PERFORMANCE EVALUATION OF AN IP VOICE TERMINAL

Harri Marjamäki & Raimo Kantola
Harri Marjamäki, M.Sc.
Raimo Kantola, Ph.D., Professor of Telecommunications Technology

Key words: Voice over IP, Playout algorithms, Performance evaluation

Abstract: This paper examines the issues related to the transmission of voice over packet networks using the Internet Protocol (IP). We focus on studying the delays that are generated in the terminal, which is a Unix workstation equipped with IP voice application software.

Delay components in the terminal are presented. We measure the processing delays in the terminal using different audio codecs and measure the end-to-end delays using different scheduling parameters for the IP voice application. A significant part of the delay is shown to be caused by buffering at the receiving terminal. This is a feature of the application software and can be removed by modifying the source code.

We also make a comparison of adaptive algorithms that are used to calculate the playout times of the voice packets. Algorithms are simulated using different network loads and thus different delay distributions of the voice packets in an Ethernet. We present a new playout algorithm, which is a combination of two existing algorithms. This algorithm is shown to outperform the other two real-time algorithms that are compared in our studies.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35581-8_35

T. Yongchareon et al. (eds.), *Intelligence in Networks*

1. INTRODUCTION

The Internet provides a simple single class best effort service. From a connection's point of view, the best effort service amounts in practice to offering a channel with time-varying characteristics such as delay and loss distributions. These characteristics are not known in advance since they depend on the behaviour of other connections throughout the network. A variety of audio tools have been available for a few years, and they have been used to audiocast conferences. Experimental evidence suggests that, although the quality of the audio delivered by Internet tools has improved, audio quality is still mediocre in many audio conferences. This is clearly a concern since audio quality has been found to be more important than video quality or audio/video synchronization to succesfully carry out collaborative work.

For audio quality in packet audio applications, the main concerns are the delay and delay variance. In earlier studies [1], [2] it was noted that in LANs and campus networks where network caused delay and delay variance was relatively small, most of the end-to-end delay was accumulated in the terminals.

In the terminal, delay is accumulated both by the hardware and the software. In the audio hardware, voice samples are A/D converted at the sender and D/A converted at the receiver. In the packet audio application software processing delay is introduced. Processing delay is very much dependent on the used speech codec. Some codecs, like PCM codec have very little to do and introduce very little delay whereas, for example, a GSM codec requires excessive computation and causes significantly more processing delay. Buffering of voice samples is necessary both at sending and receiving end. Buffering delays are introduced both in the audio hardware and in the packet audio software. Delays are introduced also by the operating system because it has to assign processor time also to other processes that are simultaneously running in the terminal.

The network caused delay variance has to be smoothed in the application software in order to preserve the sound quality. Voice packets are buffered at the receiver and they are played out periodically. The algorithms used to calculate the appropriate playout time for each packet of voice are called playout algorithms.

The rest of the paper is organized as follows: Chapter two explains playout delay adaptation: the mathematics and algorithms used in delay estimation. Chapter three reports performed measurements and obtained results. Chapter four presents conclusions and topics for future work.

2. PLAYOUT DELAY ADJUSTMENT

Packet audio tools operate by periodically gathering audio samples generated at the sending host, packetizing them, and transmitting the resulting packet to receiving site(s). For efficiency, the source audio is typically divided into talkspurts (periods of audio activity) and silence periods (periods of audio inactivity, during which no audio packets are generated). In order to faithfully reconstruct the audio at the receiving site, data in packets within a talkspurt must be played out in the same periodic manner in which packets were generated.

If the underlying network is free of variations (jitter) in packet delays, a receiving site can simply play out an audio packet as soon as it is received. However, jitter-free, in-order, on-time packet delivery rarely, if ever, occurs in today's packet-switched networks. In order to compensate for these variable delays, a smoothing buffer is thus typically used at a receiver. Received packets are first queued into the smoothing buffer and the periodic playout of packets within a talkspurt is delayed for some amount of time beyond the reception of the first packet in the talkspurt. We refer to this delay as the playout delay of the talkspurt. Clearly, the longer the playout delay, the more likely it is that a packet will have arrived before its scheduled playout time. Excessively long playout delays, however, can significantly impair human conversations. There is thus a critical tradeoff between the length of playout delay and the amount of loss (due to late packet arrival) that is incurred. Generally, delays between talkspurt generation and receiver playout of less than 400 ms [3] and a loss percentage of up to 5% [4] are considered to be quite tolerable in human conversations.

The talkspurt playout delays themselves can be either fixed for the duration of the audio session, or adaptively adjusted. In the Internet, end-to-end delays fluctuate significantly and a constant, non-adaptive, playout delay would thus likely yield unsatisfactory audio quality for interactive audio applications. There are two approaches for adaptive playout adjustment: per-talkspurt and per-packet adjustment. The former uses the same playout delay throughout a talkspurt (and, as a result, faithfully reconstructs the original periodic nature of the received audio data from the sender), but allows different playout delays from one talkspurt to another. While this may result in artificially elongated or compressed silence periods, this is not noticeable in played out speech if the change is reasonably small [5]. In the latter approach, the playout delay varies from packet to packet. A per-packet adaptive adjustment introduces gaps inside talkspurts and is cited as damaging to the audio quality [6].

2.1 End-to-end delay characteristics

Previous studies [7] have indicated the presence of "spikes" in end-to-end Internet delays. A spike constitutes a sudden, large increase in the end-to-end network delay, followed by a series of packets arriving almost simultaneously, leading to the completion of the spike.

With periodically generated packets, the initial steep rise in the delay spike and the linear, monotonic decrease after the initial rise, is due to "probe compression" – the accumulation of a number of packets from the connection under consideration (the audio session, in our case) in a router queue behind a large number of packets from other sources. Probe compression is a plausible conjecture about the cause(s) of delay spikes.

2.2 Performance of a playout algorithm

The tradeoff between the average playout delay and loss due to late packet arrival is used as the performance measure in comparing one adaptive playout delay adjustment algorithm with another. Loss and delay are considered on a per-packet rather than per-talkspurt basis for two reasons. First we note that the lengths of talkspurts depend on silence detection algorithms and their parameters. Per-talksurt results are thus closely tied to the silence detection algorithm used. More importantly, different talkspurts have different lengths.

Here the end-to-end application-to-application delay is defined to be the difference between the playout time at the receiver and the generation time at the sender. We refer to Figure 1 to show the timing information of audio packets and formally define the average playout delay [9].

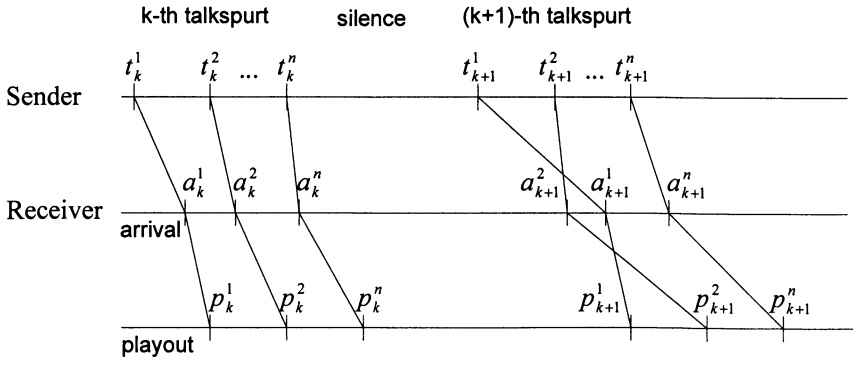

Figure 1. Timings associated with the i-th packet in the k-th talkspurt

Consider a trace consisting of M talkspurts. We define the following quantities:

- t_k^i : sender timestamp of the i-th packet in the k-th talkspurt
- a_k^i : receiver timestamp of the i-th packet in the k-th talkspurt
- n_k : number of packets in the k-th talkspurt. Here we only consider those packets actually received at the receiver.
- N: total number of packets in a trace,

$$N = \sum_{k=1}^{M} n_k \tag{1}$$

The playout time of a packet depends on which algorithm is used at the receiver to estimate the playout delay of the packet. Consider a playout algorithm A. Then $p_k^i(A)$ is the playout timestamp of the i-th packet in the k-th talkspurt under A. If the i-th packet of the k-th talkspurt arrives later than $p_k^i(A)$ (i.e., $p_k^i(A) < a_k^i$), it is considered lost. Otherwise, it is played out with the playout delay of ($p_k^i(A) - t_k^i$). Let $r_k^i(A)$ be an indicator variable for whether the i-th packet of the k-th talkspurt arrives before its playout time, as computed by playout algorithm A:

$$r_k^i(A) = \begin{cases} 0, p_k^i(A) < a_k^i \\ 1, otherwise \end{cases} \tag{2}$$

The total number of packets played under algorithm A is denoted as $N(A)$ and computed using $r_k^i(A)$:

$$N(A) = \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A) \tag{3}$$

Then the average playout delay of those played-out packets is defined as:

$$\frac{1}{N(A)} \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A)(p_k^i(A) - t_k^i) \tag{4}$$

If there are N packets in a trace and, among them, $N(A)$ packets are played out under algorithm A, the loss percentage l is:

$$l = \frac{N - N(A)}{N} * 100 \tag{5}$$

2.3 Some playout algorithms

In this section we present 4 different playout algorithms. Algorithms 1 and 2 are originally reported in [8] and algorithm 3 is suggested in [9]. Algorithm 4 is a combination of algorithms 1 and 3. The mathematical descriptions of the algorithms are not presented because of the lack of space.

2.3.1 Algorithm 1

This algorithm is based on stochastic gradient algorithms used in estimation and control theory [5], and operates by estimating two statistics characterizing the network delay incurred by audio packets: the delay itself, and a variational measure of the observed delays. Each of these estimates is recomputed each time a new packet arrives.

Algorithm 1 is a linear filter that is slow in catching up with a change in delays, but is good at maintaining a steady value, when the gain of the estimator (1-α), is set to be very low.

2.3.2 Algorithm 2

Algorithm 2 has two modes of operation, depending on whether a spike has been detected. In normal mode, it operates like algorithm 1 with a different gain, but in spike-detection mode, the playout delay is updated differently.

Algorithm 2 works as follows: It is checked if the delay between consecutive packets at the receiver is large enough for it to be called a spike. On detection of a spike, we enter the spike mode and "follow" the spike. Thus, in the spike mode, we allow our estimate to be dictated only by the most recently observed delay values.

2.3.3 Algorithm 3

The key idea behind this algorithm is to collect statistics on packets that have already arrived and to use them to estimate the playout delay. Instead of using the linear filter mechanism, each packet's delay is logged and the distribution of packet delays is updated at every packet arrival. When a new talkspurt starts, the algorithm calculates a given percentile point q in the distribution function of the packet delays for the last w packets, and uses it as the playout delay for the new talkspurt. As in algorithm 2, it detects spikes and behaves accordingly: once a spike is detected, it stops collecting packet delays and follows the spike until it detects the end of a spike. Upon detecting the end of a delay spike, it resumes its normal operation.

2.3.4 Algorithm 4

With algorithm 3, the problem is that it first needs to collect some delay statistics. In a real-time implementation it is not possible to use algorithm 3 from the beginning of a call. Therefore we can combine together algorithms 1 and 3 and the resulting algorithm is called here algorithm 4. The idea is

simply to calculate the playout delays with algorithm 1 until we have received 5000 voice packets and then switch to algorithm 3. The switch can't be done earlier because then algorithm 3 would probably give worse results than algorithm 1. The extension to algorithm 1 is that we use the two operating modes from algorithm 3 since the beginning of the call.

3. MEASUREMENTS AND RESULTS

In this chapter we present the results of our experiments. The presentation is divided in three parts. First part handles the processor time consumption in an IP voice terminal, the second part handles the end-to-end delays in an IP voice connection, and in the third part different playout algorithms are compared. The used VoIP client in all experiments is Nevot (Network Voice Terminal) [10]. Nevot was chosen because of its high configurability compared with other VoIP tools. Nevot provides for example a feature to switch off the additional playout delay for testing purposes. Nevot's source codes are freely downloadable in the Internet, which made it possible to compile it from sources and examine the program implementation in practise. Thirdly, Nevot provides a debugging option to record sent and received RTP-headers which is useful for playout algorithm simulations.

3.1 Measurement of the CPU consumption

We wanted to know how much processor time is consumed by Nevot using different audio codecs. In order to do this, version 3.35 of Nevot was compiled from sources using -p flag. This enables us after running the program by using the prof command to produce a profile file which shows for each external text symbol the number of times that function was called and the average amount of time per call.

Nevot provides the following speech codecs: PCM 64 kbit/s, ADPCM 32 kbit/s, GSM 13 kbit/s and LPC 4,8 kbit/s. PCM and ADPCM are computationally light codecs compared to GSM and LPC, which are computationally rather intensive.

3.1.1 Setup of the measurement

We had two Sun Ultra Enterprise 1 workstations connected to a 10BaseT Ethernet. Both workstations were running Nevot version 3.35 for 5 minutes in each measurement. Packet sizes were set to 20 ms and silence detection

was disabled so that both clients were continuously sending and receiving packets.

3.1.2 Results of the measurements

Consumption of CPU time for different codecs is presented in table 1. These results show that the processing delay increases with the complexity of the codec. The values in Table 1 include both coding and decoding in one workstation.

Table 1. Used CPU time in milliseconds and percent of used CPU time from total CPU time with different audio codings

Used CPU time	*PCM*	*ADPCM*	*GSM*	*LPC*
in milliseconds	0.26	0.49	1.14	1.83
in percent	1.3	2.5	5.7	9.1

3.2 Measurement of the end-to-end delay

In this measurement we had two Sun Ultra Enterprise 1 workstations with SunOS 5.5.1 connected to a 10BaseT Ethernet. Both workstations were running Nevot version 3.35. Delays were measured with four different audio codings. For each audio coding both half-duplex and full-duplex traffic was measured. We also used two different process priorities for Nevot to see if the operating system has some contribution to the delay. Used priorities were normal time-shared class priority with user priority 0 and realtime priority with the highest possible priority 59 and time slice of 1 second. Each type of measurement was repeated 10 times. Packet size was set to 20 ms in all measurements.

3.2.1 Results of the measurements

Delays were measured 10 times for each configuration. Table 2 presents the averaged results.

Table 2. Average end-to-end delays in different audio codings

	TS, half duplex [ms]	*TS, full duplex [ms]*	*RT ,half duplex [ms]*	*RT, full duplex [ms]*
PCM	32.7	32.7	31.3	31.4
ADPCM	33.4	34.9	31.6	32.0
GSM	34.7	33.5	32.5	33.6
LPC	35.0	36.4	33.0	35.2

3.2.2 Analysis of the measurements

Measurements were done over a non-loaded LAN and network caused delay was measured with Ping and was shown to be constantly around 0.5 ms. The framing delay of 20 ms is included in all results.

Hardware caused delays were measured by directing the signal from microphone input to headphones output by Audio tool program. Thus, a 1 kHz square-wave input signal was A/D converted and then D/A converted in the audio hardware. Input signal was connected to the first channel of an oscilloscope and output signal to the second channel of the oscilloscope. HW delay was measured several times and was constantly 2.2 ms.

If we take for example full duplex measurement with real time priorities and PCM audio coding and subtract 31.4 ms - 20 ms (framing delay) - 2*0.26 ms (processing delay) – 0.5 ms (network delay)- 2.2 (HW delay) we are left with 8.2 milliseconds. This delay is caused by buffering. Operating system can increase the delay if Nevot is run as a time-sharing class process because then it's not guaranteed to be scheduled at predicted time intervals. In these measurements it varied between 0 and 2.9 milliseconds. Components of the end-to-end delay are illustrated in Table 3.

Table 3. Components of the end-to-end delay

Delay component	*Delay in ms*
Framing delay	20.0
Processing delay	0.5 – 3.7
HW delay	2.2
Network delay	0.5
OS delay	0 – 2.9
Buffering delay	8.2 – 8.8

An interesting thing was noted in the end-to-end delay behaviour with Nevot. When a new talkspurt is started, the delay is first around 30 ms as shown earlier, but within two seconds it increases by 40 ms and after about 10 seconds it increases by another 20 ms. Similar behaviour was noted in [1]. This was found out to be caused by the Nevot that was receiving packets. The reason is just to avoid buffer underflows in case of late packets.

We removed this feature by modifying the source code. No reduction to sound quality caused by this change was noticed in our informal subjective listening tests. The playout delays in Nevot are calculated with algorithm 1, which was presented in section 2.3. After the correction no additional playout delays are presented.

3.3 Comparison of playout algorithms

In this section we compare the performance of the playout algorithms that were presented in Section 2.3. The performance metric we use to compare different playout algorithms is the average playout delay vs. loss percentage. To evaluate algorithms 1-4, we generate some traces, which include the sender and received timestamps of each packet from a trace. Using Matlab programs we can simulate the different algorithms. Playout delays for each packet are calculated and we can determine if a packet has arrived before its playout time. Thus we are able to calculate the loss percentage and average playout delay for each algorithm with each trace.

3.3.1 Generation of the traces

In order to simulate playout algorithms, we have to generate some traces which illustrate the delays experienced by voice packets. In the setup of this measurement a half-duplex voice connection is established between the two Sun Ultra Enterprise 1 workstations. A 1 kHz sinus signal is supplied by a function generator to the sendind workstation's microphone input. The signal generator is manually switched on/off to generate talkspurts and silent periods. Both workstations are running Nevot 3.35 and transmitted and received RTP-timestamps are recorded to files using the debugging option of Nevot. Network load was generated using Radcom Prism 200 protocol analyzer. Loads used with different traces are shown in Table 4.

Table 4. Used network loads

Tr.	*Traffic description*	*Frames/s*	*Bits/frame*	*Load/Mbps*	*Length/ packets*
1	Small packets	3000-5000	160	3.936-6.560	25830
2	Small packets, high load	2000-3500	320	5.184-9.072	25023
3	Large packets, bursty load	100-860	1450	1.163-10.000	24048
4	Variable size packets, bursty load	200-860	160-1450	0.2624-10.000	25337

3.3.2 Comparison of algorithms 1-4

In this section we compare algorithms 1-4 using each of the traces with parameters that gave the best performance in our experiments. For algorithm 1, $\alpha = 0.999$ is used for traces 1 and 2, $\alpha = 0.990$ is used for trace 3, and $\alpha = 0.998002$ is used for trace 4. For algorithm 2, $\alpha = 0.990$ in all traces and for

algorithm 3, w = 5000 in all traces. Algorithm 4 uses the same values for α as algorithm 1, and the value for w = 5000. Results are shown in Figures 2-5.

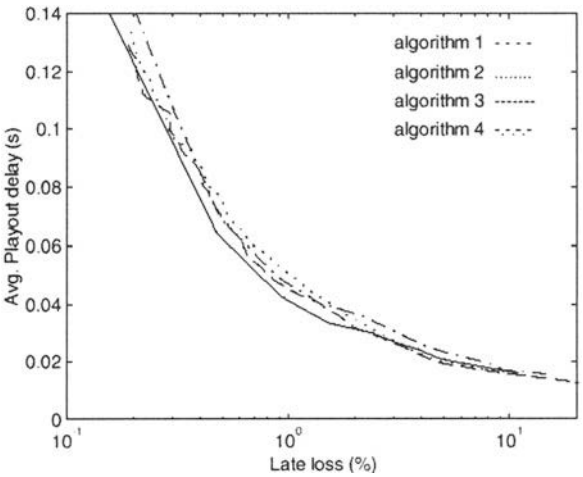

Figure 2. Algorithms 1-4 on trace 1

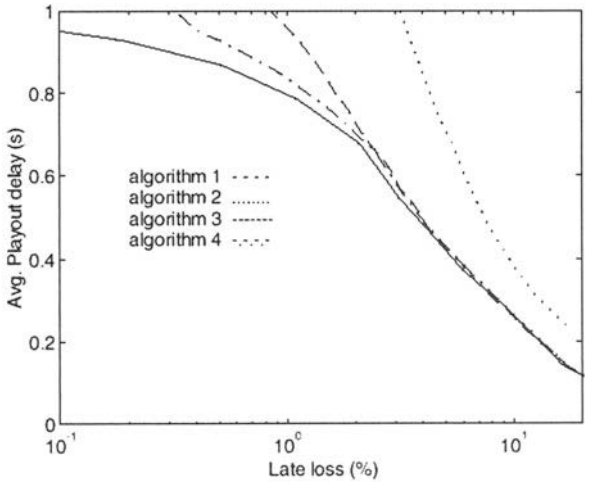

Figure 3. Algorithms 1-4 on trace 2

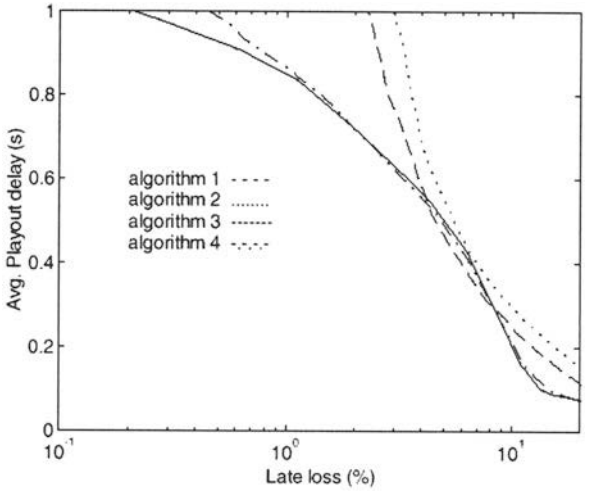

Figure 4. Algorithms 1-4 on trace 3

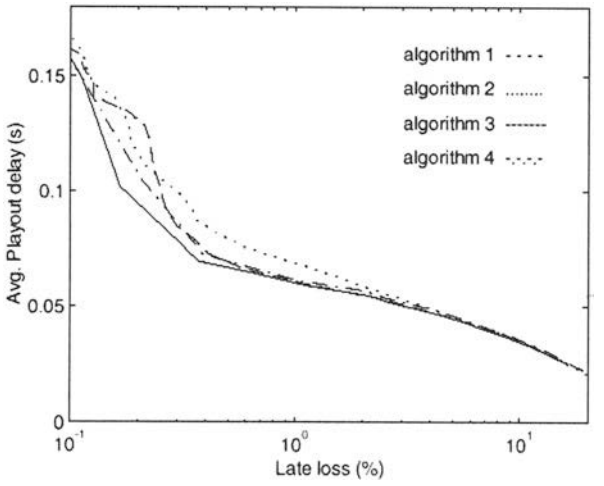

Figure 5. Algorithms 1-4 on trace 4

We can see that algorithm 3 gives the best performance over the other algorithms on all traces except on trace 3, where algorithm 1 gives the best performance at the loss rates of 4-8 %. The differences between the performance of different algorithms seemed to increase with smaller loss rates and greater delay variance. With traces 2 and 3, algorithms 1,3 and 4 gave almost similar performance at loss ratios over 4 %, but when loss ratios decrease, algorithms 3 and 4 perform a lot better.

Logically, algorithm 4 shows performance between algorithms 1 and 3 except with trace 1 where the results are about the same for algorithms 1 and 4. In the higher scale of packet loss, algorithms 1,3 and 4 perform equally well. When packet loss reduces, algorithm 3's and algorithm 4's performances improve compared to algorithm 1.

If we are using 20 ms voice packets then 5000 packets would mean 100 seconds of voice. When we consider that in average 40 % of the time the sender is active, this means that the switch from algorithm 1 to algorithm 3 happens approximately 250 seconds after the beginning of a call. The actual benefit of algorithm 4 is obtained during long calls, with duration over 4 minutes. But compared to algorithms 1 and 2, the use of a better spike detection mode from algorithm 3, improves the presentation from the beginning of a call.

4. CONCLUSIONS

In this paper we concentrated on the delays in an IP voice terminal. We measured the processing delays in a Voice over IP application software using different audio codecs. Even with the heaviest codec, the contribution of processing delay to the end-to-end delay stayed under 4 ms in a 20 ms frame time.

We also studied the end-to-end delay between two Sun Ultra workstations in a situation where the network between the workstations was unloaded and practically all delay was generated in the workstations. The components of the end-to-end delay were measured and a significant part of the delay was found out to be generated in the receiving workstation where the application software starts collecting voice packets to the playout buffer in order to avoid buffer underflows. Other components of the end-to-end delay were framing delay, processing delay, other buffering delays, HW delay in the soundcard and the operating system delay.

In our measurements, the difference in the end-to-end delays between real-time and time-sharing class processes was relatively small, between 0 and 3 ms, but these measurements were performed in the summertime when the workstations were lightly loaded.

Our experiments show that the Sun Ultra platform provides an environment where it is possible to provide bounds on the delays presented in the workstations. This can be accomplished with real-time scheduled processes that are provided by the operating system. If the application software is implemented so that it presents no additional delays and the used network connection is lightly loaded, it is possible to achieve end-to-end delays in the order of 30-40 milliseconds using 20 ms packet size.

We also made comparison of different playout algorithms under different network delay characteristics. The best performance in our simulations was obtained by an algorithm that was based on previous delay history. This algorithm was not real-time implementable as such and therefore we presented a new algorithm that was a combination of this and an existing real-time algorithm. Our algorithm was shown to outperform the other existing real-time algorithms that were compared in our studies. This algorithm gives best performance with calls that last over 250 seconds. During the first 250 seconds, delay statistics are collected and from there on, playout delays are calculated using the delay distribution of previous 5000 packets.

In this paper we simulated playout algorithms using traces that were generated by loading the network with a protocol analyzer. Future work includes simulations by using actual network traces obtained between different geographical locations and comparing the results with those presented in this paper.

ACKNOWLEDGEMENTS

This work was carried out in IPANA-project funded by Tekes and the industrial partners Miratel, Helsinki Telephone Company, Telecom Finland, Tellabs, Nokia Research Center and Nokia Telecommunications.

REFERENCES

[1] Yletyinen, T. (1997). Quality of voice over IP. *Master's thesis*, Helsinki University of technology, Telecomm tech. http://keskus.hut.fi/tutkimus/ipana/paperit.

[2] Yletyinen, T. and Kantola, R. (1998). Voice packet interarrival jitter over IP switching. *ITS '98*, Brazil.

[3] Telecommunication Standardization Sector Of ITU. (1993). ITU-T Recommendation G.114. *Technical report*, International Telecommunication Union.

[4] Jayant, N. (1980). Effects of packet loss on waveform coded speech. *Fifth Int. Conference on Computer Communications*, pp.275-280, Atlanta, GA.

[5] Montgomery, W. (1983). Techiques for packet voice synchronization. *IEEE Journal on Selected Areas In Communications*, 6(1), pp.1022-1028.

[6] Alvarez-Cuevas, F., Bertran, M., Oller, F., Selga, J. (1993). Voice synchronization in packet switching networks. *IEEE Networks Magazine*, 7(5), pp.20-25.

[7] Bolot, J. (1993). End-to-end packet delay and loss behavior in the Internet, *Proceedings of ACM SIGCOMM '93*, pp.289-298, San Francisco, CA.

[8] Ramjee, R., Kurose, J., Towsley, D., Schulzrinne, H. (1994). Adaptive playout mechanisms for packetized audio applications in wide area networks. *Proc. IEEE Infocom '94*, Montreal, Canada

[9] Moon, S., Kurose, J., Towsley, D. (1995). Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms. *Technical Paper*, Dept. of Computer Science, Univ. of Massachusetts at Amherst.

[10] Schulzrinne, H. (1992). Voice communication across the Internet: A network voice terminal. Univ. of Massachusetts, USA.