconds can be saved by using the techniques described in this paper. A prototype for both UNIX and Windows platforms has been implemented, demonstrating that our system adapts to network conditions whilst maintaining low delays.

Keywords: Packet voice, playout buffer adaption, operating systems

1 Introduction

Users of interactive VoIP applications demand low latency conversations. Replaying packetised audio requires that sufficient packets are available to the application in order to avoid gaps or glitches. The digital to analog conversion of sampled voice requires strict, synchronous timing despite the fact that the network and operating system may disrupt the process. The most common method to solve this problem is to introduce a small intermediary buffer between the decoded audio stream and the audio hardware which allows packets to be “available” for playout. Of course withholding packets instead of immediately playing them increases the total delay of a VoIP application. However,

the longer packets can be delayed, the more resilient the receiver is to adverse network conditions. We should point out that is a working implementation and the algorithmic complexity is an important factor. We motivate this approach with real delay experiments and results. Hence the goal is not to compare the merits of various playout algorithms, this has been covered by many researchers, rather to give some insight what issues are important when realising these schemes and their effects.

In this paper we refer to the mouth to ear delay as the total one way delay experienced by two speakers including the analog-digital-analog conversion. By jitter we mean the variability in the packet delay. This variability is the reason we need to buffer packets, thus our work focuses on how to detect and compensate for packet jitter in an efficient manner. Our solution is to insert packets into the memory of sound cards relieving the need for data copying or context switching. This approach saves precious time, avoids scheduling problems but requires careful buffer management.

Figure 1 illustrates the complete path of audio samples from a microphone at a sender to the loudspeaker at a receiver. Traditionally, a sender writes voice samples to the operating system which are subsequently sent across the network to a receiving host. At the receiver, data is read from the operating system interface where it is the responsibility of the application to adjust the buffer size as required, this is shown by the solid lines in the illustration.

In our approach we use the buffering scheme in the operating system and copy the packets directly into the memory of the sound card. Therefore, we save copying the data to and from the application, plus not performing the de-jittering in the application. We describe our approach in the context of DirectSound [1] on the Windows platform. It is important to point out that our approach is not confined to this architecture, a ring buffer with pointer support is sufficient to realise the ideas presented in this paper (alternatives for UNIX include [8] or [7]). However we describe the system using DirectSound as it is known to many developers

and was used in our experimental evaluation. It is important to state that we assume the systems are not under heavy load or consider Sicsophone as a hard real-time system.



Figure 1. Audio Delivery Path within Sicsophone

If we now look at the steps a receiver must take to replay packetised audio from the network in more detail, Table 1 shows four such typical steps. Firstly, de-packetisation, removes the IP and UDP headers and passes the datagram together with a Real Time Protocol (RTP) payload to a VoIP application. This step takes a few milliseconds on most systems. Step two is to decode the sound samples, this is dependent on the compression scheme as well as the packet size used. Typically this takes from a few milliseconds to tens of milliseconds. Steps three and four are usually performed as distinct steps, absorption of network delays through buffering and delivery to the sound application. Our goal was to consolidate these steps into a single step, saving the time of inter-mediatory buffering and context switching. We refer to this approach, solely for definition by its software name, Sicsophone.

Table 1. Typical Receiver Incurred Delays (ms)

Step	Process	Overhead	Depends On
1	De-packetisation	10 - 50	Pact. Size
2	Decoding	10 - 50	Coding
3	Buffer Delay	5 - 200	Network
4	Delivery	5 - 120	End System

The remainder of this paper is organised in the following fashion; Section 2 forms the main body of this work, low level adaption of playout buffers using ring buffers. Section 3 presents results of Sicsophone’s performance of mouth to ear results for different VoIP tools. We also give comparisons of the playout delay with Sicsophone against the idealised case. Section 4 is a description of related efforts with which this paper has commonalities, we round off the paper with some conclusions in Section 5.

2 End-system Adaption to Jitter

2.1 Buffering Issues

In this section we outline some issues associated with the data buffering scheme we have chosen. Our goal is to save time by avoiding data copying, setting up direct memory access (DMA) transfers ahead of time, using simple data structures and inserting de-jittered audio packets directly into the memory of the sound card. Using the sound card as a buffer has the advantage of not adding any extra buffering to the audio sample path. It also avoids copying data from a kernel to an application and back again. Direct memory access is used to move data from memory to the sound card and vice versa without intervention of the CPU. Using DMA efficiently is not trivial, as it can take some time to set up the transfer. However once it is done, the transfer can be done much quicker and more efficiently. This offers significant time savings over posting an interrupt for every packet, particularly in the older (and non-DirectX) versions of the Windows operating systems.

One potential problem of using the sound card memory as a buffer is it could be overrun by packets arriving too quickly, for example on a fast connection. Modern sound cards however are equipped with megabytes of RAM to store down-loadable sound samples, DirectSound can allocate buffers up to this physical size when a hardware buffer is initialised. Another potential cause of overrun or under-run is misaligned or drifting clocks, there is no mechanism in Sicsophone to explicitly detect this. We do however keep the buffer from being overrun by mechanisms explained in Section 2.3.

Another important but often overlooked issue is mixing. For an application like VoIP where the voice channel is stopped and started continuously, we would like to minimise this setup time. Valuable time can be lost by setting up mixers for software and hardware buffers where we normally do not want to mix audio from different sources, e.g. the VoIP and a MP3 song. Therefore we allocate a DirectSound primary buffer to give better delay characteristics, as it does not need to be mixed before outputting to the D/A conversion.

The coding scheme used is another issue. Packets have to be decoded before insertion into the buffer if they are not in PCM format. However using PCM allows us to DMA the payload into the sound card memory without any audio format conversion. This is the *fastest* path from packet reception to playout. It is however possible to support other audio formats, however they require extra CPU cycles for decoding the audio, and a small buffer to hold the data before and after decoding. A limitation of this approach is the coupling between packet size and time, we assume a certain number of bytes in the buffer corresponds to a well defined

playout time.

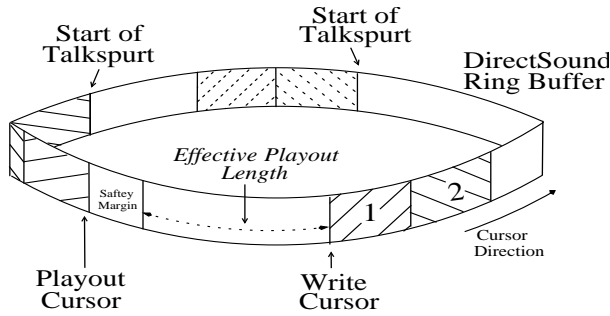


Figure 2. DirectSound Buffer Structure

Figure 2 shows the interface offered by DirectSound. Data is written at the write cursor and replayed by the trailing read pointer. The read and write pointers are updated by the system, and continuously encircle the buffer. Reading and writing the pointers requires that the system almost instantaneously updates their current positions. Some of the older operating systems used in our measurements did not give the fine granularity over the positions of the timers. In Sicsophone they are used as **both** timers and pointers. In fact we use the DMA producing and consuming data as the only clocks in the delivery system. They function as a timer by indicating if a packet is too late. If the read pointer has already passed the point where a packet should be, and it has not been written, then we know that this packet is late. Insertion is simply a modulo operation and a pointer copy.

In order not to replay old data from the buffer when no packets are being sent we write “empty” samples that sound like audible background noise into the buffer so that the listener is aware the connection is still open.

Used as pointers, the read and write cursors give memory locations where data is read or written to depending on the operation to be performed. Given these pointers it is easy to adjust the buffer length, it is simply where packets are chosen to be read from. The closer the read pointer is to the write pointer the smaller the effective buffer length will be. Note there is a small margin of 15ms in front of the read pointer to allow data that has been written to be “ready” for playback. Use of this safety margin is recommended by the developers of DirectSound.

To give a concrete example, using an estimate of the network delay and its variation, described previously, we insert packets at a specified “distance” ahead of the read pointer. Therefore a translation from milliseconds to bytes is needed; $bytes = (samples_sec \cdot bits_sample \cdot P_i) / 8000$. For example, if one substitutes 8000 for samples_sec, 8 bits per sample and 200ms for the playout point this equals 1600 bytes. This means that the write cursor can simply be set 1600 bytes in front of the read cursor. The safety margin is also included in the length of playout buffer

but it is possible to simply subtract the value (120 bytes in this case) from the calculation. To re-iterate once the playout point has been calculated it is trivial to insert packets into the buffer, no complex data operations are needed.

2.2 Fast Startup Adaption

In an adaptive VoIP application we normally consider changing the buffer size during a silence period so as not to introduce audible glitches in the analog audio stream. Since the goal of this work is to produce a low-level VoIP tool we would like to keep the buffer length close to optimal. However in the startup phase we have no idea of the network condition and therefore have to use default values for the network delay (Sicsophone uses min = 20ms, max = 60ms). We therefore adjust the buffer length after monitoring only a few packets to settle to an estimate quickly.

Figures 3 and 4 show packet delay in the jitter buffer during the start up phase of Sicsophone as an example. The y-axis shows the waiting time in the buffer (in ms) and the x-axis shows the number of packets received, sorted by the time spent in the buffer, note this is **not** the sequence number. It shows the number of packets and their respective waiting times.

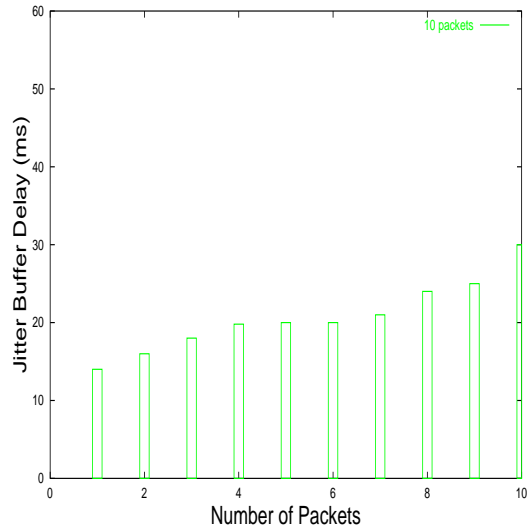


Figure 3. Jitter for the first ten packets

Figure 3 shows the buffer state after ten packets have been received and Figure 4 after an additional 40 packets have arrived, the original ten are shown with bolder lines. After ten packets were stored, the time spent in the buffer varied between 14 and 30 ms whereas after 50 packets the median delay incurred is around 20 ms. This is not surprising as packets are sent with a 20 millisecond separation.

Fast adaption is worthwhile during the start up phase of a VoIP session. The alternative approach is to be conser-

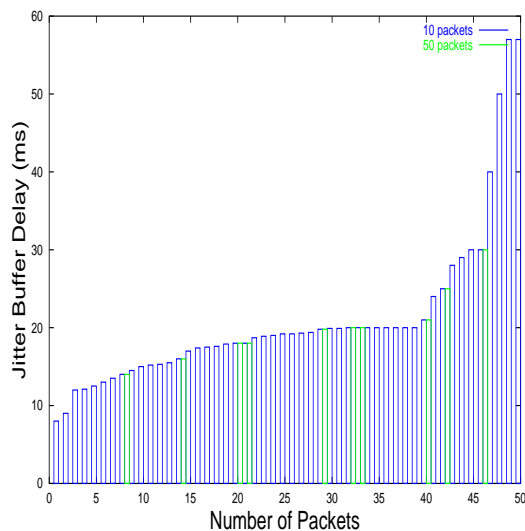


Figure 4. Arrival of 10 and 50 packets

vative in the start up phase and have long playout buffers until a value for the playout point can be calculated. In the presence of spikes [6] we can re-estimate the jitter value quickly. Since the goal of this paper is a low delay VoIP tool we chose to adapt quickly. Furthermore, usually there are sufficient silence periods during the startup phase of a conversation to perform fast adaption. In the case where there are none (such as call waiting, i.e. music playing) we adjust the buffer when a packet is excessively delayed or if there is a loss, failing these possibilities we adjust the buffer length and tolerate an audio glitch.

2.3 Bounding the Estimated Network Delay

Sudden increases in the network delay¹ can cause VoIP applications problems. A spike is referred to a sudden and rapid increase in the network delay which is typically short lived, often less than one round trip time. One solution is to follow the increase in the delay and adjust the buffer length accordingly. The alternative is to include the values of the jitter estimate but not adapt the buffer size. It is because of this temporal property plus we are dealing with real hardware, has led us to be more conservative to network conditions and not to adapt the DirectSound buffer length to sudden increases in network delay.

We have implemented a system which bounds the delay jitter estimate. As stated, the spikes are not completely ignored but we do not react immediately to their presence. The estimated jitter value should vary between an upper Q_{max_i} and a lower Q_{min_i} (see the key in Figure 5) bound in a “corridor”, where $Q_{min_i} < d_i < Q_{max_i}$.

¹referred to as spikes in [6]

If the running estimate breaks either of the boundaries we re-calculate the new buffer length, taking into account the value of the spike, but reset the mean estimate to the middle value of this new (Q_{min_i}, Q_{max_i}) . Figure 5 shows an example of a receiver jitter buffer during a conversation between two machines on a local network. The y-axis

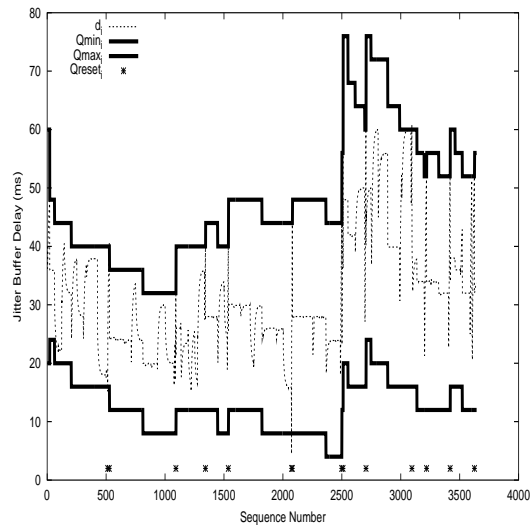


Figure 5. Jitter Buffer Delays with Bounded Mean

shows the jitter buffer length and the x-axis the sequence number. The system starts with Q_{min} and Q_{max} set to the default values 20ms and 60ms for the minimum and maximum bounds respectively. At the bottom of the figure we show the corridor breaks as stars to highlight the breaks. We have found this scheme to work well, in the given trace there were only 14 breaches of the corridor from over 3600 packets (less than 0.5%). More importantly we did not make costly, unnecessary changes to the DirectSound primary buffer.

3 Results

We divide this section into two sections, the first gives the total delay of popular VoIP tools compared to Sicsophone in a laboratory environment, with basically no, or little, network delay. Secondly, we show the performance of the playout algorithm using trace files taken over the Internet (two of the ones used in [6]). This shows a typical WAN component due to the jitter on the Internet plus the best possible playout that could have been achieved by post-processing trace files. We chose to give the results in this manner to estimate the real mouth to ear delay by including both the WAN and LAN components. It can be seen

as a sum of these two quantities. Essentially the first set includes the delay due to coping with the operating system and the second with the network conditions. Using these trace files we can compare our implementation with those published. We must point out these trace files are old so we have taken much more recent ones and describe this a little in the third subsection of the results.

3.1 Mouth to Ear Measurements

The delay contributed by the end systems is the main result of this paper. We performed one-way mouth to ear measurements with a range of VoIP tools and the results are summarised in Table 2. It’s important to state that no parameter tweaking of these tools was done, we used their default installation values. The experimental setup used was as shown in Figure 1. The measurements were done us-

Audio Tool	Latency (ms)
Sicsophone prototype	25-100
Vocal Internet Phone 4.5 (SB)	450-550
Vocal Internet Phone 4.5 (PJ)	580-620
NetMeeting 2.1 (SB)	620
NetMeeting 2.1 (PJ)	750
VAT 3.4 (Solaris)	1200
RAT 3 (Solaris)	1500

Table 2. Mouth to Ear Latency Measurements (SB SoundBlaster and PJ PhoneJack)

ing a signal generator feeding a sender and an oscilloscope to measure the time difference between the sender and receiver.

We can see that there are large variations between the various applications. One important result of this paper is to highlight the design of end systems for VoIP applications. Our goal is not simply to state Sicsophone is superior to other tools, rather to show the considerable time savings, 10’s to 100’s of milliseconds, can be saved by using the approach described.

3.2 Comparison with Ideal Playout Conditions

In the introduction we mentioned that a jitter buffer playout algorithm essentially has to tradeoff low delay or packet loss due to their arrivals being too late. Low delay implies a short playout buffer, incurring higher packet loss due to late arrivals. When comparing the performance of algorithms it makes sense, therefore, to consider loss and delay.

As stated in the introduction to this section, we include these figures to give an estimation of the delay incurred by the buffering we employ. They should be taken as the delays

incurred for a given trace at a particular time and are used to show the *network* delay which would be added to the *system* delay, i.e. those in in Table 2.

Figures 6 and 7 show the results for two Internet trace files². To calculate the optimal playout point we order all the packets and remove the 1% with the highest delay. We then calculate the delay needed to play the remaining 99% of the packets resulting in the delay for 1% packet loss, this process is repeated up to 25% packet loss (although in practice more than 10% would be deemed unacceptable). Studies have shown that 1% is acceptable without packet loss concealment and up to 10% with packet loss concealment [3]. Packet loss concealment is the process of “filling in” missing or lost packets which has been shown to be more preferable than glitches in the voice. Figures 6 and 7 show Sicsophone’s playout delay performance in comparison.

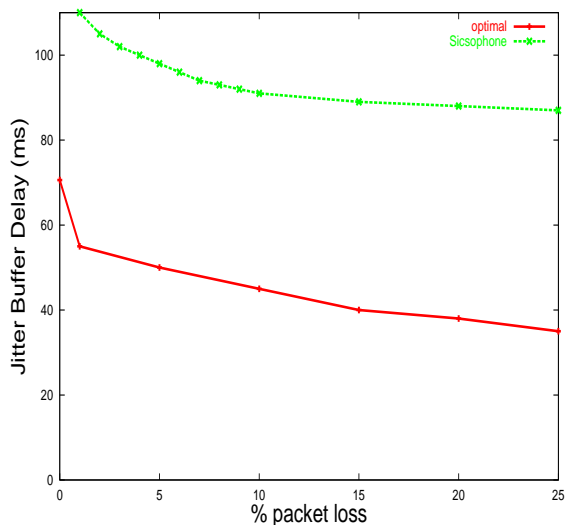


Figure 6. Playout Delays for a Trace from UCI, California to INRIA in France

In Figure 6 Sicsophone is about 50ms from the ideal playout point and remains more or less constant as the packet loss increases. For a given loss rate, e.g. 5% Pinto and Christensen ([5]) quote a slightly lower delay than we do, 72ms compared to our 98ms and similarly so for other loss rates. We should re-iterate the focus of this paper is the implementation and the absolute/measurable delays rather than the playout algorithm itself. Nevertheless, the result in this case is due to the large variation of jitter ($\pm 20ms$), which makes it hard to settle to a constant value for the minimum buffer length, this can be verified by looking at the absolute jitter. The ITU G.114 document recommends that the one-way delay should not exceed 150ms, so this is about one third of the recommended delay used in the

²<http://gaia.cs.umass.edu/~sbmoon/traces.html>

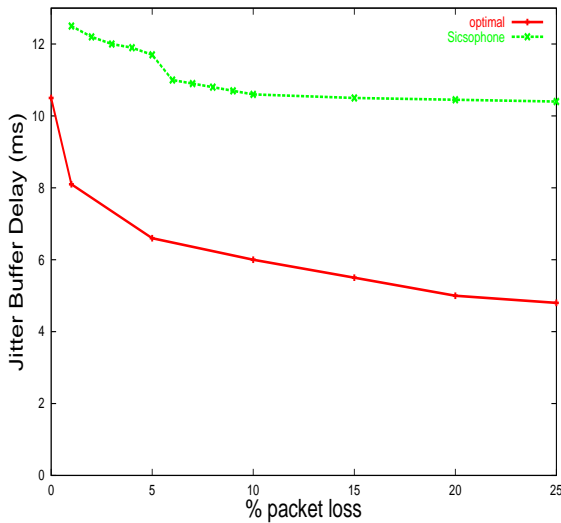


Figure 7. Playout Delays for a Trace from Amherst, Mass to GMD, Berlin

buffer playout algorithm. A second test is shown in Figure 7 which shows a trace from the University of Massachusetts in Amherst to GMD in Berlin. In this case the jitter is much better and the difference between the optimal and Sicsophone is only 5ms. We have included both the worst and best test cases of the five traces available. The traces are seven years old so we took some new ones using Sicsophone invoked automatically by scripts to obtain more and newer data for test traces, this is described next.

4 Related Work

The early 90’s produced a surge in packet audio playout research. One of the first efforts to implement a voice application on an IP network with an adaptive buffer playout strategy was NeVoT [10]. The playout algorithm implemented is almost identical to NeVoT [10]. They use a variation estimate similar to the one given earlier, however they make a slight distinction for the first packet in a talkspurt and subsequent ones. The playout for the first packet is delayed longer due to lack of information on the network state after the silence period. Our work shares theirs in the choice of a ring buffer for buffering packets, only we perform the copying by using DMA transfers directly rather than copying the data from the application to the operating system. Using a ring buffer in Sicsophone is identical to that described in [10] where the authors motivate their choice of using a circular buffer for performance reasons. VAT (Visual Audio Tool) [2] is a well known VoIP tool that implements a playout buffer similar to the one described, including a circular buffer to hold the packets before play-

out. We use an additional scheme to prevent the jitter estimates from varying too rapidly plus focus on the efficient insertion of packets into the playout buffer. Moon *et al.* [6] present four different playout algorithms for packet audio. All calculate an estimate of the network delay and jitter as an average from all the packets measured. The authors study jitter spikes in traces and also do not adapt the buffer size to these spikes. Pinto and Christensen [5] describe an algorithm for jitter compensation based on the target packet loss rate. Their “gap based” approach compares the current playout time with the arrival time and calculate a gap for both early and late packets. They compare the current playout delay, for any particular talkspurt in progress, with an optimal playout delay. This optimal theoretical delay is defined as minimum amount of delay to be added to the creation time of each packet which would result in a playout of a talkspurt at the given loss rate. Our calculation of the optimal playout is similar to the one described in this paper. Luigi Rizzo describes a generic sound card driver for FreeBSD [8]. Aspects of it resemble our work, in particular, handling of timers, DMA transfer and buffer size allocation. They include hooks to use the driver for VoIP applications, one such example is a select() call which can be scheduled to return only when a certain amount of data is ready for consumption. As stated earlier, it is possible to implement Sicsophone on this device driver. [9] looked at combining target-based playout algorithms in conjunction with FEC schemes, and propose a number of new playout algorithms based on this coupling. [4] keep the flow of audio constant during operating system load by using buffering in the audio hardware. They also look at reducing the amount of buffering in the application by keeping the buffers in the application as small as possible. In our case we try and totally eliminate it by only using the hardware buffers.

5 Conclusions

In this paper we have shown how careful buffer management combined with a simple statistical playout scheme can reduce mouth to ear delay for VoIP applications. As stated at the start of this paper, delay is one of the most important factors in the perceived QoS and this has been the focus of this work. The results are encouraging as the mouth to ear delay of Sicsophone on a LAN is around 50ms on a Windows NT system with DirectX 8.0.

We also include an estimate of the delay induced by network conditions using the playout algorithm, but include it to help estimate the network delay of VoIP tool such as Sicsophone. We have proposed a system which tries to reduce the perceived mouth-to-ear delay of real-time packet audio communication.

It is well known that users are sensitive to delay, however we are not aware of any studies that have been conducted

on the effect of dynamically *changing* the perceived delay (even reducing) by techniques such as the one suggested in this work. If this is found to be important then one could consider it in the design of the algorithm i.e. by inhibiting too frequent changes in the playout delay.

Future work in this direction is to measure the delay with operating systems such as Windows XP (with DirectX 9.0) and UNIX platforms. We have gathered more than 25,000 VoIP trace files from around the world using Sicsophone and plan to use them for further jitter, loss and delay analysis.

References

- [1] B. Bargen and P. Donnelly. *Inside DirectX*. Microsoft Press, 1998.
- [2] V. Jacobson and S. McCanne. vat - LBNL audio conferencing tool, July 1992. Available at <http://www-nrg.ee.lbl.gov/vat/>.
- [3] J. Janssen, D. D. Vleeschauwer, and G. H. Petit. Delay and distortion bounds for packetized voice calls of traditional PSTN quality. In *Proceedings of the 1st IP Telephony Workshop (IPTel 2000)*, pages 105–110, Berlin, Germany, Apr. 2000. GMD Report 95.
- [4] I. Kouvelas and V. Hardman. Overcoming workstation scheduling problems in a real-time audio tool. In *Proc. of Usenix Winter Conference*, Anaheim, California, Jan. 1997.
- [5] J. Pinto and K. Christensen. An algorithm for playout of packet voice based on adaptive adjustment of talkspurt silence periods. In *Proceedings of the IEEE 24th Conference on Local Computer Networks*, pages 224–231. ACM, Oct. 1999.
- [6] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 680–688, Toronto, Canada, June 1994. IEEE Computer Society Press, Los Alamitos, California.
- [7] D. Reed. A new audio device driver abstraction. In *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1998.
- [8] L. Rizzo. The FreeBSD audio driver. *Lecture Notes in Computer Science*, 1356, 1997.
- [9] J. Rosenberg, L. Qiu, and H. Schulzrinne. Integrating packet FEC into adaptive voice playout buffer algorithms on the internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel Aviv, Israel, Mar. 2000.
- [10] H. Schulzrinne. Voice communication across the Internet: A network voice terminal. Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.