VirtualQueue: A Technique for Packet Voice Stream Reconstruction

Norival Figueira
nfigueir@nortelnetworks.com

Bay Architecture Lab
Nortel Networks
4401 Great America Parkway
Santa Clara, CA 95054

Joseph Pasquale
pasquale@cs.ucsd.edu

Computer Systems Laboratory
Dept. of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

Statistical multiplexing in packet-switched networks creates problems for packetized voice streams by introducing variable delays on delivered packets. The resulting jitter needs to be filtered so that received voice packets can be reconstructed as a continuous stream at the receiver. One common approach to reconstruction is to play back the received voice data after a delay offset from the departure time at the source of the packet stream. While the added delay helps filter jitter, one cannot introduce too much delay, otherwise, interactiveness suffers. This paper presents a new technique to find the necessary delay offset (or play-back delay) to recreate the original voice data stream. This technique gives the user control over the fraction of packets that should arrive in time to be played back so that the added play-back delay can be effectively minimized.

1. Introduction

A significant technical problem created by the integration of voice and data in a packet-switched network is the reconstruction of voice data at a receiver as a continuous stream [2, 5, 8, 9]. This is generally done by playing back the voice data after a delay offset from the departure time at the source of the voice data stream. This delay offset is called the *play-back delay*, and it is an estimate for the upper bound on delay a packet may experience.

If the network imposes a bounded delay, a fixed play-back delay may suffice. A fixed play-back delay could be set to a value large enough so that all packets arrive in time to be played back, but small enough so that humans would not be disturbed by the added delay. Some techniques have been proposed to guarantee upper bounds for delay in packet-switched networks [1, 3, 4, 6, 10, 11]. Unfortunately, these guarantees are not yet available as a common service in present networks.

Therefore (at least until such services become available), an alternative approach is to define a play-back delay that meets some more modest performance goal. For example, the user may be willing to accept some audio quality degradation in exchange for a smaller play-back delay. Thus, instead of trying to play all the received voice data, the play-back delay would be set to a value such that the likelihood that some minimum fraction of all packets would arrive in time to be played is maximized. Packets that arrived late would simply be discarded.

We present an improved technique to find the necessary play-back delay to effectively reconstruct the original voice data stream. The technique allows the user to select the fraction of packets that should arrive in time to be played. The selected performance is then used in the calculation of the play-back delay. Our technique is characterized as *tolerant* and *adaptive* (regarding determination of the play-back delay), according to the characterization space of real-time networking applications by Clark *et al* [1].

This work was supported by grants from NASA and the UC MICRO program.

2. Model of Delivery

Consider a system where audio data is generated by a source, packetized, sent over a network, received at the destination site, and sent to the audio device queue to be played. In this model, the delay d_n of packet n includes all delays the packet experiences from its generation time to its arrival time at the audio device queue. The audio device queue arrival time a_n of packet P_n is related to the delay d_n and generation time g_n by $a_n = g_n + d_n$. The audio device queue interarrival time of packet n ($n \geq 1$) is then defined by $i_n = a_n - a_{n-1}$ or $i_n = T_n + d_n - d_{n-1}$, where T_n is the inter-generation time.

In general, each packet arrives with a different delay. The variation in this delay is called *delay jitter*. As the packets arrive, the receiver process has to reconstruct the received stream of packets into a continuous stream of audio samples, and send it to the audio device to be played. To do this, a target play-back delay must be found to compensate for the delay jitter. Each arriving packet is delayed by the difference between the desired play-back delay and the delay experienced by the packet. The *play-back time* of a packet is then defined as the time the packet will be played. Packets already delayed by more than the target play-back delay are said to be *late* and are generally discarded. Late packets cause the audio device to starve, creating *silent events*.

3. The VirtualQueue Technique

VirtualQueue provides information on the amount of time a packet is late or ahead with respect to the (implicitly) calculated play-back delay. To simplify the presentation, we assume that the packet voice stream is ordered, without packet losses, and continuous (i.e., silent intervals are not eliminated from the packet stream; however, the VirtualQueue technique is usable

0-7695-0253-9/99 $10.00 © 1999 IEEE

with and will certainly take advantage of systems that support silence suppression, which is now commonly used in most Internet audio systems).

Consider the model described in Section 2 with $T_j = T$ (j = 0,1,...) and $r_{source} = r_{receiver} = r$, the sample rate of the audio devices of the sender and receiver, respectively. Suppose that all received packets are sent immediately to the audio device queue. Consider an interval of time in which the audio device has plenty of samples to play (no starvation). Call this interval of time a *non-starvation interval*. Let w_i be the size of the audio queue at the time it is about to receive packet P_i (i = 0...n):

$$w_0 = 0\,, \quad w_1 = Tr - r(a_1 - a_0) \quad ,...,$$

$$w_{n(n>0)} = nTr - r\sum_{1}^{n}(a_j - a_{j-1})$$

$$\text{or} \quad w_{n(n>0)} = nTr - r\sum_{1}^{n}(T + d_j - d_{j-1}) \quad \text{since (for } j > 0)$$

$$a_j - a_{j-1} = T + d_j - d_{j-1}\,.$$

where T is the inter-generation time of packets and Tr is the packet size in audio samples. It follows that:

$$w_n = r(d_0 - d_n) \tag{1}$$

Equation (1) shows that the delay d_n of all packets in a non-starvation interval is at most d_0, since $w_n \geq 0 \Rightarrow d_n \leq d_0$. Besides, the play-back delay of all packets played in the same non-starvation interval is fixed, and is equal to d_0. The play-back delay Pb_n of packet n is:

$$\mathrm{Pb}_n = d_n + w_n/r \tag{2}$$

since the play-back delay Pb_n of packet n is equal to its delay d_n plus the time necessary to play the amount of audio data found in the audio device queue (w_n) when packet n arrived there. Therefore, from (1) and (2): $\mathrm{Pb}_n = d_0$.

When the audio device queue size reaches zero, a silent event happens and the non-starvation interval ends. The next packet to arrive will start a new non-starvation interval, and its delay will be the new fixed play-back delay for the new non-starvation interval. The new fixed play-back delay is necessarily larger than d_0.

If there are no packet losses in the network, the play-back delay of successive non-starvation intervals will increase. If the delay distribution were bounded, the audio device queue would eventually reach a non-starvation interval with fixed play-back delay d_{max}, where d_{max} is the maximum delay experienced by a packet.

Now, suppose we have a *virtual audio device* that permitted a *negative size* to its queue. The equation describing the size of the virtual audio queue would still be equation (1), but now, valid for all time, independent of the starvation of the virtual audio device. This means that the delay d_0 in equation (1) would always correspond to the delay of the first received packet. The size of the virtual audio queue just before receiving a packet describes how much this packet is delayed relative to the delay d_0. If the virtual queue size is zero at this moment, the present packet has delay equal to d_0. If the size is negative (positive), the packet is delayed by more (less) than d_0.

If the delay distribution of received packets were known, it would be possible to calculate a play-back delay that met some performance goal defined by the user. Unfortunately, the delay distribution is generally not known. However, by recording the delays of previously received packets, one could approximate the delay distribution.

Suppose that we simulate the behavior of the virtual audio device queue. Every received packet is "sent" to the virtual audio queue at the same time it is sent to the actual audio device queue. The size of the virtual audio queue is then stored in a list every time the virtual audio queue is about to receive a packet. Define the value $w_{s\%}$ to be the largest virtual audio queue size in the saved list for which no more that $s\%$ of the saved values are smaller than $w_{s\%}$. This can be formalized as (Pr means probability): $w_{s\%} = \max\{w_i: \Pr\{w \leq w_i\} \leq s\%\}$. But, since $w_{s\%} = r(d_0 - d_{s\%})$ (this is just equation (1) with $d_n = d_{s\%}$), it can be concluded that $d_{s\%}$ is the minimum delay for which at most $s\%$ of the delays of the packets in that list are larger than $d_{s\%}$, or:

$$d_{s\%} = \min\{d_i: \Pr\{d \geq d_i\} \leq s\%\} \tag{3}$$

This shows that $d_{s\%}$ is simply the (100 - s)$^{\text{th}}$ percentile in the list of past observed packet delays, and that $w_{s\%}$ is the s^{th} percentile in the list of past saved virtual audio queue sizes. If we assume that the (100 - s)$^{\text{th}}$ percentile of the delay distribution of packets is an acceptable performance goal for the play-back delay, we can define a *performance knob* (Perf-knob) to represent this percentile. For instance, if Perf-knob is set to 99% of its maximum scale, it would mean that at least 99% of all packets should arrive in time to be played, and that at most 1% may be late (i.e., s = 1%), and dropped.

Late packets cause silent events. The *percentage of silent events* (PSE) is simply the percentage of packets that caused silent events relative to the total number of received packets. Thus, the performance knob can also be seen as implicitly defining a target maximum PSE (i.e., s = PSE = 100 - Perf-knob) for the reconstruction process.

VirtualQueue uses a *sliding window* with M slots to store the last M virtual audio queue sizes w_v. For every packet that arrives, VirtualQueue stores the virtual audio queue size w_v at the time the packet is about to be sent to the virtual audio queue (and the actual audio queue); VirtualQueue finds the value $w_{s\%}$ such that $w_{s\%} = \max\{w$: the number of virtual audio queue sizes smaller than w is less than or equal to $s\%$ of $M\}$; and VirtualQueue finds the amount c which denotes by how much the packet is late in units of audio samples. Note that s is set to the selected PSE. The calculus of c depends on the present size of the virtual audio queue w_v (from which we define that the arrived packet has delay d_x), the present size of the actual audio queue w_a, and the present value of $w_{s\%}$. If $c > 0$, the packet is late by the equivalent amount of time to play the amount c of audio samples. If $c < 0$, the packet arrived before its play-back time and there is not enough audio data in the audio queue to make the packet begin play-back with the implicitly calculated play-back delay $d_{s\%}$. Silent audio data (c silent audio samples) is inserted in the audio queue immediately followed by the insertion of this packet to bring the play-back delay to the desired value (usually done at the beginning of talk spurts).

The number of audio samples c by which a received packet is late is equal to $r(\mathrm{Pb}_x - d_{s\%})$, where Pb_x is the play-back delay

the arrived packet will have if it is immediately sent to the audio device queue upon arrival, and $d_{s\%}$ is the calculated (target) play-back delay. According to equation (1), $d_x + (w_v - w_{s\%})/r$ is equal to $d_{s\%}$. This result, and the fact that $Pb_x = d_x + w_a/r$ (since the packet is already delayed by d_x and in addition will be delayed by w_a/r at the actual audio queue), implies that:

$$c = w_a - (w_v - w_{s\%}).$$

To shed light on the algorithm, we study the four situations an arrived packet can encounter. Note that, if $w_v \geq w_{s\%}$, the arrived packet is not late (see equation (1)), since $d_x \leq d_{s\%}$, and, if $w_v < w_{s\%}$, the arrived packet is late, since $d_x > d_{s\%}$. Note also that a comparison between w_a and $(w_v - w_{s\%})$ can be reduced to a comparison between $(d_x + w_a/r)$ and $(d_x + (w_v - w_{s\%})/r)$. But $(d_x + w_a/r)$ is the play-back delay Pb_x, which is equal to $d_{s\%}$.

The four cases are: (1) $(w_v \geq w_{s\%})$ and $(w_a = w_v - w_{s\%})$. Thus, $d_x \leq d_{s\%}$, $Pb_x = d_{s\%}$, and $c = 0$. No play-back delay adjustment is needed; (2) $(w_v \geq w_{s\%})$ and $(w_a < w_v - w_{s\%})$. Thus, $d_x \leq d_{s\%}$, $Pb_x < d_{s\%}$, and $c < 0$, which indicates that the packet is ahead of time and could be delayed by the amount c; (3) $(w_v \geq w_{s\%})$ and $(w_a > w_v - w_{s\%})$. Thus, $d_x \leq d_{s\%}$, $Pb_x > d_{s\%}$, and $c > 0$. The packet is not late (since $d_x \leq d_{s\%}$), however, the audio queue has a fixed play-back delay larger than $d_{s\%}$. This situation probably happened because VirtualQueue reduced the value of $d_{s\%}$ (the target play-back delay), and the audio queue is showing that it is still playing with the old (and larger) play-back delay; (4) $(w_v < w_{s\%})$. This implies $d_x > d_{s\%}$, $Pb_x > d_{s\%}$, and $c > 0$. Therefore, the packet is late.

To compute the virtual audio queue size, it is necessary to use the *samples converted counter* (SCC) typically found on most audio devices. Its value is incremented at the same rate that the audio device plays audio samples. This counter runs in real-time independent of the state of the audio queue (empty or not). The size of the virtual audio queue is set to zero by recording the initial value of SCC (say, in SCC_init). The present size of the virtual audio queue is calculated as w_v = sent_audio - (SCC - SCC_init), where sent_audio is the number of audio samples already sent to the virtual audio queue. Note that all packets sent to the virtual audio queue are never clipped or dropped.

4. Simulations

This section presents simulations of the VirtualQueue technique to evaluate the effectiveness of the windowing heuristic (i.e., the sampling of the delay distribution). We compare the VirtualQueue technique to MeanDev and StDev, upon which many commonly-found techniques are based. These techniques use a play-back delay calculation that is based on the estimation of the delay variance of the received packets (similar to [5, 8]). MeanDev and StDev assume that the play-back delay should be K (a constant) times the mean deviation or standard deviation of the interarrival times of packets, respectively.

The simulations use trace-driven data acquired with an experimental voice-quality audio conferencing system called Phone which we have built to explore methods of transporting packetized voice streams over packet-switched networks. The interarrival times of all received packets are saved by Phone. All packets have the equivalent of 1024 bytes of audio data, which represents 128ms of speech (i.e., a rate of 8000 samples per second). The packet size was set to 1024 bytes because the audio driver of our workstations makes audio data available in minimum chunks of 1024 audio samples. The trace-driven simulation allows us to make a fair comparison of the PSE obtained by the different proposed techniques since they will be driven by the same delay distribution and without the interference of the CPU scheduler.

Network characteristics of the four traces are given in Table 1. The traced data comprises 45 minutes of digitized sound for each experiment. There are two local area network traces and two long-haul transport network traces. The long-haul transport network has a lower bandwidth than that of the local area network, often making it a bottleneck because of congestion. All the traces were acquired on the same workstation. The *Loop* trace used the same pair of workstations as in the *Long-haul* trace, except that one workstation was running a user process (a reflector) to send back all the received packets. The loop experiment was used to evaluate the heuristics when submitted to additional delay jitter created by the use of user level filters or gateways. Table 2 summarizes the statistics of the traced data.

Trace	Hops	Network
One-hop	1	LAN
Two-hop	2	LAN
Long-haul	9	long-haul
Loop	18	long-haul loop

Table 1 : Trace types.

	One-hop	Two-hop	Long-haul	Loop
Minimum	118.9	1.7	2.0	0.6
Maximum	136.5	499.7	585.3	928.9
Median	128.7	129.7	129.7	124.8
Mean	128.0	128.0	128.0	128.0
Std. Dev.	11.1	42.6	1056.1	7922.5

Table 2 : Statistics of the interarrival time in msecs.

Figures 1 and 2 present the results of simulations of MeanDev and StDev heuristics, showing the PSE obtained for different values of K, and showing that a fixed value K for different delay distributions is inadequate. To see this, assume that the user wants a PSE of less than 1% (the user always wants the minimum possible play-back delay that achieves the selected PSE). In this case, K must be greater than 10 for MeanDev, and greater than 6 for StDev, in order to bring the PSE of Long-haul and Loop to less than 1%. However, these values would be too large for the other trace types, since the play-back delay would needlessly increase. Therefore, both techniques would need an adaptive algorithm to dynamically adjust K to the observed

packet interarrival time distribution, as in [7].

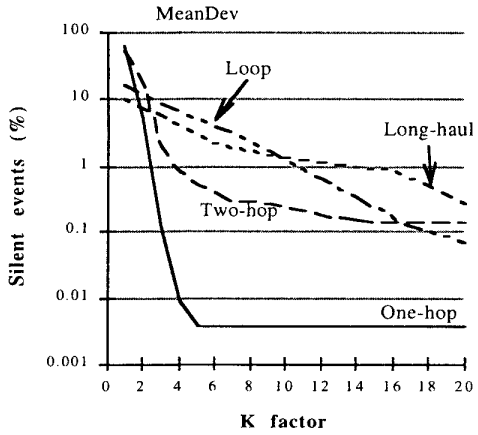

Figure 1 : MeanDev heuristic. The play-back delay is K times the mean deviation of the interarrival times of packets.

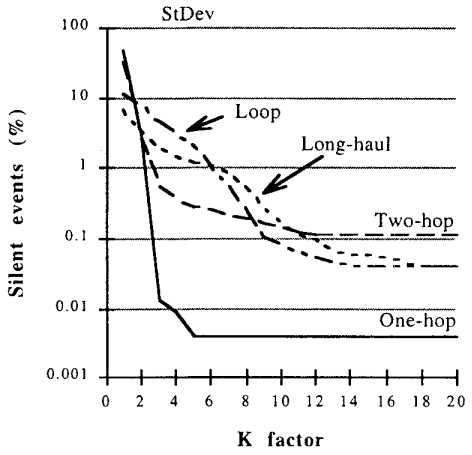

Figure 2 : StDev heuristic. The play-back delay is K times the standard deviation of the interarrival times of packets.

Figures 3 to 6 show the simulation of VirtualQueue for One-hop, Two-hop, Long-haul, and Loop, networks. Three values for PSE (or s), 1%, 5%, and 10% (i.e., user selected performance goals of 99%, 95%, and 90%, respectively) are used. VirtualQueue performed well for all traces and PSEs for window sizes larger than 300 packets.

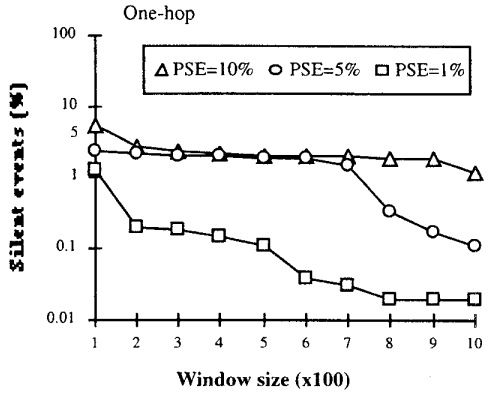

Figure 3 : One-hop using VirtualQueue.

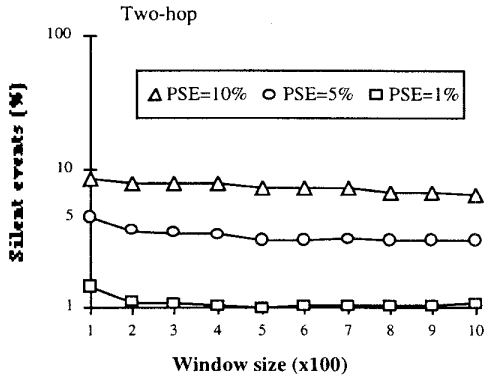

Figure 4 : Two-hop using VirtualQueue.

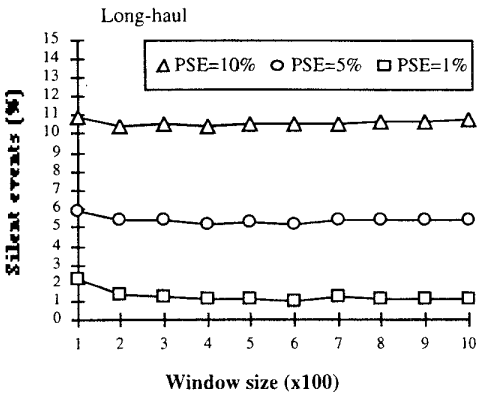

Figure 5 : Long-haul using VirtualQueue.

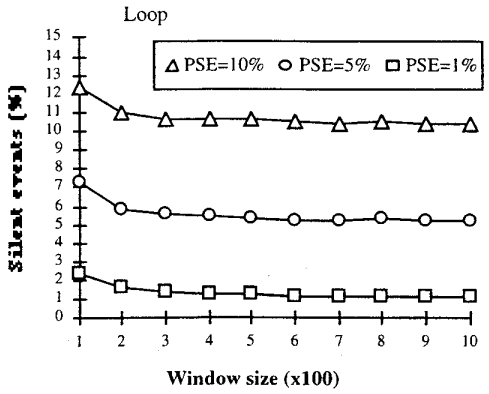

Figure 6 : Loop using VirtualQueue.

Figure 7 shows the minimum selectable PSE (the "ideal" curve) for each window size, and the actual obtained PSE. For example, the smallest selectable PSE for a window with 1000 packets is 0.1%, since this is 1 silent event per 1000 packets (or 1 late packet per 1000 packets). VirtualQueue was able to get close to the ideal curve for all window sizes.

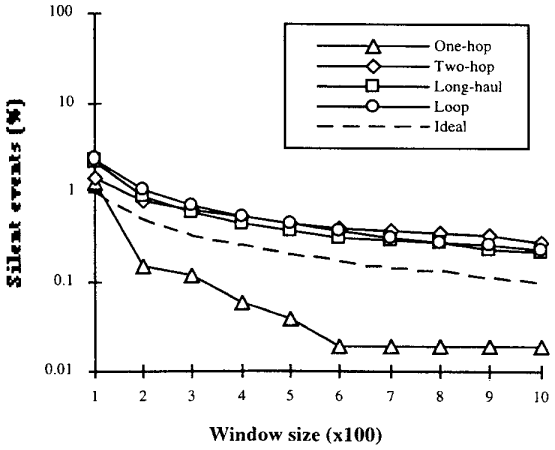

Figure 7 : VirtualQueue for the minimum selectable PSE for all traces. VirtualQueue can reliably control the play-back delay even at the minimum selectable PSE.

The set of simulations shows that the windowing heuristic used in VirtualQueue allows the implementation of a user selectable knob to reliably control the play-back delay.

5. Conclusions

We presented VirtualQueue, a technique used by a receiver of audio packets to find a good play-back delay so that the original voice stream can be effectively reconstructed. We showed that the VirtualQueue technique can be used to implement a user selectable performance knob to reliably control the play-back delay. An application with VirtualQueue acts as a non-real-time control process that periodically makes adjustments on a real-time (hardware) process, the audio device queue.

Using VirtualQueue, the play-back delay is not controlled by the buffering of audio data in the user process. Rather, the receiver process sends all audio data directly to the audio device. The negative effects of the system scheduler on the delay jitter of the received packets are then minimized. As an additional benefit, VirtualQueue avoids the use of system timers.

The VirtualQueue technique does not imply a specific play-back delay control mechanism. Instead, it provides information on the amount of time a packet is late or ahead with respect to the calculated play-back delay. Thus, it is orthogonal (and complements) techniques that use silent periods to control play-back delay.

References

[1] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. ACM SIGCOMM*, pp. 14-26, 1992.

[2] F. Alvarez-Cuevas, M. Bertran, F. Oller, and J. Selga, "Voice Synchronization in Packet Switching Networks," *IEEE Network*, Vol. 7 (5), pp 20-25, Sept. 1993.

[3] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE JSAC*, Vol. 8 (4), pp. 368-379, April 1990.

[4] S. J. Golestani, "Congestion-Free Transmission of Real-Time Traffic in Packet Networks," *Proc. IEEE INFOCOM*, pp. 527-536, 1990.

[5] V. Jacobsen and S. McCanne, "The VAT Audio Conferencing Software," Lawrence Berkeley Laboratory, University of California, Berkeley, 1992.

[6] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate Controlled Servers for Very High-Speed Networks," *Proc. GlobeCom*, pp 300.3.1-300.3.9, 1990.

[7] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proc. IEEE INFOCOM*, 1994.

[8] H. Schulzrinne, "Voice Communication Across the Internet: A Network Voice Terminal," The NEVOT Audio Conferencing Software, Univ. of Massachusetts, 1992.

[9] R. Terek and J. Pasquale, "Experiences with Audio Conferencing Using the X Window System," UNIX, and TCP/IP, *Proc. USENIX*, pp 405-417, Summer 1991.

[10] D. Verma, H. Zhang, and D. Ferrari, "Delay Jitter Control for Real-Time Communication in a Packet Switching Network," *Proc. TriCom*, pp 35-43, 1991.

[11] L. Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *ACM Trans. Computer Systems*, Vol. 9 (2), pp 101-124, May 1991. Also in *Proc. ACM SIGCOMM*, pp 19-29, 1990.