

QoS Measurement of Internet Real-Time Multimedia Services

Wenyu Jiang, Henning Schulzrinne
{wenyu,schulzrinne}@cs.columbia.edu
Technical Report CU-CS-015-99
Department of Computer Science
Columbia University

December 1999

Abstract

Real-time applications such as IP telephony, Internet radio stations and video conferencing tools require certain levels of QoS (Quality of Service). Because the Internet is still a best-effort network, the QoS of these applications must be measured and monitored in order to provide feedbacks to applications for rate and/or error control, and to both end-users and service providers. A standardized objective measurement technology makes it possible to compare between service providers in a fair way. We address the problems of packet delay and loss measurement, since they are the major determining factor of multimedia quality. We first describe the problems and techniques in obtaining good measurement results. Then, we discuss the modeling and analysis of delay and loss. Our goal is to establish feasible metrics that can reliably predict perceived quality. We find that the extended Gilbert model (2-state being a special case) is a suitable loss model, and the inter-loss distance metric is useful in capturing the burstiness between loss runs. For delay, besides the autocorrelation metric, a conditional cumulative distribution function may be useful. We apply these models to some of our Internet packet traces, and find that losses are generally bursty, and that delays usually have a strong temporal dependency component. We also find that the final loss pattern after applying playout delay adjustment (and FEC if used) still corresponds well to the extended Gilbert model.

1 Introduction

1.1 Introduction to Real-time Multimedia Services

Examples of real-time multimedia applications include IP telephony, Internet radio stations, and video conferencing. Their contents are transmitted via the network as packets at regular intervals and must be received without significant loss, preferably with a low delay. In addition, these packets must have a small delay variation (jitter) to prevent late loss, that is, an effective loss because a packet arrives too late.

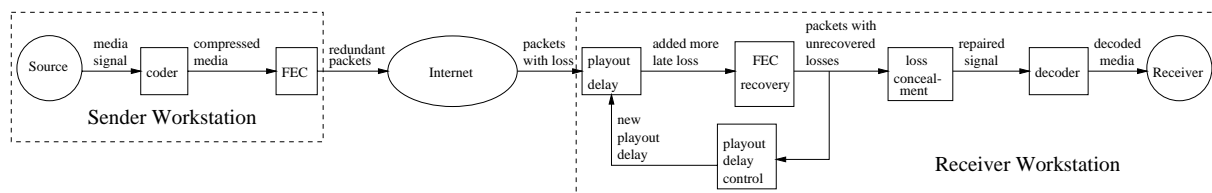


Figure 1: Loss recovery and concealment in packet audio

In these applications, audio/video signals are digitized before transmission. If every packet is received, the receiving quality will be perfect. But if a packet is lost, the quality will degrade, unless there is some

error correction mechanism based on Forward Error Correction (FEC) [34] or retransmission. A late loss due to delay jitter has the same effect as a real packet loss. Finally, a high overall delay may impede interactive communication such as a phone call. The QoS of multimedia sessions is therefore determined in general by packet loss and delay. Figure 1 illustrates how media encoding/decoding, FEC coding/recovery and playout delay adjustment work together in a typical real-time multimedia application. Playout delay adjustment is explained in the next few paragraphs.

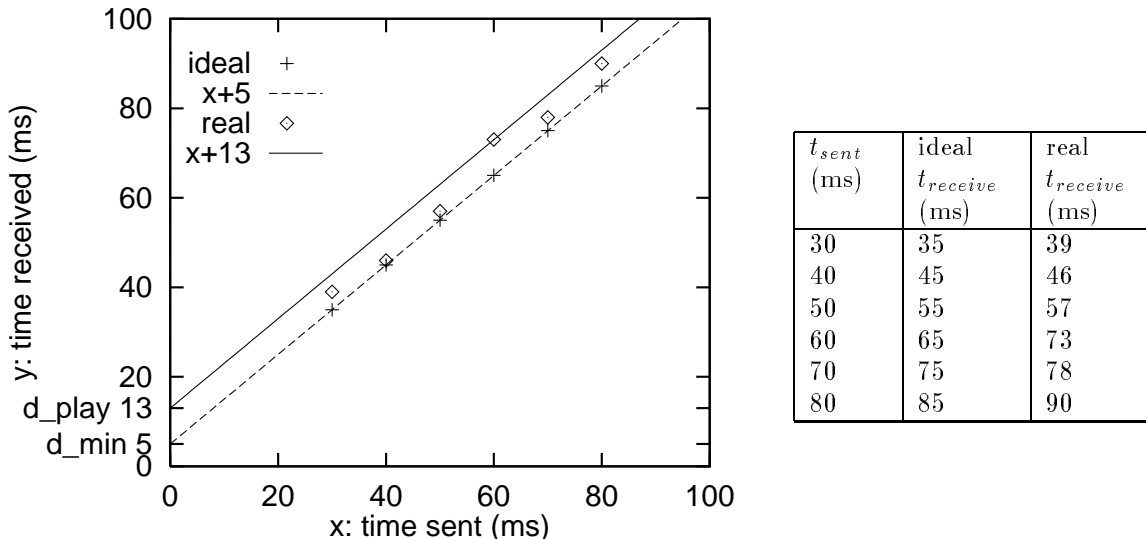


Figure 2: Illustration of packet delay and playout delay

Figure 2 shows a sample of packet delays. The abscissa is the time at which a packet is sent, the ordinate is the time at which the packet is received, and the time here is measured by a globally synchronized clock. In an ideal network such as a circuit-switched network, the delay d for a given path is constant and hopefully low, so the (x,y) points form a line $y = x + d$ (here $d = 5$ ms). This means packets can be played out as soon as they are received without having to pause.

In a packet-switched network such as the Internet, delays are not constant, as queuing delays can vary significantly over time. An example is the diamond-shaped plot in Figure 2. For audio applications, if the receiver plays out audio packets as they come in, it will have to generate a pause if the next packet arrives too late. Therefore, in Figure 2, we must wait at least d_{play} time to prevent this situation. The term d_{play} is called the *playout delay*. Usually, d_{play} is calculated by subtracting the actual play time of the *first* packet from its receiving time. In this example, the first packet is sent at 30 ms, received at 39 ms, and played at 43 ms. Therefore, d_{play} is 4 ms, although the actual play times of all packets form a line $y = x + 13$. An alternative definition of playout delay is the delay between sending time and playout time. The reader can choose either of the two definitions as long as its meaning is clear.

Many techniques have been developed for controlling playout delay [13] [36] [28] [7] [6], etc. Simple ones use a fixed playout delay, either static or determined at the start of a session. More advanced techniques exploit the existence of talk-spurts (speech) and pauses (silence) [8] in human speech. The length distribution of talk-spurts and pauses depends on silence detector settings and the speaker, but Brady [8] reported that the average length of spurts and pauses is on the order of 1 second. Since human perception is less sensitive to the length of silence, the application can choose a new playout delay at the beginning of each talk-spurt, effectively squeezing or lengthening silence gaps. This way the playout delay can be kept low when jitter is small.

Since delay and loss are the determining factors of real-time multimedia quality, we examine the issues of delay and loss measurement and various solutions. In section 2, we describe the general problems encountered in delay/loss measurement and some solutions. In section 4 we discuss the analytical modeling of loss. We

find that the distribution of loss burst length is generally heavy tailed, and the n -state extended Gilbert model [39] is a good choice for loss models. Section 5 continues with the analytical modeling of delay. In section 6 we use several queueing systems to explain the delay model introduced in section 5. Finally, in section 7, we examine the effect of playout delay control and Forward Error Correction (FEC) on our delay and loss models. Our observation is that after applying playout and/or FEC, the final effective losses are still heavy tailed, and fit well in the extended Gilbert model.

There are other factors affecting the quality of a multimedia session, such as end system delays and jitter due to the operating system.

1.2 One-Way vs. Two-way Metrics

Both delay and loss can be measured in one-way or two-way. A two-way delay is the Round Trip Time (RTT). In real-time multimedia applications, one-way characteristics is usually what determines the service quality. For example, in IP Telephony, if the $A \rightarrow B$ loss rate is low, and the $B \rightarrow A$ loss rate is high, B can still hear A's voice clearly. For one-way delay, ITU-T G.114 [16] recommends 150 ms as the upper limit for most applications, 150 to 400 ms as potentially intolerable, and above 400 ms as generally unacceptable delay. The one-way delay tolerance for video conferencing is in a similar range, 200 to 300 ms [14].

Two-way metrics are also important, especially for interactive applications such as IP telephony. It is well known that a large RTT (> 600 ms) will degrade the interactivity of an application [9]. However, if we obtain a good estimate of one-way delay in both directions, a simple addition easily yields results for two-way delay. The two-way loss probability, on the hand, is not a simple addition of one-way loss probabilities in both directions. If the one-way loss probabilities are p_1, p_2 respectively, then the two-way loss probability p is $1 - (1 - p_1)(1 - p_2)$, assuming the one-way losses are random.

The remainder of the paper will focus on one-way metrics.

1.3 Different Aspects of Delay and Loss

There are many different metrics for packet delay and loss: absolute delay, delay variation (jitter), delay correlation, that is, how previous delays affect the likely values of the next packet delay, loss rate, and loss correlation, that is, how previous losses affect the loss probability of the next packet. Section 4 - 6 discuss the modeling of delay and loss.

1.4 Components of Delay

An end-to-end delay has a number of components. The first is network delay, which consists of transmission delay, propagation delay, and queueing delay. The second is OS delay due to scheduling behavior of the operating system in the sender or receiver. The third is hardware input/output delay, for instance, a PC sound-card typically has 20 ms of output delay [22]. The hardware input delay is generally caused by packetization, for example, a G.729 frame is 10 ms, so the system must wait at least 10 ms before any further processing. The fourth is look ahead delay, because some multimedia codecs (coder/decoder) needs to look at information in the near future in order to gain better compression. An example is the G.729 codec, which uses a 5 ms look ahead time [37]. The fifth is application delay. The best known example is the play out delay that compensate for delay variation (jitter). Another example is the compression and decompression time of audio/video.

All of the components above make up the end-to-end delay. In this paper, we are mainly concerned with the measurement and modeling of network delays.

2 Measurement Problems and Techniques

2.1 General Measurement Technique for One-Way Data

To measure one-way delay and loss, we need some cooperation between two hosts. There are several possibilities:

Rely on receiver’s kernel support only, such as ICMP Timestamp request/reply message [35]. The receiver’s kernel puts its receipt time in the reply packet, which gives one-way delay if both hosts’ clocks are synchronized. It requires no human intervention at the receiver side, making it ideal for ad-hoc tests. But it is not suitable for one-way loss measurement, since the sender cannot tell if an unacknowledged packet was lost in the outgoing or returning direction. Note that when a message is lost in the returning direction, its one-way delay value is also lost, since the kernel won’t log any packets.

Run a user-level application on the receiver. When measuring one-way delays, it is effectively the same as the previous method. But it can measure one-way losses if it keeps a log of received packets.

In both approaches, sequence numbers should be put in the sender packet so that we can detect losses.

2.2 IETF IPPM Specification on Delay and Loss Measurement

The Internet Engineering Task Force’s IPPM Working Group (<http://www.ietf.org/html.charters/ippm-charter.html>) has published several specifications for measuring delay and loss. The following table is a partial summary:

Document	Reference	Topics Covered
RFC 2679	[2]	one-way delay measurement
RFC 2680	[3]	one-way loss measurement
RFC 2681	[4]	two-way (round-trip) delay measurement
ippm-loss-pattern	[20]	one-way loss pattern characterization
ippm-ipdv	[12]	jitter measurement

RFC 2679 has several key characteristics. First, it states that the clock difference and drift at both sender and receiver must be synchronized or compensated for, but it does not specify how to achieve synchronization. Second, the inter-packet sending interval should conform to a Poisson process, in order to limit any bias in the measurements. In fact, the Poisson process is used in nearly all the IPPM specifications. Third, any lost packet is considered to have infinite delay, which affects median finding. Fourth, it notes that determining whether a packet is lost can be hard, but it does not provide any particular suggestion except that the tester should record the policy used (e.g., a 5 second timeout).

The ippm-ipdv draft [12] computes jitter by subtracting one-way delays of two consecutive packets. It is equivalent to the RTP [40] definition of jitter. This is also the definition we use here for all the tables and figures in this paper.

The ippm-loss-pattern draft [20] defines several new metrics to capture packet loss patterns. These include *loss distance* (sequence difference of current and previous lost packets), *loss period* (numbering of loss bursts and their lengths), *noticeable loss rate* (percentage of lost packets with loss distances smaller than a threshold) and *inter-loss-period-length* (distance between loss bursts). In the ippm-loss-pattern draft, a sending process other than the Poisson is recommended, since the Poisson does not correspond to the periodic nature of most real-time applications. The *loss distance* and *noticeable loss rate* are particularly useful in capturing the bursty pattern between consecutive loss runs (the modeling of consecutive loss runs is discussed in section 5.2).

The IPPM specifications provide guidelines in performance measurement, but it does not provide an explicit solution to certain problems, such as clock synchronization. We address these problems in the following sections. In addition, only the ippm-loss-pattern draft addresses the problem of loss modeling

using higher order statistics. Therefore we provide more details on the current state of art in loss and delay modeling.

2.3 Measuring Loss

To detect loss, every probing packet must have a unique sequence number. It is however, difficult to know whether a packet has been lost if it has not arrived. In practice, a timeout value of 5 second for the round-trip time is often sufficient. For one-way delays, it is slightly more difficult to use timeout if clocks are not synchronized, but if several packets with a higher sequence number have arrived but the expected packet hasn't, it is almost certain that that packet has been lost.

2.4 Measuring Two-Way Delays

Measuring two-way delays does not need remote clock synchronization. Typically, the sender sends a probing packet (either UDP or ICMP echo [35]), and waits for the reply packet, and measures the round-trip time. In case of UDP, the receiver must run as a user-level process, but for ICMP, the echo/reply is handled by the kernel.

2.5 Measuring End-to-End Delays

The end-to-end delays consist of many components such as OS and playout delay. Many of them are not easy to measure in a black-box setting, that is, without modifying application software. A solution is to use another monitoring station to inspect its final output port. An example of a final output port is the audio *line-out* or *speaker* port in the case of IP telephony. Figure 3 illustrates this method. The sender transmits a signal at global time T_1 , and the monitor records and detects the signal at global time T_3 . Then the end-to-end delay between sender and receiver is $T_3 - T_1$. For the monitor to detect that a signal has arrived, it can constantly listens on the line-in port, and let the sender transmit a pre-recorded signal such as a short voice speech.

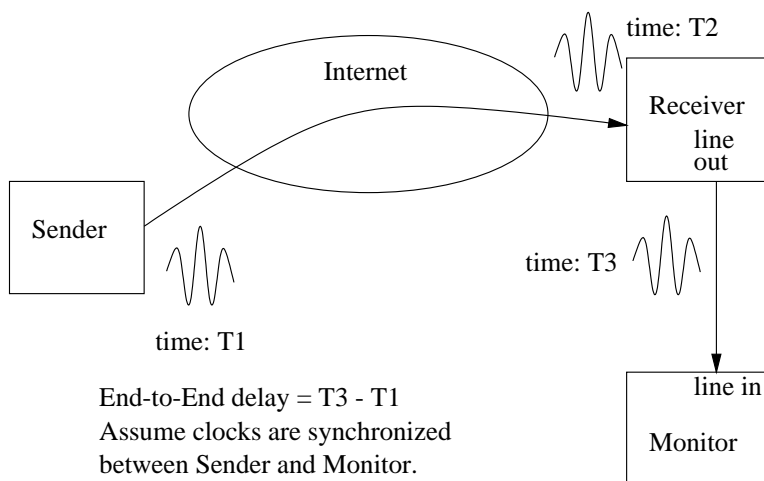


Figure 3: Measuring true end-to-end delay

This method assumes that the clocks are synchronized between the sender and the monitor, but not necessary for receiver. The clock synchronization is achieved with one of the techniques in section 2.6, such as synchronization by telephone network, or by GPS.

In later sections we focus on network delays, although the same principles can be applied to end-to-end delays as well.

2.6 Problems in One-Way Delay Measurement

Three potential problems exist in One-Way Delay measurement:

Clock synchronization. Two different hosts (especially remote ones) usually do not have their computer clocks synchronized. The NTP protocol [25] has been suggested to synchronize remote clocks. The typical performance of NTP on the Internet is in the order of 10 ms [24]. This is a non-negligible portion for a typical Internet one-way delay. If we consider the propagation delay to be a main contributor of one-way delay, then a US coast-to-coast delay is around 30ms, plus some variation.

Clock drift, or clock skew. it is a slow shift of time between two clocks because they operate at slightly different frequencies. Since no two clocks are identical, the clock drift is inevitable. In computers, the frequency of a crystal oscillator depends on its shape, size, temperature, etc. Therefore its drift rate remains mostly constant when surrounding conditions such as temperature are stable. Typical drift rates of crystal oscillators compared to Universal Coordinated Time (UTC) are on the order of 100 μ s per second. An example can be found in page 6 of [42].

Clock resolution. Many old computer systems (SVR4/386, BSD/386 V0.9, Linux x86 kernel V2.0) do not provide a time resolution higher than 10 ms, either because they do not have better clock hardware or because their system times are generated by a 10 ms periodic interrupt, as explained in [43] Appendix B. This can seriously distort the measurement values. One solution is to measure the time every N loops and divide the time by N . N must be large enough so that the measured time is larger than the clock resolution. Most new micro-processors such as Pentiums, (Ultra-)Sparcs and SGI MIPS have special hardware (e.g., the Intel 8253 clock register [43]) that provide 1 μ s resolution, which is sufficient for any practical Internet performance measurement. Therefore, we are not concerned with clock resolution problems in this paper.

A related problem is *timer* resolution, which controls the granularity of how often an interrupt occurs. [18] finds that because Windows NT has a coarse timer resolution (1 ms) and an imprecise round-off mechanism, it can cause timer “glitches”, especially in its multi-processor kernel version. It distorts the periodic nature of multimedia streams and introduces jitter at the application/OS level. When using a higher layer transport protocol such as RTP [40], the sender timestamp is always derived from the theoretical inter-packet interval of the stream, not the actual time when a packet is sent. So the receiver only sees it as a larger network delay jitter, but this effect should be thought of when writing multimedia applications.

2.7 Techniques in One-Way Delay Measurement

2.7.1 Assuming Symmetric Delays

By measuring the timestamps when a packet is sent and received, the subtraction gives a “nominal” one-way delay. As pointed out earlier, it is subject to both the clock synchronization and the drift problems.

A simple way to estimate the clock difference is by measuring the minimum Round Trip Time (RTT_{min}) over a short interval. Assuming a symmetric network, $\frac{RTT_{min}}{2}$ should be the minimum one-way delay. This is illustrated in Figure 4. [11] [33] point out, however, that network topologies are often asymmetric. Of course, a symmetric path does not necessarily mean symmetric delays, if asymmetric links exist [17]. Therefore we should always examine first whether the assumption of symmetric delays is true.

If the RTT_{min} is obtained when a packet is sent at T_1 and received at T_2 , and echoed back at T_3 , where T_1, T_3 are local time measured by the sender, and T_2 is local time measured by the receiver, then one-way delay $D = T_2 - T_1 + \delta = \frac{T_3 - T_1}{2}$, where δ is the clock difference between two hosts. Therefore $\delta = \frac{T_3 - T_1}{2} - (T_2 - T_1)$

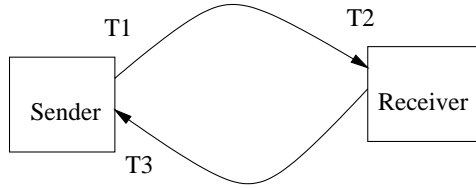


Figure 4: Clock synchronization assuming symmetric delays, T_1, T_2, T_3 are all local time

Later, for any packet sent at T_s and received at T_r , both local time. The one-way delay is adjusted as:

$$D = T_r - T_s + \delta$$

If we are only interested in the relative delays such as the jitter, clock synchronization is not necessary. But clock drift can still be a problem. With a typical drift rate of $100 \mu\text{s}/\text{sec}$, after 100 seconds, the skew would be 10 ms.

We know from previous section that assuming a stable temperature, clock drift rate is mostly constant. This corresponds to a linear change in the one-way delay over a long time. Therefore, we can split a long series of one-way delays into small intervals (e.g. of 1 sec), and extract minimum delays from them, and then perform a linear regression to calculate the drift rate. How long the time should be is a trade off. A longer time means better estimate of the drift rate, but it makes the on-line (real-time) analysis harder. There are also several techniques that improves upon the simple linear regression approach [26].

Apparently, the above method can be applied anywhere the clock drift occurs, for example between any two personal computers or workstations.

2.7.2 Telephone-Network-Based Synchronization

A traditional telephone network, also known as Public Switched Telephone Network (PSTN), can help to synchronize clocks as follows: (see Figure 5)

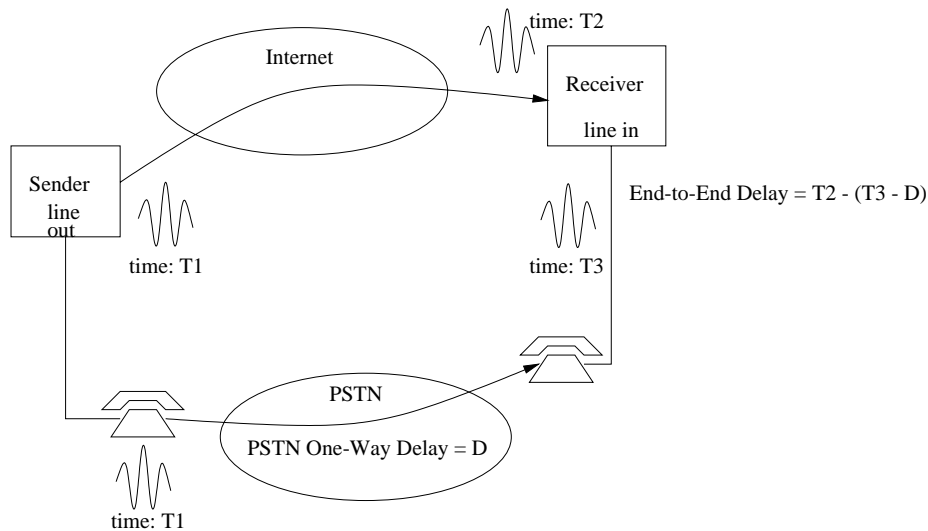


Figure 5: Using PSTN for clock synchronization, T_1, T_2, T_3 are all local time

1. Call from a telephone near the sender to another telephone near the receiver station.

2. Measure the PSTN delay. Inject a special signal into the sender telephone; short-circuit the receiver phone's headphone and microphone, this should echo the signal back; Perform a correlation analysis of sent vs. echoed signal, we can get a good estimate of the PSTN round trip delay for this phone call. Since the PSTN can be safely assumed to be asymmetric, its one-way delay is $D = \frac{1}{2}RTT_{PSTN}$.
3. Later on, the network delay can be computed as $T_2 - (T_3 - D)$, as illustrated in Figure 5. This is because the packet was sent at $T_3 - D$ (receiver clock).

This mechanism works no matter how different paths the Internet and PSTN may traverse, as long as the telephone and the workstation are close to each other.

After measuring D and the first true packet delay: we can either terminate the PSTN connection or keep it open. It may be expensive to keep the phone line active for hours. But if we terminate the connection, we are subject to the clock drift problem again. We can use the techniques developed earlier to estimate and compensate for the clock drift.

The precision of PSTN clock synchronization is typically on the order of 1 ms.

2.7.3 GPS

The Global Positioning System is a U.S. satellite-based navigational system that provides both location and time services. Among all the methods described, the GPS is the most precise one. The Surveyor project [32] uses GPS to perform highly accurate one-way delay measurements. Its project page currently advertises a precision of $50 \mu\text{s}$ for one-way delay measurement [31]. Its main drawback is the cost of equipment and deployment. The price of GPS receivers has come down to hundreds of dollars for low-end systems, but the high end ones are still expensive. <http://www.cs.columbia.edu/~hgs/internet/ntp.html> gives some informal price quotes. The measurement using GPS is very simple, so we will not repeat it here.

3 General Characteristics of Delay and Loss

3.1 Periodicity

Internet traffic has its cycles. In general, day time means heavier traffic [44] [30], hence higher delay and loss rates. Similarly, weekend usually has less traffic than weekdays. This has been mentioned in [21] [30] [23] [29].

3.2 Correlation between Delay and Loss

On a long time scale (say, every minute), the daily cycles of delay and loss matches well with each other [21]. This result is intuitive because losses generally occur because of congestion, and congestion leads to higher delays.

On a short time scale of around 100 ms, however, the inter-dependencies of delay and loss are quite different. Moon et al [27] reports that loss is not very sensitive to (either long or short) delay, either backward or forward in time, and delay is quite sensitive to loss, generally a loss appears after several high delays, and it is followed by low delay. The second result is intuitive because a loss in the Internet is mostly due to congestion, and most routers “drop-tail” when their buffers are full. After loss clears, presumably the buffers are empty again, leading to low delays.

This correlation effect has certain implications for real-time multimedia applications. Because large delays are likely followed by one or more losses, the final loss pattern after playout delay compensation will be even burstier. A burstier loss pattern may lead to lower perceptual quality, and it will affect the performance of FEC mechanisms.

Furthermore, we will show in later sections that, delay and loss have strong auto-correlation, i.e., dependency on its previous values.

4 Analyzing and Modeling Delay and Loss

Once we collect large amounts of measured data, it is important to analyze them and extract useful information from them. By developing (and verifying) various models of delay and loss, we can gain a better understanding of the network behavior, and help us predict its near-term behavior and service quality.

4.1 Causes of Delay and Loss

First, we examine the causes of network delay and loss. Network delay consists of propagation delay, processing delay, transmission delay, and queueing delay which is caused by congestion. The propagation delay is constant at $5 \mu\text{s}/\text{km}$ for optical fibers and copper wires. The transmission delay is linear to packet size. Together they form the network delay, which is part of the end-to-end delay discussed in section 1.4.

Loss is caused by congestion (router buffer overflow), routing instability such as route changes, link failure, and lossy links such as telephone modems and wireless links. Congestion is the most common cause of loss. Routing instability and lossy links are less common. Link failure rarely occurs in backbone networks.

In this paper, we are most interested in queueing delay and congestion loss, both of which are common and not easily predictable.

5 (One-Way) Loss Models

It is generally agreed that packet losses are not approximated well by a Bernoulli model [39], [46], [6]. The Bernoulli model is a model for a random process that consists of Bernoulli trials. That is, the outcome of each experiment (e.g., a packet lost or delivered) must be independent of previous trials. In the Internet, since a packet loss is likely an indication of congestion buildup, the next packet may also be lost with a high probability, leading to the temporal dependency of loss.

5.1 Spatial Loss Dependency

When multicast is used, loss also exhibits spatial dependency, that is, two or more receivers may lose the same packet. Yajnik et al [45] and Caceres et al [10] shows that spatial loss dependency sometimes exists in the Mbone, and it is usually attributed to the link between source and the first Mbone backbone router. The formula for estimating the spatial correlation [45] is:

$$\text{corr}(X, Y) = \frac{E[(X - \bar{X})(Y - \bar{Y})]}{\sqrt{E[(X - \bar{X})^2]}\sqrt{E[(Y - \bar{Y})^2]}} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2 \sum_{i=1}^n (Y_i - \bar{Y})^2}, \quad (1)$$

where X_i is 1 if packet i is lost at receiver X , 0 otherwise; Y_i is defined similarly; n is total number of packets sent. \bar{X} is $E[X]$, the expectation or average loss probability; \bar{Y} is defined similarly.

For the remainder of this paper, we focus on temporal dependency.

5.2 Temporal Loss Dependency

5.2.1 The Gilbert Model

[39], [46], [6] recommend use of a Markov model to model temporal dependency of loss. All of them analyzed the 2-state Markov model, also known as the Gilbert model (Figure 6). This model is less accurate than a general n th order (2^n -state) Markov model and an n -state extended Gilbert model (to be mentioned in the next two sections), but it is simpler to implement in monitoring applications.

p is the probability that the next packet is lost, provided the previous one has arrived. q is probability that the next packet is delivered, provided the previous one was lost. Note that normally $p + q < 1$. If $p + q = 1$, the Gilbert model reduces to a Bernoulli model.

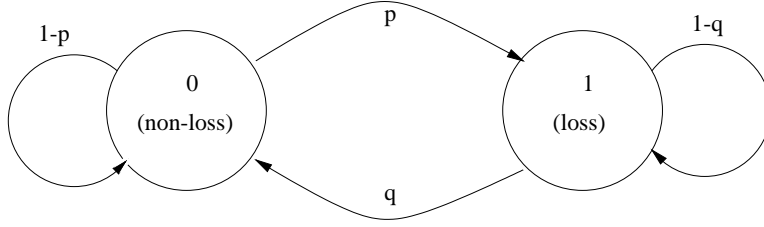


Figure 6: The Gilbert Model

From the definition, we can compute π_0 and π_1 , the state probability for state 0 and 1, respectively. In the Gilbert model they also represent the mean non-loss and loss probability, both averaged over time.

The procedure to compute π_0 , π_1 is as follows. At steady state, we should have

$$\begin{aligned} \begin{bmatrix} 1-p & q \\ p & 1-q \end{bmatrix} \begin{bmatrix} \pi_0 \\ \pi_1 \end{bmatrix} &= \begin{bmatrix} \pi_0 \\ \pi_1 \end{bmatrix} \implies \pi_0 = (1-p) \cdot \pi_0 + q \cdot \pi_1 \\ \therefore \pi_1 &= 1 - \pi_0, \\ \therefore \pi_0 &= \frac{q}{p+q}, \quad \pi_1 = \frac{p}{p+q} \end{aligned} \quad (2)$$

We can also compute p_k , the probability of a burst loss having length k , provided at least one loss occurred. The frequency a loss of length k among both loss and non-loss packets is:

$$f_k = p \cdot (1-q)^{k-1} \cdot q.$$

adding all f_k 's we get

$$\sum_{k=1}^{\infty} f_k = p \cdot q \cdot \sum_{k=1}^{\infty} (1-q)^{k-1} = p \cdot q \cdot \frac{1}{1-(1-q)} = p \cdot q \cdot \frac{1}{q} = p$$

Since probabilities must add up to 1, p_k should exclude non-loss packets,

$$\therefore p_k = \frac{f_k}{p} = (1-q)^{k-1} \cdot q \quad (3)$$

Therefore, lengths of loss bursts in a Gilbert model have a geometric probability distribution.

To calculate p, q from a packet trace, one can either inspect the entire trace, or a much simpler way is using the loss length distribution statistics. Let o_i , $i = 1, 2, \dots, n-1$ denote the number of loss bursts having length i , where $n-1$ is the length of the longest loss bursts. Let o_0 denote the number of delivered packets. Then p, q can be calculated using the formulas in [39]:

$$p = \left(\sum_{i=1}^{n-1} o_i \right) / o_0 \quad q = 1 - \left(\sum_{i=2}^{n-1} o_i \cdot (i-1) \right) / \left(\sum_{i=1}^{n-1} o_i \cdot i \right) \quad (4)$$

The correctness of Formula 4 can be validated by inspecting how a burst of length i contributes to the changes of states in the Gilbert model. As an example, for one of the Internet packet traces we collected, the CU-GMD (Columbia University to GMD Fokus, Germany) Sep 1997 trace, has the following loss burst distribution:

burst length	count
0	9992
1	34
2	5
3	1

where the count for burst length 0 means the number of delivered packets. That makes totally 10039 (around 10000) packets.

We can calculate the Gilbert model parameters as follows:

$$\pi_1 = (34 \cdot 1 + 5 \cdot 2 + 1 \cdot 3) / 10039 = 0.004682,$$

$$p = (34 + 5 + 1) / 9992 = 0.004003, \quad q = 1 - (5 + 1 \cdot 2) / (34 + 5 \cdot 2 + 1 \cdot 3) = 1 - 0.1489 = 0.8511$$

In this trace, the unconditional loss probability is equal to $\pi_1 = 0.4682\%$, whereas the conditional loss probability is $1 - q = 14.89\%$, significantly higher than π_1 .

5.2.2 Approximation Error of the Bernoulli Model on a Gilbert Process

The Bernoulli model has only one parameter: mean loss probability, \hat{p} . When it is used to approximate a Gilbert process with parameter p, q , the measured \hat{p} will be equal to π_1 in the corresponding Gilbert model:

$$\hat{p} = \pi_1 = \frac{p}{p+q} \quad (5)$$

When we use a Bernoulli model with parameter \hat{p} , we are essentially using a Gilbert model with parameter \hat{p}, \hat{q} where $\hat{p} + \hat{q} = 1$. Using Equation (3), we would predict burst loss length distribution as:

$$\hat{p}_k = (1 - \hat{q})^{k-1} \cdot \hat{q} = \hat{p}^{k-1} \cdot (1 - \hat{p}) = \left(\frac{p}{p+q}\right)^{k-1} \cdot \frac{q}{p+q} \quad (6)$$

In the Internet, since a packet loss is an indication of congestion buildup, usually $p < q$ and $p + q < 1$, when $k = 1$, $\hat{p}_1 = \frac{q}{p+q} < q = p_1$, so the Bernoulli model over-predicts single losses, compared to the Gilbert model.

When $k > 1$,

$$\hat{p}_k = \left(\frac{p}{p+q}\right)^{k-1} \cdot \frac{q}{p+q} = \left(1 - \frac{q}{p+q}\right)^{k-1} \cdot \frac{q}{p+q}$$

Because the component $(1 - \frac{q}{p+q})$ is smaller than $(1 - q)$, even if \hat{p}_k could be larger than p_k , as k increases, eventually \hat{p}_k will be smaller than p_k .

Therefore, the Bernoulli model will under-predict the probability of long bursts compared to the Gilbert model.

To illustrate, let's compare using the CU-GMD Sep 1997 trace again. Here $\hat{p} = \pi_1 = 0.004682$, the average loss length distribution is:

loss length k	1	2	3	4	5
\hat{p}_k	0.99532	0.004660	0.00002182	1.02e-7	4.78e-10
p_k	0.8511	0.1267	0.01887	0.002810	0.0004184

It is evident that for this trace the Bernoulli model over-estimates single loss probability while it under-estimates probability of longer loss bursts. Under the Bernoulli model, even double losses are highly unlikely for this trace, with average number of occurrence equal to $0.004660 * (34 + 5 + 1) = 0.1864$, whereas the trace has 5 double losses.

5.2.3 General Markov Model

An n th-order Markov chain model is the most general model for capturing dependencies among events. n means it needs to remember the last n events. Because the next event is assumed to be dependent on the last n events, there are 2^n possible states to describe past n binary events. Let X_i denote the binary event for i th packet, 1 for loss, 0 for non-loss. The parameters to be determined in an n th order Markov model are:

$$P[X_i | X_{i-1}, X_{i-2}, \dots, X_{i-n}] \text{ for all combinations of } X_i, X_{i-1}, X_{i-2}, \dots, X_{i-n}.$$

For instance, $P[X_i = 0 | X_{i-1}, X_{i-2}, X_{i-3} = 001]$ means the probability packet i is not lost, provided packet $i - 1, i - 2$ are not lost and packet $i - 3$ is lost. 2^n states are needed to distinguish past history, and there are 2 transition choices, either 0 (next packet delivered) or 1 (next packet lost). Therefore, one

needs to measure 2^{n+1} parameters. 2^{n+1} is a very large number even for modest n like 20. Yajnik et al [46] show that their experiments typically have $n \leq 6$, but some experiment traces require n to be 20 to 40, which is practically impossible to implement. For small n 's, say $n < 8$, a full n th order Markov model is still feasible. Yajnik et al [46] calculates the necessary order n for their packet traces, but it does not give any quantitative analysis on how much precision the n th Markov model gains over other simple models such as the 2-state Gilbert model. In a real Internet packet trace, provided that the path monitored is reasonably healthy, there will not be many packet losses to begin with, so the gain of precision by the general Markov model may not be enough to justify its complexity. Furthermore, it is much harder to compare the health of two paths using the general Markov model, since one needs to compare on the order of 2^n parameters. For this reason, we discuss a simplified Markov model - the extended Gilbert model, in the next section.

5.2.4 Extended Gilbert Model

The problem with a general n th-order Markov model is its exponential number of states. Sanneck et al [39] proposes a different approach that leads to fewer states. In fact, to remember n events, one only needs $n + 1$ states. It is called the *extended* Gilbert model. The key distinction between them is: a general Markov model assumes all past n events can affect the future; whereas in an extended Gilbert model only the past (up to) n consecutive loss events will affect the future. Although the latter assumption may not always be valid, if we agree delivered packet between two loss bursts is a possible indicator of congestion relief, then the correlation between two such loss bursts will be small.

The consecutiveness of past events (losses) in the extended Gilbert model leads to an exponential reduction in the number of states. It works as follows: the system keeps a counter l , which is the number of consecutively lost packets, but it is reset whenever the next packet is delivered. In this model, only the past adjacent loss burst counts, which may not always be true, but it could be a good heuristic. The parameter to determine in an extended Gilbert model is $P[X_i | X_{i-1} \text{ to } X_{i-l} \text{ all lost}]$, where X_i has the same definition as in the Markov model. Figure 7 illustrates how the extended Gilbert model works:

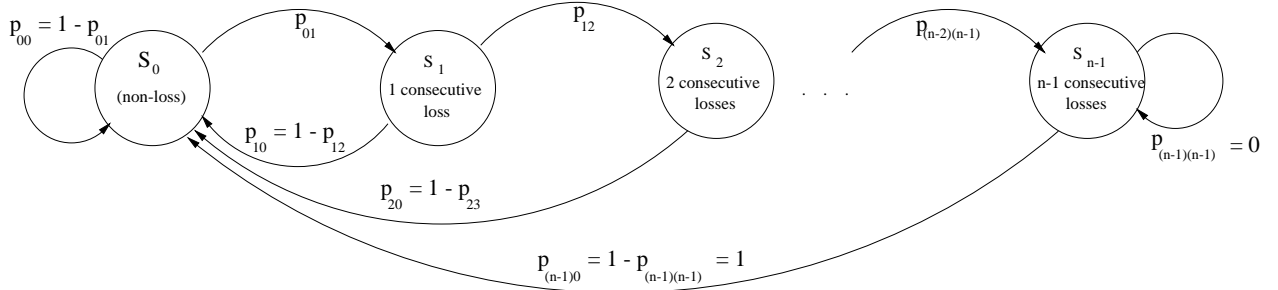


Figure 7: The Extended Gilbert Model

Therefore, the transition matrix for the above model is:

$$\begin{bmatrix} p_{00} & p_{10} & p_{20} & \dots & p_{(n-2)0} & p_{(n-1)0} \\ p_{01} & 0 & 0 & \dots & 0 & 0 \\ 0 & p_{12} & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{(n-2)(n-1)} & p_{(n-1)(n-1)} \end{bmatrix}$$

Therefore, its steady probability $(\pi_0, \pi_1, \dots, \pi_{n-1})$ can be calculated as follows:

$$\begin{bmatrix} p_{00} & p_{10} & p_{20} & \dots & p_{(n-2)0} & p_{(n-1)0} \\ p_{01} & 0 & 0 & \dots & 0 & 0 \\ 0 & p_{12} & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{(n-2)(n-1)} & p_{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \dots \\ \pi_{(n-1)} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \dots \\ \pi_{(n-1)} \end{bmatrix}$$

plus the equation

$$\sum_{i=0}^{n-1} \pi_i = 1$$

It is now clear that the Gilbert model is a special case of the extended Gilbert model when $n = 2$. [39] gives the formula to calculate the parameters for the extended Gilbert model, as follows:

$$p_{01} = \left(\sum_{i=1}^{n-1} o_i \right) / o_0 \quad p_{(k-1)(k)} = \left(\sum_{i=k}^{n-1} o_i \right) / \sum_{i=k-1}^{n-1} o_i \quad (7)$$

where $k = 2, 3, \dots, n-1$, o_i is the same as defined in section 5.2.1.

As an example, the parameters calculated from the CU-GMD Sep 1997 trace are:

$$p_{01} = 0.004003 \quad p_{12} = 0.150000 \quad p_{23} = 0.166667$$

6 Delay Models

6.1 First-order Statistics

For packet delays, the most common metrics are mean and median of delay, variance of delay. The median is less sensitive to occasional high delays. They provide an overview of network conditions, but more details are required to predict performance of real-time multimedia applications. Simple examples include pmf (probability mass function), pdf (probability density function), and cdf (cumulative distribution function). More elaborate metrics try to capture correlation and dependency in delays.

6.2 Observing the Dependency Component in Delays

Like packet loss, delay also has some degree of dependency on the previous values because delay is also a likely indication of congestion.

One way to measure this dependency is by auto-correlation analysis. Let d_i denote the delay of i th packet, n the total number of packets measured, d the delay random variable, and \bar{d} the average delay.

$$\rho(d, l) = \frac{\sum_{i=1}^{n-l} (d_i - \bar{d})(d_{i+l} - \bar{d})}{\sum_{i=1}^n (d_i - \bar{d})^2} \quad (8)$$

l is called the lag of the correlation. The auto-correlation gives a value between $[-1, 1]$. If it is almost 0, then the dependency is small. If it is close to 1, it is positively correlated (a high delay will be followed by a high delay). If it is close to -1, it is negatively correlated (a high delay will follow by a low delay). For different lags, the value will differ. In general, it is bigger when $l = 0$, and drops to 0 as l increases. This matches our intuition, that adjacent delays are correlated, but delays of packets far apart are independent. However, it is shown in [27] that sometimes auto-correlation can go up periodically. They conjectured it is caused by TCP or some other synchronization effect in the network.

The auto-correlation value is a good indicator of dependency, but it is difficult to quantify the dependency using this metric. Therefore, we introduce a new metric for this purpose: **ccdf**, the conditional cumulative delay distribution.

It is defined as:

$$f(t) = P[d_i \geq t | d_{i-l} \geq t], l = 1, 2, 3, \dots, \quad (9)$$

where l is the lag.

According to the above definition, the ccdf should be more appropriately called the inverse conditional cumulative distribution function. For brevity, we still call it ccdf. The ccdf is not the only way to capture conditional dependency. An alternative form of ccdf could be:

$$f(t) = P[d_i \geq t | d_{i-1}, d_{i-2}, \dots, d_{i-l} \geq t] \quad (10)$$

Also, we can compute the non-cumulative conditional pdf as:

$$f(t) = P[d_i | d_{i-l} \in (t_1, t_2)] \quad (11)$$

Similarly, we can have:

$$f(t) = P[d_i | d_{i-1}, d_{i-2}, \dots, d_{i-l} \in (t_1, t_2)] \quad (12)$$

There are many kind of metrics and functions we can use to characterize conditional delay dependency. The author have found little known literature on this topic. Bolot [5] analyzed the conditional property of consecutive delays. We have examined both (9) and (11). For (11), the results are intuitive, for previous high delays, the conditional pdf also shifts toward right.

Among all these metrics, it seems ccdf is more useful, especially in real-time multimedia applications. This is because in such setting, any packet with a delay higher than the playout delay is effectively lost. By inspecting cdf (unconditional), at a given playout delay D_p , the percentage of late (lost) packets is $1 - cdf(D_p)$. By inspecting ccdf (conditional) at D_p , we can estimate the burstiness of late losses.

Examples of real-world ccdf graphs are shown in Figure 8. It uses the CU - GMD Sep 1997 trace we mentioned in previous sections. The inter-packet interval is 30 ms.

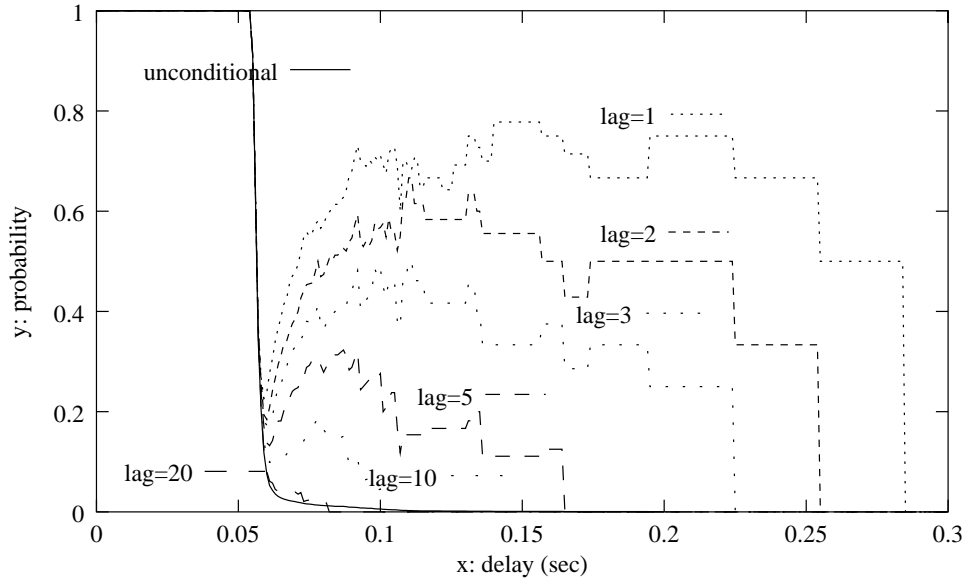


Figure 8: conditional cdf (ccdf) for CU - GMD Sep 1997 trace

We can see that delays in Figure 8 have a relatively high dependency. But as the lag l increases, it drops quickly, which is also confirmed by the auto-correlation function. What auto-correlation does not tell, is that beyond a certain threshold D_t , the conditional dependency increases rapidly. In this graph, this turning

point D_t is at around \bar{D} (average delay), which is about 58 ms (not shown here). Some theoretical analysis may be required to interpret whether and why the turning point D_t occurs at \bar{D} .

Figure 8 shows that low delays are highly independent of each other temporally, because their variations are small, and such variations are mostly random noise. That's why the ccdf matches almost perfectly with the inverse cdf. This again, matches our intuition, but it is not directly evident from auto-correlation result.

6.3 Theoretic Models and Analysis of ccdf

In an attempt to explain why the ccdf in Figure 8 increases significantly beyond a certain point, we use some approximated model to analyze the queueing delay. These models are not identical to real networks, but they offer some insight. Plus, when we tried different models, all of them showed a high conditional delay component. Hence it seems to imply conditional delay is ubiquitous. We also used computer-based simulations to verify some of our findings.

6.3.1 $M/D/1$ Queueing System

In this model, we assume the whole network is a queueing system with a Poisson arrival process (M), and the time needed to process a packet is constant (D). Under this model, it is relatively easy to infer how many packets are in the queue, if you know the total delay (waiting time) of the last packet. Note that the delay doesn't include service time, which is constant.

[Analysis]

Assume average arrival rate is λ , service rate is μ .

If packet i experiences delay $d_i > 0$, it means $\lfloor d_i \cdot \mu \rfloor$ packets are ahead of it, in the queue.

If after time τ , packet $i + 1$ arrives, then its waiting time would be

$$d_{i+1} = \max(0, d_i - \tau + \frac{1}{\mu}) \quad (13)$$

That is, if packet $i + 1$ came very late, then the entire queue would have been cleared up, so waiting time would be 0. Otherwise, since service time is constant, packet $i + 1$ has one more packet ahead of it in the queue, so waiting time is increase by $\frac{1}{\mu}$, but it came τ later, so delay is decreased by τ .

Equation (13) is also called Lindley's recurrence equation [19].

Due to the Poisson arrival process, τ is exponentially distributed: $pdf(t) = \lambda e^{-\lambda t}$

The distribution of d_{i+1} is shown in Figure 9.

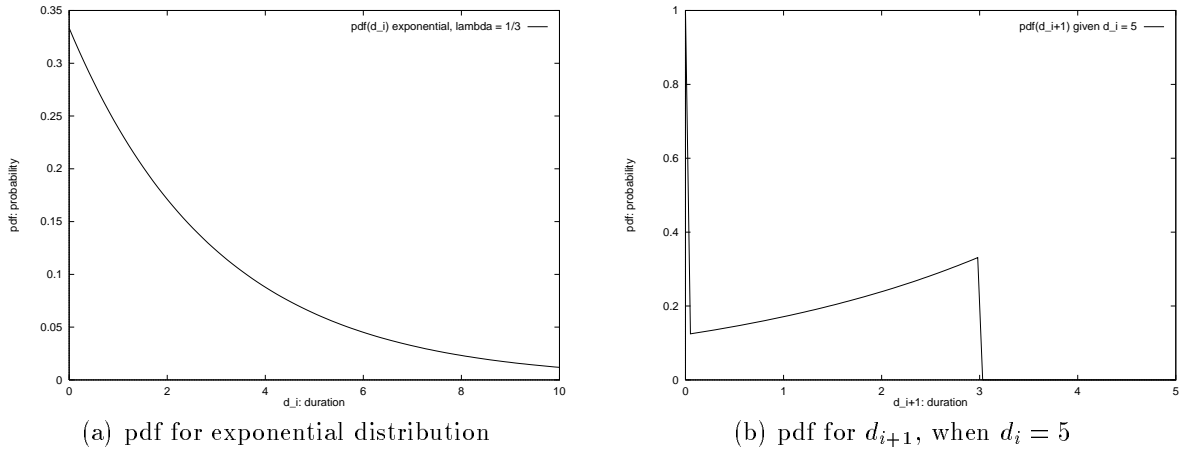


Figure 9: effects of delay auto-correlation for exponential distribution, $\lambda = \frac{1}{3}$, $\mu = \frac{1}{2}$

Note that in the right graph, its pdf value at 0 is ∞ , because it has a finite probability mass at 0. Therefore,

$$\begin{aligned} P[d_{i+1} \geq t] &= 1, & t = 0 \\ &= 1 - e^{-\lambda(d_i + \frac{1}{\mu}) - t}, & t \leq d_i + \frac{1}{\mu}, \\ &= 0, & t \geq d_i + \frac{1}{\mu}. \end{aligned}$$

We can readily see that if d_i is large, d_{i+1} will also likely be large. The actual derivation of $P[d_{i+1} \geq t | d_i \geq t]$, however, is difficult. Therefore we used approximation to draw its curve, then we used simulation for verification.

From Appendix C of [1], Table 18, an $M/D/1$ system has the following steady state probabilities:

$$p_0 = 1 - \rho, \quad p_1 = (1 - \rho)(e^\rho - 1),$$

$$p_n = (1 - \rho) \sum_{j=1}^n \frac{(-1)^{(n-j)} (j\rho)^{(n-j-1)} (j\rho + n - j) e^{(j\rho)}}{(n-j)!}, \quad n \geq 2.$$

where $\rho = \frac{\lambda}{\mu}$, and p_n is the steady state probability that the queue has a length n .

The distribution function of q (queueing time) is given by

$$P[q \leq t] = W_q[t] = \sum_{n=0}^{k-1} p_n + p_k \left(\frac{t - (k-1)W_s}{W_s} \right),$$

where $(k-1)W_s \leq t \leq kW_s, k = 1, 2, \dots$

Therefore, the computer approximation (pseudo) code is as follows:

```

for (t = 0 ; t ≤ t_max ; t += dt) {
  C_b = 0 ; /* base counter */
  C_m = 0 ; /* match counter */
  for (q_l = t ; q_l ≤ q_max ; q_l += dq) {
    p = W_q[q_l + dq] - W_q[q_l] ; /* P[q_l ≤ q ≤ q_l + dq] */
    C_b += p ;
    if (t == 0) C_m += p ;
    else C_m += p · (1 - e^{-λ(q_l + 1/μ - t)})
  }
  f(t) = C_m / C_b ; /* P[d_{i+1} ≥ t | d_i ≥ t] */
}

```

The precision of the above code depends q_{max}, dt, dq . The larger q_{max} , the smaller dt, dq , the better precision the simulation achieves.

Figure 10 (the dashed curve) shows the approximated plot. The solid curve shows the computer simulation result, which maps closely with the approximation. We used *simul* [41] as the simulation tool. It is a simple event-based simulator for modeling queueing systems.

6.3.2 $E_k/D/1$ Queueing System

The arrival process in the real world are not necessarily Poisson. Therefore, we also examine Erlang- k arrival process. The steady state formulas for $E_k/D/1$ systems are more complex than that of $M/D/1$ systems, so we only include simulation results below in Figure 11.

For Figure 11, we showed the simulated cdf plot for different lags. It is easy to see that as the lag increases, the conditional component (correlation) drops off. As a comparison, we also included the same plot for $M/D/1$ system.

6.3.3 $G/D/1$ Queueing System with multiple input streams

In a real packet-switched network, the traffic comes from multiple input lines and goes to multiple destinations. To simulate this “fan-in” and “fan-out” behavior, we modified our simulation program to generate

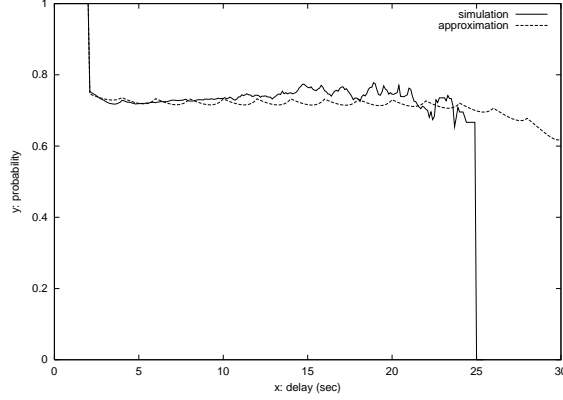


Figure 10: Approximation and Simulation of an $M/D/1$ system, $\lambda = \frac{1}{3}$, $\mu = \frac{1}{2}$.

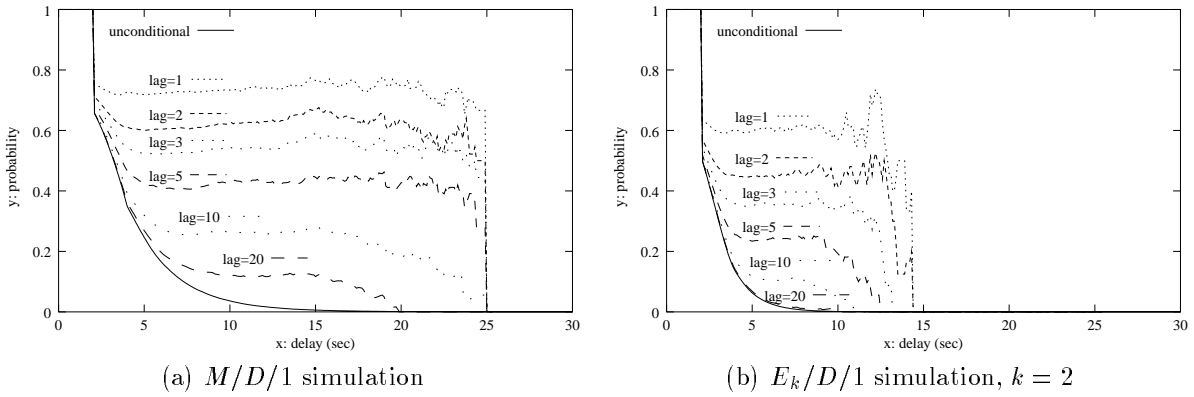


Figure 11: Simulation of $M/D/1$ and $E_k/D/1$ systems with different lags, $\lambda = \frac{1}{3}$, $\mu = \frac{1}{2}$

two input streams, one foreground traffic and another background traffic. We also used slightly different service rates between foreground and background traffic, but the results are similar to the case using the same service rate. For simplicity, we did not implement the “fan-out” behavior, so there is only one output stream. This is illustrated in Figure 12, where two independent streams enter a shared queueing system. The simulation program examines the delay dependency of the foreground traffic and outputs the results in Figure 13.

The notation used in Figure 13 is as follows: the first symbol in the parenthesis is the pattern of foreground traffic, the second symbol in the parenthesis is the pattern of background traffic. As an example, 13(a) has an exponential foreground traffic and an Erlang- k background traffic. λ_1, λ_2 are the arrival rates of foreground and background traffic, respectively. μ_1, μ_2 are the service rates of foreground and background traffic, respectively. From the result of simulation, it seems that exponential foreground traffic exhibits stronger delay dependency than period foreground traffic. Exponential (as opposed to Erlang- k) background traffic leads to higher delay dependency in the foreground traffic. But in all of the sub figures, the existence of delay dependency is evident.

6.3.4 Implications of Temporal Delay Dependency

The auto-correlation gives a coarse estimate of how network delays exhibit temporal dependency. The conditional cdf (ccdf) gives a more quantitative measure of this dependency. We have used several theoretical queueing models to explain why the shape of ccdf gets elevated when the previous delay is high. Although

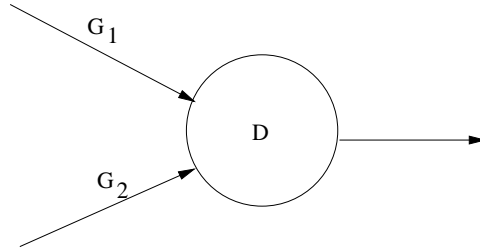


Figure 12: A $G/D/1$ system with 2 input streams

these models are not perfect, all of them consistently show the existence of delay dependency in a real queueing system.

The delay dependency has a strong implication on the quality of real-time multimedia services. In general, it means when a packet's delay exceeds a certain value (say the playout delay), the same can happen to the next packet, with a strong probability. The result is burstier late-losses, which would degrade the performance of FEC and could degrade the performance of loss concealment.

In the next section, we will examine the characteristics of final losses after applying all the playout/FEC controls.

7 Applying Delay and Loss Models to Real-Time Multimedia Streams

Our goal is to predict end-user's perceived quality when using a real-time multimedia application, based on network performance. The final quality perceived by an end-user depends on a lot of factors, but Figure 1 gives a general view of how such an application processes multimedia streams. Here we assume loss recovery is done by FEC (Forward Error Correction), but the diagram is similar for a retransmission-based technique. The FEC we refer to is the traditional FEC, rather than a low bit-rate redundancy FEC as mentioned in [15]. A low bit-rate redundancy FEC would serve as a type of loss concealment in Figure 1.

So far we discussed the modeling of delay and loss independently. However, from the above diagram, it is obvious that delays are translated into late-losses when a playout delay adjustment algorithm is applied. It would be interesting to see what loss pattern the playout mechanism introduces. Table 1 is a brief summary of loss bursts for a CU-GMD Sep 1997 trace. The "unrecovered loss" is simply network packet losses if FEC is not used. If FEC is used, it means the number of loss-bursts that could not be recovered by FEC. The "merged bursts" column is the number of loss-bursts after merging late losses and unrecovered losses. For example, if a single late loss occurs at packet 37, and an unrecovered loss occurs at packet 38, then they form one loss burst of length 2, assuming no other packets are lost before or after them.

In IP telephony applications, silence suppressions are often used to transmit only talk-spurts. [8] has a few statistics on spurt/gap distributions, but its data is more than 30 years old. Lack of a better model, we used exponential distribution (1.5 sec average) plus a bottom threshold (240 ms) to describe the length distribution of both talk-spurts and silence gaps. The randomly generated spurts and gaps are then applied to an existing packet trace for playout control simulation.

We used several playout control algorithms, the first is Exp-Avg, the exponential average algorithm in [36]. The second is the virtualized delay version of Exp-Avg, as mentioned in [38]. When FEC is used, if the playout delay adjustment algorithm is not aware of how long it needs to wait for FEC data to perform in-time recovery, the recovered data are often too late for playout. Always postponing playout for the whole FEC block to arrive is too conservative when loss rate is low. Therefore the playout algorithm should be "virtualized" by taking the minimum of a normal packet's arrival time and the time when necessary FEC data arrive. The third is Prev-Opt, also mentioned in [38]. The fourth is the virtualized version of Prev-Opt when FEC is used.

In Table 1 (a), we see that there are 17 single network losses and 51 single late losses. In Table 1 (b),

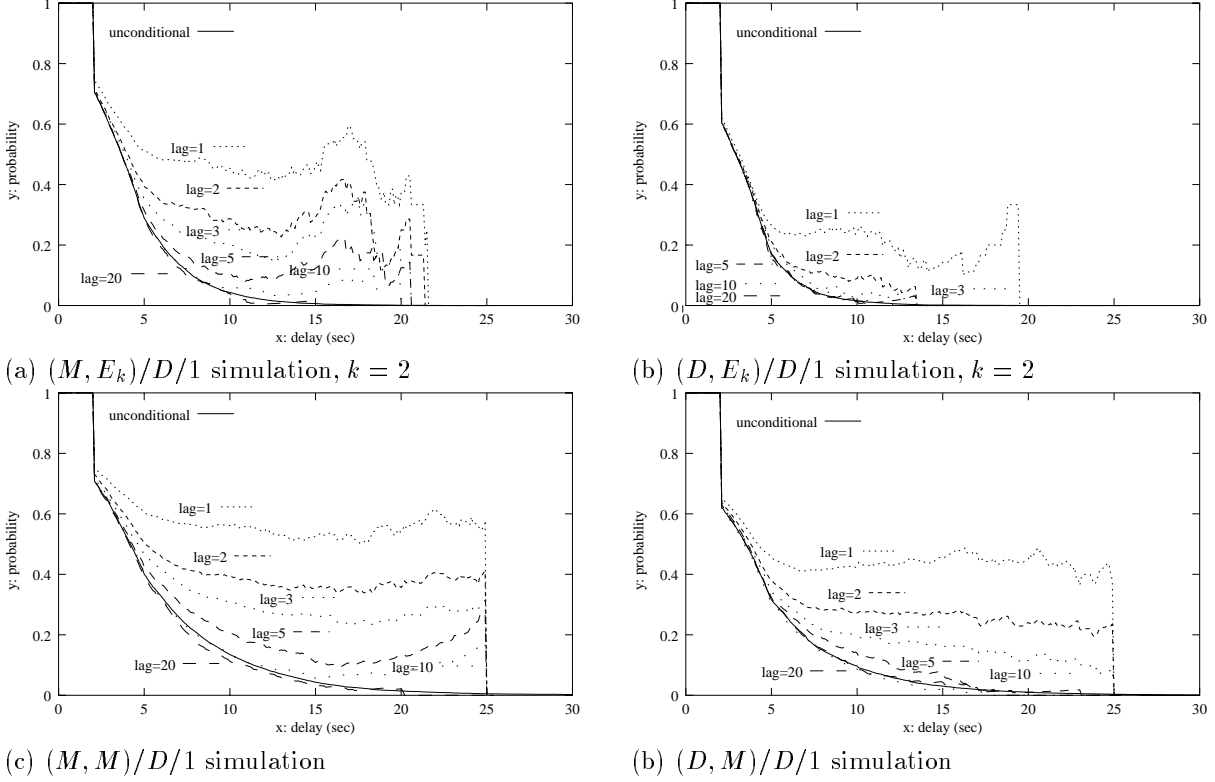


Figure 13: Simulation of $G/D/1$ systems with multiple input streams, only foreground traffic is examined. $\lambda_1 = \frac{1}{18}, \lambda_2 = \frac{1}{5}, \mu_1 = \frac{1}{2}, \mu_2 = \frac{1}{3}$

only 1 out of 17 single network losses could not be recovered by FEC. That means the FEC does a good job of loss recovery. But since the loss rate is already very low (there are totally about 10000 packets sent), and the jitter of this trace is also low, most of the recovered packets are not played back in time.

Also, if the late losses and real losses are adjacent to each other, they merge into bigger loss bursts, as we have just explained. This is evident in the last column of Table 1 (a), where number of triple losses increased from 3 to 7. This effect is much less visible in Table 1 (b), because there are not many unrecovered losses to begin with.

We have performed the same experiment on other network traces we obtained. Table 2 uses a CU-UMass Sep 1997 trace. The results are in general similar: the final loss pattern are still best described by an n -state extended Gilbert model, and usually $n > 2$. But the effects of merging between unrecovered and late losses are less evident. This is because the other traces have a much larger jitter, and hence a more conservative (larger) playout delay. The end result is there are far fewer late losses in these traces.

The conclusion we draw here is: after applying playout control algorithms and possibly FEC, the final loss pattern is still best described an n -state extended Gilbert model, and usually $n > 2$. There is also a merging effect between late losses and unrecovered losses. This effect, however, is minimized when both delay jitter is high and FEC is employed, which leads to a more conservative (higher) playout delay and recovery of most lost packets.

In brief, the final loss pattern after playout adjustment is burstier than one would have expected. How this affects end-user perceptual quality is still unknown. [37] has reported that the built-in loss concealment mechanism of G.729 codec can usually repair a single loss well, but does not work well on longer bursts. Therefore with the same loss probability, a burstier loss pattern could degrade a voice signal to a greater degree than random losses, but there might well be exceptions. For video streams, since a video frame often

burst length	late-loss only	unrecovered loss	total loss	merged bursts
1	51	17	68	56
2	8	3	11	9
3	3	0	3	7
4	0	0	0	1
5	1	0	1	1

(a) Without FEC, using Exp-Avg playout

burst length	late-loss only	unrecovered loss	total loss	merged bursts
1	50	1	51	51
2	8	0	8	8
3	7	0	7	7
4	0	0	0	0
5	1	0	1	1

(b) With FEC, using (5,3) Reed-Solomon code and virtualized Exp-Avg

burst length	late-loss only	unrecovered loss	total loss	merged bursts
1	1	17	18	18
2	0	3	3	3
3	0	0	0	0
4	1	0	1	1

(c) Without FEC, using Prev-Opt

burst length	late-loss only	unrecovered loss	total loss	merged bursts
1	8	1	9	9
2	1	0	1	1
3	1	0	1	1

(d) With FEC, using (5,3) Reed-Solomon code and virtualized Prev-Opt

Table 1: Effect of playout control on final loss burstiness, CU-GMD Sep 1997 trace

consists of several network packets, bursty losses may actually be more preferable than random losses [14].

8 Conclusion

We discussed several topics in real-time multimedia QoS measurement. The first is how to obtain more precise measurement data, in particular, how to overcome problems such as clock skew and synchronization. The second topic is the modeling of network delay and loss. The n -state extended Gilbert model is proposed as a general loss model. It is a generalization of 2-state Gilbert model, and yet less complex than an n th order Markov model that requires 2^n states. The *loss distance* and *noticeable loss rate* defined by IPPM working group is a good measure for capturing burstiness between consecutive loss runs. The conditional cumulative distribution function (ccdf) is proposed to capture the temporal dependency in network delays. Observations of our Internet packet traces have suggested that Internet packet losses are bursty, and the extended Gilbert model is suitable for capturing this burstiness. The ccdf we computed from the traces shows that when previous delays are high, the next delay is also very likely to be high. The end result is, after applying playout delay adjustment and optionally FEC, the final loss pattern still fits with the extended Gilbert model. That is, the final losses are still burstier than random losses. Particularly if FEC is not employed and jitter is low, the late losses and unrecovered losses often merge into larger loss bursts. In brief, the final loss pattern after playout is still bursty, but how it affects perceptual quality to the user is not well understood. The perceptual quality depends on the type of media, codecs, media content, etc. Our future work is to perform subjective listener tests to examine how loss burstiness actually relate to perceptual quality.

A Software Packages for Delay and Loss Measurement and Modeling

In the process of examining measurement techniques and modeling of delay and loss, we have written various C/C++ programs and shell scripts to perform these tasks. The whole software package is made available

burst length	late-loss only	un-recovered loss	total loss	merged bursts
1	7	223	230	227
2	4	62	66	66
3	0	10	10	10
4	0	4	4	4
5	0	2	2	1
6	0	3	3	2
7	0	2	2	4
11	0	1	1	1

(a) Without FEC, using Exp-Avg playout

burst length	late-loss only	un-recovered loss	total loss	merged bursts
1	25	22	47	43
2	8	10	18	18
3	0	5	5	6
4	0	5	5	5
5	0	2	2	2
6	0	4	4	3
7	0	0	0	1
10	0	1	1	1

(b) With FEC, using (5,3) Reed-Solomon code and virtualized Exp-Avg

burst length	late-loss only	un-recovered loss	total loss	merged bursts
1	0	223	223	223
2	0	62	62	62
3	0	10	10	10
4	0	4	4	4
5	0	2	2	2
6	0	3	3	3
7	0	2	2	2
11	0	1	1	1

(c) Without FEC, using Prev-Opt

burst length	late-loss only	un-recovered loss	total loss	merged bursts
1	0	22	22	22
2	0	10	10	10
3	0	5	5	5
4	0	5	5	5
5	0	2	2	2
6	0	4	4	4
10	0	1	1	1

(d) With FEC, using (5,3) Reed-Solomon code and virtualized Prev-Opt

Table 2: Effect of playout control on final loss burstiness, CU-UMass Sep 1997 trace

at: http://www.cs.columbia.edu/wenyu/research/qos_tools/ Please refer to the “README” file under this URL for instructions on installation and usage.

A.1 Packet Trace Generator

The tool “ott_mon” is a program that generates one-way and/or two-way packet traces. The user must start the server program on both the sender and receiver. Then the user runs the client program to create and start a test that generates packet traces.

The traces are stored as ascii files, with a few lines of header information, including IP addresses of sender and receiver, packet interval, packet size, etc. “ott_mon” does not perform clock synchronization and de-skewing. That task is performed in the next tool.

A.2 Clock Synchronization and De-skewing Tool

The second tool performs several tasks. It sorts the traces by packet sequence number, does clock synchronization and de-skewing. It can also select to process only the first N packets.

A.3 Delay and Loss Modeling Tool

The third tool takes the synchronized traces as input, and generates several output files. One is the unconditional delay pdf and cdf. The second is the conditional cdf (ccdf). The third output is the conditional pdf of next delay if previous delay falls in a specified range. The program can also specify the lag for ccdf

and conditional pdf. The fourth output is loss burst length distribution, which then are fit to both a 2-state Gilbert model and an extended Gilbert model.

A.4 Playout Delay Adjustment Simulation Tool with Optional FEC

This program simulates a selected playout delay control algorithm on a packet trace, and outputs the final loss burst length distribution. Currently two algorithms are implemented: Exp-Avg (exponential average) and Prev-Opt (previous optimal). FEC is also selectable. Currently the program implements (5,3) Reed-Solomon codec.

A.5 Simulation of Queueing System Delay Dependency Based on *Simul*

Finally, a program using the *Simul* [41] simulation package is written to simulate the various queueing systems shown in Figure 10 - 13.

References

- [1] Arnold O. Allen. *Probabilities, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press, 1990.
- [2] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way delay metric for IPPM. Request for Comments (Proposed Standard) 2679, Internet Engineering Task Force, September 1999.
- [3] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way packet loss metric for IPPM. Request for Comments (Proposed Standard) 2680, Internet Engineering Task Force, September 1999.
- [4] G. Almes, S. Kalidindi, and M. Zekauskas. A round-trip delay metric for IPPM. Request for Comments (Proposed Standard) 2681, Internet Engineering Task Force, September 1999.
- [5] Jean-Chrysostome Bolot. Characterizing end-to-end packet delay and loss in the Internet. *Journal of High Speed Networks*, 2(3):305-323, 1993.
- [6] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley. Adaptive FEC-Based error control for interactive audio in the internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.
- [7] Jean-Chrysostome Bolot and Andres Vega Garcia. Control mechanisms for packet audio in the internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, San Fransisco, California, March 1996.
- [8] Paul T. Brady. A technique for investigating on-off patterns of speech. *Bell System Technical Journal*, 44(1):1 - 22, January 1965.
- [9] Paul T. Brady. Effects of transmission delay on conversational behavior on echo-free telephone circuits. *Bell System Technical Journal*, 50(1):115-134, January 1971.
- [10] Ramon Caceres, Nick Duffield, Joseph Horowitz, Don Towsley, and Tian Bu. Multicast-based inference of network-internal characteristics: Accuracy of packet loss estimation. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.
- [11] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. Measurement considerations for assessing unidirectional latencies. Technical Report UCSD Report CS92-252, SDSC Report GA-A21018, Computer Systems Laboratory, UCSD, San Diego, California, 1993. also *Journal of Internetworking*, 4(3), September 1993.

- [12] C. Demichelis and P. Chimento. Instantaneous packet delay variation metric for IPPM. Internet Draft, Internet Engineering Task Force, June 1999. Work in progress.
- [13] Prabandham M. Gopal, J. W. Wong, and J. C. Majithia. Analysis of playout strategies for voice transmission using packet switching techniques. *Performance Evaluation*, 4(1):11–18, February 1984.
- [14] Roch Gurin and Henning Schulzrinne. Network quality of service. In Ian Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, California, 1998.
- [15] Vicky Hardman, Angela Sasse, Mark Handley, and Anna Watson. Reliable audio for use over the internet. In *Proc. of INET'95*, Honolulu, Hawaii, June 1995.
- [16] International Telecommunication Union (ITU). Transmission systems and media, general recommendation on the transmission quality for an entire international telephone connection; one-way transmission time. Recommendation G.114, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, March 1993.
- [17] Wenyu Jiang. Detecting and measuring asymmetric links in an IP network. Technical Report CUCS-009-99, Columbia University, New York, New York, January 1999.
- [18] Michael B. Jones and John Regehr. Issues in using commodity operating systems for timer-dependent tasks: Experiences from a study of windows NT. In *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, England, July 1998. At <http://www.nossdav.org/1998/papers/nossdav98-079.ps.gz>.
- [19] Leonard Kleinrock. *Queueing Systems — Computer Applications*, volume 2. Wiley-Interscience, New York, New York, 1976.
- [20] R. Koodli and R. Ravikanth. One-way loss pattern sample metrics. Internet Draft, Internet Engineering Task Force, June 1999. Work in progress.
- [21] Thomas Kostas and Michael S. Borella. Analysis of internet delay and loss measurements. In *Network Interop*, 1998.
- [22] Thomas J. Kostas, Michael S. Borella, Ikhtlaq Sidhu, Guido M. Schuster, Jacek Grabiec, and Jerry Mahler. Real-time voice over packet-switched networks. *IEEE Network*, 12(1):18–27, January 1998.
- [23] N. F. Maxemchuk and S. Lo. Measurement and interpretation of voice traffic on the internet. In *Conference Record of the International Conference on Communications (ICC)*, Montreal, Canada, June 1997.
- [24] David L. Mills. On the accuracy and stability of clocks synchronized by the network time protocol in the internet system. *ACM Computer Communication Review*, 20(1):65–75, January 1990.
- [25] David L. Mills. Network time protocol (version 3) specification, implementation. Request for Comments (Draft Standard) 1305, Internet Engineering Task Force, March 1992.
- [26] Sue Moon, Paul Skelly, and Don Towsley. Estimation and removal of clock skew from network delay measurements. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.
- [27] Sue B. Moon, Jim Kurose, Paul Skelly, and Don Towsley. Correlation of packet delay and loss in the internet. Technical report, University of Massachusetts, January 1998.
- [28] Sue B. Moon, Jim Kurose, and Don Towsley. Packet audio playout delay adjustment algorithms: performance bounds and algorithms. Research report, Department of Computer Science, University of Massachusetts at Amherst, Amherst, Massachusetts, August 1995.

- [29] Amarnath Mukherjee. On the dynamics and significance of low frequency components of internet load. Technical Report MIS-CIS-92-83, University of Pennsylvania, December 1992.
- [30] Andrew Odlyzko. The economics of the internet: Utility, utilization, pricing, and quality of service. Technical report, AT&T Labs - Research, Florham Park, New Jersey, July 1998.
- [31] Advanced Org. Participants guide to the Surveyor daily summary reports. <http://telesto.advanced.org/kalidindi/STR/surveyor-reports-guide.html>.
- [32] Advanced Org. Surveyor home page. <http://www.advanced.org/surveyor/>.
- [33] Vern Paxson. End-to-end routing behavior in the internet. In *SIGCOMM Symposium on Communications Architectures and Protocols*, Stanford, California, August 1996.
- [34] Colin Perkins, Orion Hodson, and Vicky Hardman. A survey of packet loss recovery techniques for streaming audio. *IEEE Network*, 12(5):40–48, September 1998.
- [35] J. Postel. Internet control message protocol. Request for Comments (Standard) 792, Internet Engineering Task Force, September 1981.
- [36] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning 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.
- [37] Jonathan D. Rosenberg. G.729 error recovery for internet telephony. Technical report, Columbia University, 1997. At <http://www.cs.columbia.edu/~jdrosen/e6880/index.html>.
- [38] Jonathan D. Rosenberg, Lili Qiu, and Henning Schulzrinne. Integrating packet fec into adaptive voice playout buffer algorithms on the internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, 2000.
- [39] H. Sanneck, G. Carle, and R. Koodli. A framework model for packet loss metrics based on loss run-lengths. In *SPIE/ACM SIGMM Multimedia Computing and Networking Conference*, January 2000.
- [40] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. Request for Comments (Proposed Standard) 1889, Internet Engineering Task Force, January 1996.
- [41] Henning Schulzrinne. SIMUL simulation system manual. internal memorandum, February 1989.
- [42] Dallas Semiconductor. DS1553: 64k NV Y2KC Timekeeping RAM Manual. Technical report, Dallas Semiconductor, 1999. At <http://www.dalsemi.com/DocControl/PDFs/1553.pdf>.
- [43] W. R. Stevens. *TCP/IP Illustrated: The Protocols*, volume 1. Addison-Wesley, Reading, Massachusetts, December 1993.
- [44] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November/December 1997.
- [45] Maya Yajnik, Jim Kurose, and Don Towsley. Packet loss correlation in the Mbone multicast network. In *Proceedings of Global Internet*, London, England, November 1996.
- [46] Maya Yajnik, Sue Moon, Jim Kurose, and Don Towsley. Measurement and modelling of the temporal dependence in packet loss. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.