Estimation and Removal of Clock Skew from Network Delay Measurements*

Sue B. Moon†, Paul Skelly‡, Don Towsley†

†Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{sbmoon,towsley}@cs.umass.edu

‡GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254
pskelly@gte.com

Technical Report 98-43
Department of Computer Sicnece
University of Massachusetts at Amherst

October 1998

Abstract

Packet delay and loss traces are frequently used by network engineers, as well as network applications, to analyze network performance. The clocks on the end-systems used to measure the delays, however, are not always synchronized, and this lack of synchronization reduces the accuracy of these measurements. Therefore, estimating and removing relative skews and offsets from delay measurements between sender and receiver clocks are critical to the accurate assessment and analysis of network performance. In this paper we introduce a linear programming based algorithm to estimate the clock skew in network delay measurements and compare it with three other algorithms. We show that our algorithm has the time complexity of $O(N)$, leaves the delay after the skew removal positive, and and is robust in the sense that the error margin of the skew estimate is independent of the magnitude of the skew. We use traces of real Internet delay measurements to assess the algorithm, and compare its performance to that of three other algorithms. Furthermore, we show through simulation that our algorithm is unbiased, and that the sample variance of the skew estimates is better(smaller) than existing algorithms.
Keywords: clock skew, clock ratio, end-to-end delay, delay measurement.

1 Introduction

End-to-end delay and loss traces are frequently used in analyzing the network performance. The accuracy of such measurements is important in several reasons. First, end-to-end measurements may be the only way of measuring network performance, especially when there is no provision inside the network to provide end-systems with information about the current status of the network. The current Internet has no mechanism for providing feedback on network congestion to end-systems at the IP layer, and neither does IPv6 [DH95]. Second, protocols and applications that behave adaptively at the end-system base their control on observed network performance, and it is critical that they obtain correct measurements.

*This research was supported in part by funding from GTE Laboratories, Inc., and by the National Science Foundation under Grant No NCR-9508274. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Packet loss can be detected if a sender puts a sequence number on every packet it sends out, and the receiver sees a gap in the sequence numbers of packets arriving within a reasonable amount of time. For delay measurements, a sender needs to add timestamps to packets for a receiver to gather delay information [SCFJ96]. Since the clocks at both end-systems are involved in measuring delay, the synchronization between the two clocks becomes an issue in the accuracy of delay measurement. The Network Time Protocol(NTP) [Mil92b] is widely used on the Internet for clock synchronization, and provides accuracy in the order of milliseconds under reasonable circumstances. The accuracy, however, is not guaranteed, and not all hosts on the Internet support it.

Packet loss and delay are crucial in understanding the performance and reliability of the Internet. To provide unbiased and quantitative measures of performance, there has been much effort to define one-way loss and delay metrics [PAMM98]. To obtain an accurate measurement of one-way delay, errors and uncertainties related to clocks need to be accounted for. When two clocks involved in the measurement run at different frequencies (that is, have a clock skew), inaccuracies are introduced in the measurement. In this paper we focus on filtering out the effects of clock skew specifically in one-way delay measurements.

The rest of the paper is organized as follows. In Section 2 we present a typical delay trace that motivated us to design a skew estimation algorithm. In Section 3 we define the terms needed to describe clock behavior, and introduce the notation to be used in the remainder of the paper. In Section 4 we formalize the clock synchronization problem between two hosts, and show how the skew and offset affect the delay measurements. Then we list several desirable properties expected of a skew estimation algorithm. We introduce our skew estimation algorithm based on a linear programming technique in Section 5, and three existing algorithms in Section 6. In Section 7 we compare the four algorithms presented in this paper, and in Section 8 discuss how this work can be extended to the cases where the skew is not constant. We conclude the paper in Section 9.

2 Motivation

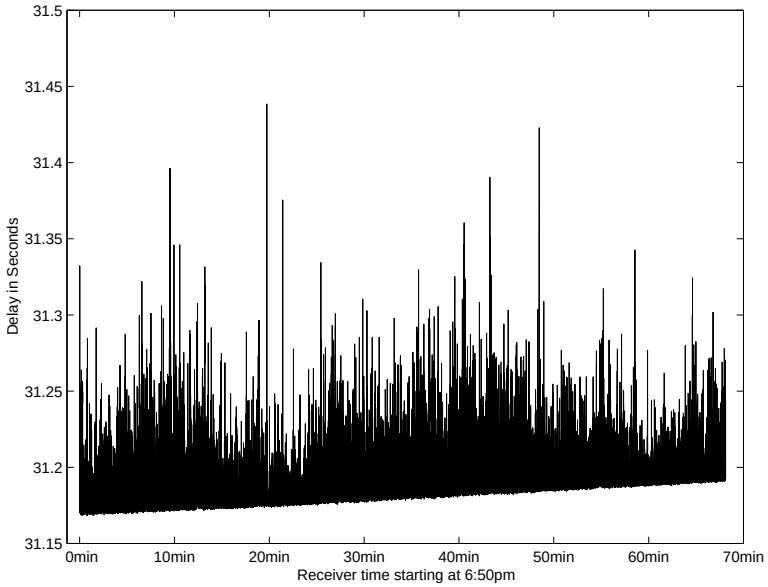


Figure 1: Trace 1

Before we introduce the skew estimation algorithms, let us first examine a sample of one-way delay

measurements. This sample illustrated in Figure 1 is taken from a trace described in Section 7.4 (Trace 1 in Table 2). The x-axis is the sender timestamp, and the y-axis is the delay calculated by subtracting the sender timestamp from the receiver timestamp. The measured delay lies in the range of 31.15 to 31.5 seconds. The measured delay is not the actual end-to-end delay, but includes the clock offset between the two clocks plus the end-to-end delay. Clock offset is the difference in time, and skew is the difference in clock speed. We defer the formal definitions of offset and skew to Section 3.

In Figure 1 the delay shows an increasing trend of about 100 milliseconds over the duration of 70 minutes at the receiver. It is significant enough to distort performance metrics such as the average and autocorrelation of end-to-end delay. While one might imagine that the ever increasing minimum delay is due to increasing congestion and queueing delay, this is unlikely as the minimum observed delay increases over time. Instead, the linear increase in delay attests to a constant speed difference between two clocks.

The end-to-end delay consists of transmission and propagation delays plus variable queueing delay. When all of the packets go through the same route to the receiver, they have the same propagation delay, and, if they have the same size, the transmission delay also is the same. Even if the packets go through the same route, and have the same size, the packets experience different levels of queueing inside the network. This is what causes the variability in the end-to-end delay.

Previous work by Paxson[Pax97, Pax98] addresses problems in delay measurements due to clock adjustments and rate mismatches. It uses forward and reverse path measurements of delay between a pair of hosts to deal with clock synchronization problems, such as relative offset and skew. Many applications, however, see only one-way delay, and still have to deal with the clock synchronization problems in packet delay. One-way measurements alone are not enough to infer the clock offset, and we cannot distinguish the clock offset from the fixed portion of end-to-end delay. For example, in the figure shown above, it is difficult to tell how much of the 31.15 seconds is due to the time difference between clocks and the fixed transmission and propagation delay, without the availability of more information. Due to this lack of information in one-way delay, we focus on the variable portion in one-way delay measurements.

The variable queueing delay serves a very important role in network and application design. Continuous-media applications such as audio and video need to absorb the delay jitter perceived at the receiver for smooth playout of the original stream [RKTS94, MKT98, DS95]. Determining the correct amount of buffering, and reconstructing the original timing is crucial to the performance of continuous-media applications. The variable queueing delay is also useful in monitoring the network performance at the edges of the network; the transmission and propagation delay is fixed per route, and does not convey any information about the dynamic changes inside the network when packets follow a fixed route.

3 Background

3.1 Clock terminology

In this section we introduce the terminology we use to describe clock behavior. Let us begin by defining a *clock*. It is a piecewise continuous function that is twice differentiable except on a finite set of points:

$$C : \mathcal{R} \longrightarrow \mathcal{R}$$

where $C'(t) \equiv dC(t)/dt$ and $C''(t) \equiv d^2C(t)/dt^2$ exist everywhere except for $t \in P \subset \mathcal{R}$ where $|P|$ is finite.

A "true" clock reports "true" time at any moment, and runs at a constant rate. Let C_t denote the "true" clock; it is the identity function given below,

$$C_t(t) = t \text{ and } P_t = \emptyset$$

In the paper, we use the following nomenclature from [Mil92a, Mil92b] to describe clock characteristics. Let C_a and C_b be two clocks:

- **offset**: the difference between the time reported by a clock and the "true" time; the offset of C_a is $(C_a(t) - t)$. The offset of the clock C_a relative to C_b at time $t \geq 0$ is $C_a(t) - C_b(t)$.
- **frequency**: the rate at which the clock progresses. The frequency at time t of C_a is $C_a'(t)$.
- **skew**: the difference in the frequencies of a clock and the "true" clock. The skew of C_a relative to C_b at time t is $(C_a'(t) - C_b'(t))$.
- **drift**: The drift of clock C_a is $C_a''(t)$. The drift of C_a relative to C_b at time $t \geq 0$ is $(C_a''(t) - C_b''(t))$.

Two clocks are said to be *synchronized* at a particular moment if both the relative offset and skew are zero. When it is clear that we refer to clocks that are not the "true" clock in our discussion, we simply refer to relative offset and relative skew as offset and skew, respectively.

It is sometimes convenient to compare the frequency ratio between two clocks instead of the skew. This is captured by the following definition.

- **clock ratio**: the frequency ratio between a clock and the "true" clock; the ratio of C_a is $C'(a)$. The ratio of C_a relative to C_b at time t is $C_a'(t)/C_b'(t)$.

Let C_a and C_b have constant frequencies, and α and δ be the clock ratio and skew of C_b relative to C_a, respectively. $\alpha = C_b'/C_a'$ and $\delta = C_b' - C_a'$. Then the relation between the clock ratio and the skew is:

$$\delta = C_b' - C_a' = \alpha C_a' - C_a' = (\alpha - 1)C_a' \tag{1}$$

From now on, we assume that the sender and receiver clocks have constant frequencies, and thus their skew and clock ratio are constant over time; we use them interchangeably, and use (1) whenever necessary to convert one from the other.

3.2 Time duration consistent with a clock

In the previous section, we have defined a *clock* and terms relevant to its behavior. In this section we look at how a *time duration* is measured according to a clock. Let $\Delta(t_1, t_2, C_a)$ denote the time that has passed according to C_a between t_1 and t_2 of the "true" clock. Since a clock is a piecewise continuous function, we define the time duration as:

$$\begin{aligned}
\Delta(t_1, t_2, C_a) &\equiv \int_{t_1}^{t_2} C_a' dt \\
&= \int_{t_1}^{p_1} C_a' dt + \int_{p_1}^{p_2} C_a' dt + \cdots + \int_{p_{n-1}}^{p_n} C_a' dt + \int_{p_n}^{t_2} C_a' dt \\
&\text{where } P_a \bigcap (t_1, t_2) = \{p_1, p_2, \ldots, p_n\} \text{ and } t_1 < p_1 < p_2 < \cdots < p_n < t_2, 1 \leq i \leq n
\end{aligned}$$

If $P_a \bigcap (t_1, t_2) = \phi$, then

$$\Delta(t_1, t_2, C_a) = \int_{t_1}^{t_2} C_a' dt = C_a(t_2) - C_a(t_1) \tag{2}$$

When two clocks are not synchronized and, more specifically, have different frequencies, time duration measured with one clock will be different from the other. We say that a time duration measured with a clock is **consistent** with any other clock of the same frequency and any offset. If two clocks have a non-zero skew, time measured on one clock will not be consistent with the other clock.

We have modeled a clock as a piecewise continuous function in order to take into account the restrictions of real clocks. The resolution of a clock on a computer system is the smallest unit by which the clock's time is updated, and is greater than zero. At best, a clock in a computer is a step function with increments at every unit of its time resolution. We consider the time reports by a real clock with a fixed minimum resolution as samples of a continuous function at specific moments, and thus circumvent the discretization effect of the real clock. Another problem a real clock poses is the abrupt time adjustment possible through a time resetting system call. Some systems that do not run NTP [Mil92b] have a very coarse-grain (in the order of hours) synchronization mechanism in the `cron` table. The time adjustment in such a case can be several orders of magnitude larger than the usual increment of the clock resolution, and the time can even be set backward. The piecewise nature of a clock function accommodates the abrupt time adjustment.

When a delay measurement involves more than one clock, the synchronization between those clocks has a tremendous impact on the accuracy of the measurement. Let us consider a case of measuring a packet delay between two hosts. The sender adds a timestamp to a packet when it leaves the sender, and the receiver records the time the packet arrives at the receiver. When the two host clocks are perfectly synchronized, the difference between the two timestamps is the end-to-end network delay experience by that packet. If the clocks on the two hosts have a non-zero offset, but no skew, the difference between two timestamps includes not only the end-to-end delay, but also the offset. Given only a one-way measurement, we cannot distinguish the offset from the measurement, unless we are given the network delay, which is what we intended to measure in the first place. If the clocks have a non-zero skew, not only is the end-to-end delay measurement off by an amount equal to the offset, but it also gradually increases or decreases over time depending on whether the sender clock runs slower or faster than the receiver clock.

In the following sections we formalize the clock synchronization problem outlined above, and show how to remove the clock skew in measurements.

4 Basics of a Skew Estimation Algorithm

In the previous section we have defined a clock, and what is meant for a delay to be consistent with a clock. In this section we discuss the estimation and removal of the effects of clock skew in delay measurements. We first derive how much the clock skew contributes to the measured end-to-end delay if the skew is non-zero and constant. This derivation provides a basis for the discussion of several desirable properties for skew estimation algorithms.

4.1 Delay measured between two clocks

From Section 4.1, if the clock ratio between the sender and receiver clocks is greater than or less than 1, network delays will appear to become longer or shorter over the course of a measurement period. The purpose of removing this effect of skew on the delay measurements is to transform the delay measurements so that they are consistent with a single clock. In our work, we have chosen to make the delay measurements consistent with the receiver clock. When there is no provision at the receiver to the "true" clock, the only clock the receiver has access to is its own clock. It is thus natural to measure one-way delay according to the receiver clock. Without loss of generality, we fix the **receiver clock** to be the **"true clock"**, i.e. $C'_r(t) = 1$ and $\alpha = C'_s(t)$, and derive the notations below as such.

For packets of different sizes, the clock skew may not be distinguishable from the delay trend, if any. For example, if the packet size grows over time and the route from the sender to the receiver is fixed, then the transmission delay gradually increases, and it is hard to distinguish a skew from this delay trend. Thus we assume all the packets have the same size in the following sections.

Let us now introduce the terminology for clocks, timestamps, and delays used in measurements.

- C_s: sender clock.
- C_r: receiver clock.
- N: number of packets that arrive at the receiver.
- l_i: timestamp of the i-th packet leaving the sender according to C_r, $i = 1, 2, \ldots, N$.
- t_i^s: timestamp of the i-th packet leaving the sender according to C_s, $i = 1, 2, \ldots, N$; $t_i^s = C_s(l_i)$.
- t_i^r: timestamp of the i-th packet arriving at the receiver according to C_r, $i = 1, 2, \ldots, N$.
- d_i: end-to-end delay measurement of the i-th packet, $i = 1, 2, \ldots, N$;

$$d_i = t_i^r - t_i^s \tag{3}$$

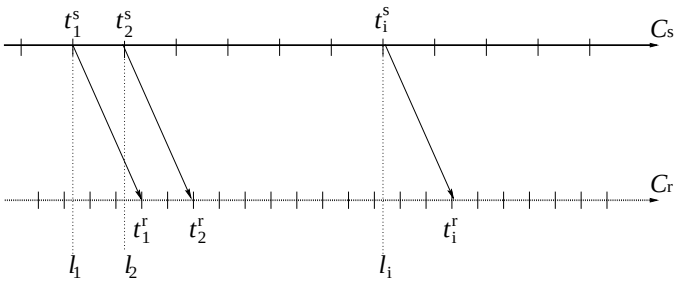


Figure 2: Timing chart showing constant delay

Figure 2 shows the timing between C_s and C_r when C_s runs half the speed of C_r and all the packets experience the same network delay. The end-to-end delay of the i-th packet consistent with C_r is $t_i^r - l_i$. l_i, however, is not known at the receiver, and we calculate d_i using t_i^s and t_i^r. As a result, in this case, the end-to-end delay is consistent with neither C_s nor C_r. To make it consistent with C_r, we need to determine the skew of C_r relative to C_s, and remove it from the measurement d_i.

4.2 Clock Skew in Delay Measurements

When there is a constant clock skew between two clocks, the clock offset between them gradually increases or decreases over time, depending on the sign of the skew. The amount of increase or decrease in the clock offset is proportional to the time duration of observation. The longer you observe, the larger the offset is. We use this amount of offset change to estimate the clock skew. Thus it is more convenient to use timestamps relative to a specific point in time, such as the departure and arrival times of the first packet, than absolute timestamps. Below we introduce relative timestamps at the sender and the receiver.

- $\tilde{t}_i^s$: time duration between the first and the i-th packets' departures at the sender consistent with C_s.

$$\begin{aligned} \tilde{t}_1^s &= 0 \\ \tilde{t}_i^s &= \Delta(l_1, l_i, C_s) = t_i^s - t_1^s \end{aligned}$$

- $\tilde{t}_i^r$: time duration between the first and the i-th packets' arrivals at the receiver consistent with C_r.

$$\begin{aligned} \tilde{t}_1^r &= 0 \\ \tilde{t}_i^r &= t_i^r - t_1^r \end{aligned}$$

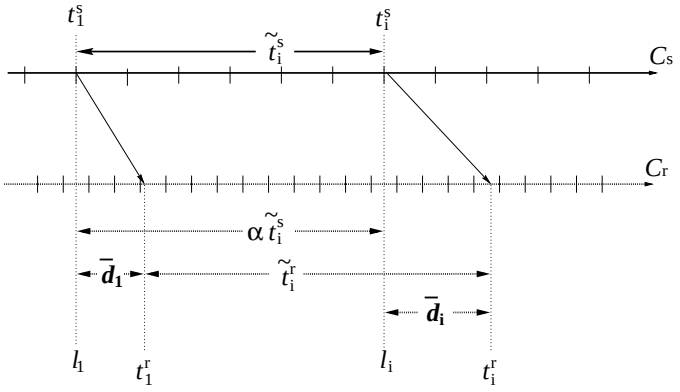


Figure 3: Timing chart showing variable delay

By (1) and (2),

$$\begin{aligned} \Delta(l_1, l_i, C_r) &= l_i - l_1 \\ &= \alpha\Delta(l_1, l_i, C_s) = \alpha\tilde{t}_i^s \end{aligned} \tag{4}$$

Figure 3 illustrates the relationship between $\Delta(l_1, l_i, C_r)$ and $\tilde{t}_i^s$ on a timing chart. The quantities $\bar{d}_1$ and $\bar{d}_i$ shown in the figure are defined as below.

- $\bar{d}_i$: end-to-end delay consistent with C_r.

$$\begin{aligned} \bar{d}_1 &= \Delta(l_1, t_1^r, C_r) = t_1^r - l_1 \\ \bar{d}_i &= \Delta(l_i, t_i^r, C_r) = t_i^r - l_i \\ &= t_i^r - l_1 - \alpha\tilde{t}_i^s = (t_i^r - t_1^r) + (t_1^r - l_1) - \alpha\tilde{t}_i^s \\ &= \tilde{t}_i^r + \bar{d}_1 - \alpha\tilde{t}_i^s \end{aligned} \tag{5}$$

The quantity $\bar{d}_i$, however, is not obtainable directly from measured timestamps, due to the skew between the sender and receiver clocks. The quantity that is obtainable from actual timestamps is the following.

- $\tilde{d}_i$: delay calculated from $\tilde{t}_i^s$ and $\tilde{t}_i^r$.

$$\begin{aligned} \tilde{d}_i &= \tilde{t}_i^r - \tilde{t}_i^s = \tilde{t}_i^r + \bar{d}_1 - \alpha\tilde{t}_i^s + (\alpha - 1)\tilde{t}_i^s - \bar{d}_1 \\ &= \bar{d}_i + (\alpha - 1)\tilde{t}_i^s - \bar{d}_1 \end{aligned} \tag{6}$$

The goal of estimating and removing the clock skew is to obtain $\bar{d}_i$ from the actual delay measurement, $\tilde{d}_i$. From (3) and (6), we note that the difference between d_i and $\tilde{d}_i$ is:

$$d_i - \tilde{d}_i = t_1^r - t_1^s.$$

Also note in (6) that $\tilde{d}_i$ is different from $\bar{d}_i$ by $(\alpha - 1)\tilde{t}_i^s$ minus a constant $\bar{d}_1$. If $\alpha > 1$, $(\alpha - 1)\tilde{t}_i^s$ grows linearly with $\tilde{t}_i^s$, and thus $\tilde{d}_i$ gets larger. This is what contributed to an increasing trend we observed in Figure 1. Finally, from (6), it is obvious that the measured network delays can be made consistent with C_r given α and $\bar{d}_i$ according to:

$$\bar{d}_i = \tilde{d}_i - (\alpha - 1)\tilde{t}_i^s + \bar{d}_1. \tag{7}$$

Let $\hat{\alpha}$ and $\hat{\beta}$ be the estimates for α and $\bar{d}_1$. Then the delay after the skew removal, $\hat{d}_i$, is:

$$\hat{d}_i = \tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta} \tag{8}$$

4.3 Desirable Properties for Skew Estimation Algorithms

Before we delve into details of the skew estimation algorithm, we first state the desirable properties that any such algorithm should exhibit. We use these properties later as a basis of comparing different estimation algorithms.

To formally state the properties, we hereby introduce the notations for an estimation algorithm and estimates parametrized by the estimation algorithm. Let $\mathcal{A}$ be a skew estimation algorithm. We make the same assumption as in Section 4.1 that the skew between the sender and the receiver clocks is constant, and the receiver clock is the "true" clock. Given a set of end-to-end delays, $\mathcal{D} = \{\bar{d}_i\}_{i=1}^{N}$, which are predetermined and fixed, and also consistent with the "true" clock, we know that the delay measurements, $\tilde{d}_i$'s, are equal to $\bar{d}_i$ if there is no clock skew between the two hosts ($\alpha = 0$); $\tilde{d}_i$'s are different from $\bar{d}_i$ if the clock skew is not zero ($\alpha \geq 0$). In that sense $\tilde{d}_i$ is dependent on the clock skew, α, and $\bar{d}_i$, and is noted $\tilde{d}_i(\alpha, \mathcal{D})$.

We define $\hat{\alpha}_{\mathcal{A}}(\alpha, \mathcal{D})$ and $\hat{\beta}_{\mathcal{A}}(\alpha, \mathcal{D})$ as the estimates of α and $\bar{d}_1$, respectively, delivered by algorithm $\mathcal{A}$, when given $\tilde{d}_i, 1 \leq i \leq N$ and α. Below is a list of desirable properties that should be exhibited by algorithm $\mathcal{A}$.

- **Property 1**: The time and space complexity of algorithm $\mathcal{A}$ should be linear in N. The computational complexity of an algorithm in terms of time and space is an important metric in assessing the performance and applicability of the algorithm. We will compare the *time complexity* of skew estimation algorithms as a function of the number of delay measurements.

- **Property 2**: Since the purpose of the skew estimation is to remove the skew from delay measurements, it is desirable that *the delays be non-negative after the skew is removed*.

$$\hat{d}_i = \tilde{d}_i(\alpha, \mathcal{D}) - (\hat{\alpha}_{\mathcal{A}}(\alpha, \mathcal{D}) - 1)\tilde{t}_i^s + \hat{\beta}_{\mathcal{A}}(\alpha, \mathcal{D}) \geq 0$$

- **Property 3**: The skew estimation algorithm should be robust in the sense that it is not affected by the magnitude of the actual skew. That is, the difference between the estimate and the actual skew should be independent of the actual skew. Under the same network condition, the skew estimate for different skews should exhibit the same margin of error from the actual skew, no matter how small or large the skew is. We state this property as follows:

$$\hat{\alpha}_{\mathcal{A}}(\alpha, \mathcal{D}) - \alpha = \hat{\alpha}_{\mathcal{A}}(1, \mathcal{D}) - 1 \tag{9}$$

  for any $\alpha > 0$.

In the following section, we introduce a new algorithm based on linear programming to estimate α in delay measurements, and use the result to remove the skew from one-way delay measurements to make them consistent with the receiver clock. In the next two sections, we focus on a simple case where the clock skew is constant, and defer the discussion of a time-varying skew case to Section 8.

5 Linear Programming Algorithm

Figure 1 illustrates a trace where the skew between two clocks was nearly constant over the measurement duration. Looking at the figure, one is tempted to pick up a ruler, draw a line that skims through the bottom of the mass of the scatter-plot, measure the angle between the line and the x-axis, and calculate the skew using simple trigonometry. This approach is hard to automate, and invites human errors that are untraceable. A second thought would be to pick the first and last data points, and draw a line between them. The accuracy

of this approach, however, can be easily thrown off, since delay has formidable variability that is in the order of magnitude bigger than the skew all through the measurement duration. Our approach is to fit a line that lies under all the data points, but as closely to them as possible.

We have formulated the above idea as a linear programming problem. The condition that the line should lie under all the data points forms the first part of our linear programming problem, and defines the feasible region for a solution; the objective function of the linear programming problem is to minimize the sum of the distances between the line and all the data points on the y-axis.

5.1 Algorithm

Having presented our intuition behind the algorithm, we now introduce the algorithm formally. The goal of the skew estimation algorithm is to estimate the clock ratio α given $\tilde{t}_i^s$ and $\tilde{d}_i$. The output of the skew estimation algorithm is: $\hat{\alpha}$ and $\hat{\beta}$, where $\hat{\alpha}$ is the estimate of α, and $\hat{\beta}$ is the estimate of $\bar{d}_1$. We return to the interpretation of $\hat{\beta}$ the end of this section. If we estimate both α and $\bar{d}_1$ correctly, then we can subtract $(\alpha - 1)\tilde{t}_i^s - \bar{d}_1$ from $\tilde{d}_i$, and obtain $\bar{d}_i$, which is the end-to-end delay consistent with C_r and free of clock skew. Even when the estimates $\hat{\alpha}$ and $\hat{\beta}$ are not exactly the same as α and $\bar{d}_1$, we still want the resulting end-to-end delay to be non-negative, after the skew is removed. When we formulate our skew estimation as a linear programming problem, this condition defines the feasible region where a solution should lie.

$$\tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta} \geq 0, \quad 1 \leq i \leq N \tag{10}$$

There are infinitely many pairs of $\hat{\alpha}$ and $\hat{\beta}$ that satisfy the condition above, if the feasible region from (10) is not trivial. Our objective function to minimize the distance between the line and all the delay measurements is stated as:

$$\min\{\sum_{i=1}^{N} \left(\tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta}\right)\} \tag{11}$$

and is used to determine the solution to $\hat{\alpha}$ and $\hat{\beta}$ from (10).

One important point to note in (10) is that the estimated end-to-end delay of $\bar{d}_i$, calculated as $(\tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta})$ once $\hat{\alpha}$ and $\hat{\beta}$ are obtained, will be greater than zero, instead of being greater than $\min_i \bar{d}_i$. Thus $\hat{\beta}$ is actually an estimate of ($\bar{d}_1$ + $\min_i \bar{d}_i$). The resulting delay of $\tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta}$ is not the end-to-end delay, but rather the variable portion of the end-to-end delay.

In the following sections, we look into other algorithms that can be used in skew estimation, and compare them with our linear programming algorithm in terms of the properties listed in Section 4, and their performance in actual and synthetic measurements.

6 Other Algorithms

6.1 Paxson's algorithm

In [Pax97, Pax98], Paxson designed an algorithm for removing a clock skew from a set of forward and reverse path delay measurements. In this section we briefly describe how we use his algorithm when given delays in only one direction. Assume that the input to the algorithm is the same as in the previous linear programming algorithm: $\tilde{d}_i$ and $\tilde{t}_i^s$, for $1 \leq i \leq N$. Readers are referred to [Pax97, Pax98] for more detail.

Paxson's algorithm is as follows:

- **Step 1.** Partition $\tilde{d}_i$'s into $\sqrt{N}$ segments, and pick the minimum delay measurement from each segment. The selected measurements are called the "de-noised" one-way transit times (OTTs).

- **Step 2.** Pick the median of the slopes of all possible pairs of the "de-noised" OTTs. If the median slope is negative, assume that the OTTs have a decreasing trend (here we assume a decreasing trend is detected).
- **Step 3.** Select the cumulative minima test from the "de-noised" OTTs (see [Pax97]), and test if the number of cumulative minima is large enough to show that the the decreasing trend found in Step 2 is probabilistically not likely, if there is no trend.
- **Step 4.** If it passes the cumulative minima test, pick the median from the slopes of all possible pairs of the cumulative minima: output it as the estimate of $\alpha - 1$. Otherwise, the algorithm concludes no skew, and outputs zero.

The core of Paxson's algorithm is the robust line fitting technique based on robust statistics[HMT83]. It uses the median as a robust estimate for the slope. As mentioned in [Pax97, Pax98], robust line fitting alone fails in estimating the slope of the trend due to the high variability in OTTs, and that is why the "de-noised" OTTs and cumulative minima are used in his algorithm.

6.2 Linear regression algorithm

Linear regression is a standard technique for fitting a line to a set of data points. It is optimal in the mean square sense if the network delays are normally distributed, but is not robust in the presence of outliers. As pointed out in [Pax97, Pax98], it is not a good choice for a skew estimation, even when applied to the "de-noised" OTTs above. Here we use it only as a reference algorithm that has no knowledge of the underlying behavior of delay measurements.

The linear regression algorithm in a skew estimation provides estimates of α and β that minimize the mean square error e in:

$$e = \sum_{i=1}^{N} \{\tilde{d}_i - (\hat{\alpha} - 1)\tilde{t}_i^s + \hat{\beta}\}^2. \tag{12}$$

6.3 Piecewise minimum algorithm

There is another simple algorithm to illustrate the difficulty in estimating the skew. It partitions the delay measurements into segments, picks a minimum from each segment, and connects them to obtain a concatenation of line segments. The minima are the same as the "de-noised" OTTs in Section 6.1. The resulting concatenation of line segments is the estimate of the skew, and is very unlikely to be a straight line.

When the skew is as obvious as in Figure 1, the resulting concatenation of line segments is close to a straight line, and can be used as a rough estimate. It, however, does not perform as well in other situations. We look at such cases later in Section 7.4

7 Comparison of the Four Algorithms

In this section we compare the four algorithms based on the desirable properties from Section 4. Also we apply the algorithms to actual delay measurements, and compare their performance by looking at the adjusted delays. Lastly, we use delay measurements from simulation to compare our approach to Paxson's algorithm.

7.1 Computational complexity

The time complexity of a linear programming problem of a finite dimension of two is proven to be $O(N)$ [Dye83, Meg83]. We have implemented a simple and efficient algorithm that exploits the fact that $\tilde{t}_i^s$'s are sorted in our specific problem. Refer to Appendix A for its pseudo code. Paxson's algorithm has $O(N \log N)$ complexity. Going back to the algorithm in Section 6.1, all the other steps have $O(N)$ or smaller complexity except for Step 2. The number of slopes of all possible pairs of the "de-noised" OTTs is $O(N)$, and finding a median of $O(N)$ numbers is $O(N \log N)$. The time complexity of the linear regression and piecewise minimum algorithms is $O(N)$.

7.2 Non-negative delay after the skew removal

In order to guarantee that the delay remains positive after the skew is removed, a skew estimation algorithm must estimate $\bar{d}_1$ correctly. The linear programming algorithm, however, is the only one that estimates $\bar{d}_1$ (or $\bar{d}_1 + \min_i \bar{d}_i$), as explained in Section 5. Paxson's original algorithm for skew estimation is for two-way measurements *after* the clock offset has been removed. The linear regression algorithm provides an estimate of $\hat{\beta}$. However this is just a y-intercept of the regression line which bears no relevance to the correct estimation of $\bar{d}_1$. The piecewise minimum algorithm outputs a concatenation of line segments, and the slopes of those line segments are skew estimates. The algorithm does not have any provision to guarantee that all the data points lie above the concatenation of line segments.

For the three algorithms that do not provide an estimate for $\bar{d}_1$ that ensures that delays are non-negative after the skew removal, we choose a $\hat{\beta}$ that satisfies the following condition for all $\hat{\alpha}$'s in each algorithm:

$$\max_{1 \le i \le N} \{\hat{\beta}_i : \tilde{d}_i - (\hat{\alpha}_i - 1)\tilde{t}_i^s + \hat{\beta}_i > 0\} \tag{13}$$

where $\hat{\alpha}_i = \hat{\alpha}$ and $\hat{\beta}_i = \hat{\beta}$ for $1 \le i \le N$ in Paxson's and linear regression algorithms; in the piecewise minimum algorithm $\hat{\alpha}_i$ and $\hat{\beta}_i$ are determined by the line segment to which $\tilde{d}_i$ and $\tilde{t}_i^s$ belong to.

7.3 Robustness

We will next demonstrate another property of the linear programming algorithm that distinguishes it from the other three algorithms: its performance, as measured by the difference between the estimate and the actual skew, depends only on the variability of the network delays, and not on the magnitude of the clock skew. It guarantees that the estimation algorithm performs reliably, in the sense that the margin of error remains the same, no matter how large the skew is.

We first show that the linear programming algorithm satisfies this property, and follow it with a discussion about other algorithms.

7.3.1 Linear Programming Algorithm

We use the same assumptions and notations for the skew estimation algorithm and estimates as in Section 4.3 when considering two different clock skews: a set of delays, $\mathcal{D} = \{\bar{d}_i\}_{i=1}^N$, where $\bar{d}_i$'s are fixed, and the clock ratio varies from one to some constant. It can be restated as follows. From one set of measurements to the other, nothing changes except for the frequency of the sender clock relative to the receiver clock. The receiver observes that the delay measurements, $\tilde{d}_i$'s, are different between two sets, but the end-to-end delays consistent with the receiver clock remain the same in both sets. We also note that $\tilde{t}_i^s$'s remain the same in both sets.

Consider the sender and receiver clocks are "true" clocks, and a set of packet delays, $\mathcal{D} = \{\bar{d}_i\}_{i=1}^N$, is consistent with the "true clock." Suppose that we measure those delays when the frequence of the sender

clock changes so that the skew is $\alpha \neq 1$. We have

$$\tilde{d_i}(1, \mathcal{D}) = \bar{d_i} - \bar{d_1} \quad (14)$$

$$\tilde{d_i}(\alpha, \mathcal{D}) = \bar{d_i} + (\alpha - 1)\tilde{t_i^s} - \bar{d_1} \quad (15)$$

from (6).

Let $\mathcal{A}$ be the linear programming algorithm, and consider the problem of determining $\hat{\alpha}$ and $\hat{\beta}$ when both clocks are "true" clocks. By (10), (11), and (14), the problem becomes minimizing

$$\sum_{i=1}^{N} \{\bar{d_i} - \bar{d_1} - (\hat{\alpha} - 1)\tilde{t_i^s} + \hat{\beta}\}$$

such that

$$\hat{\beta} \geq (\hat{\alpha} - 1)\tilde{t_i^s} - \bar{d_i} + \bar{d_1}, \quad \text{for} \quad 1 \leq i \leq N.$$

Let $\hat{\alpha}_{\mathcal{A}}(1, \mathcal{D})$ and $\hat{\beta}_{\mathcal{A}}(1, \mathcal{D})$ be the values that solve this problem.

Now define $\alpha^* = (\alpha + \hat{\alpha}_{\mathcal{A}}(1, \mathcal{D}) - 1)$, and substitute $\alpha^* - \alpha$ with $\hat{\alpha}_{\mathcal{A}}(1, \mathcal{D}) - 1$ above, and the above problem is now equivalent to choosing α^* and $\hat{\beta}$ that minimize

$$\sum_{i=1}^{N} \{\bar{d_i} - \bar{d_1} - (\alpha^* - \alpha)\tilde{t_i^s} - \hat{\beta}\}$$

such that

$$\hat{\beta} \geq (\alpha^* - \alpha)\tilde{t_i^s} - \bar{d_i} + \bar{d_1}, \quad \text{for} \quad 1 \leq i \leq N.$$

By (15), it is equivalent to choosing α^* and $\hat{\beta}$ to minimize

$$\sum_{i=1}^{N} \{\tilde{d_i}(\alpha, \mathcal{D}) - (\alpha^* - 1)\tilde{t_i^s} + \hat{\beta}\}$$

such that

$$\hat{\beta} \geq (\alpha^* - 1)\tilde{t_i^s} - \tilde{d_i}(\alpha, \mathcal{D}), \quad \text{for} \quad 1 \leq i \leq N,$$

which solves the case when the skew is $\alpha \neq 1$. Let $\hat{\alpha}_{\mathcal{A}}(\alpha, \mathcal{D})$ and $\hat{\beta}_{\mathcal{A}}(\alpha, \mathcal{D})$ be the values that solve the above problem. Then we can conclude:

$$\hat{\alpha}_{\mathcal{A}}(\alpha, \mathcal{D}) - \alpha = \hat{\alpha}_{\mathcal{A}}(1, \mathcal{D}) - 1 \quad \text{and} \quad \hat{\beta}_{\mathcal{A}}(\alpha, \mathcal{D}) = \hat{\beta}_{\mathcal{A}}(1, \mathcal{D}).$$

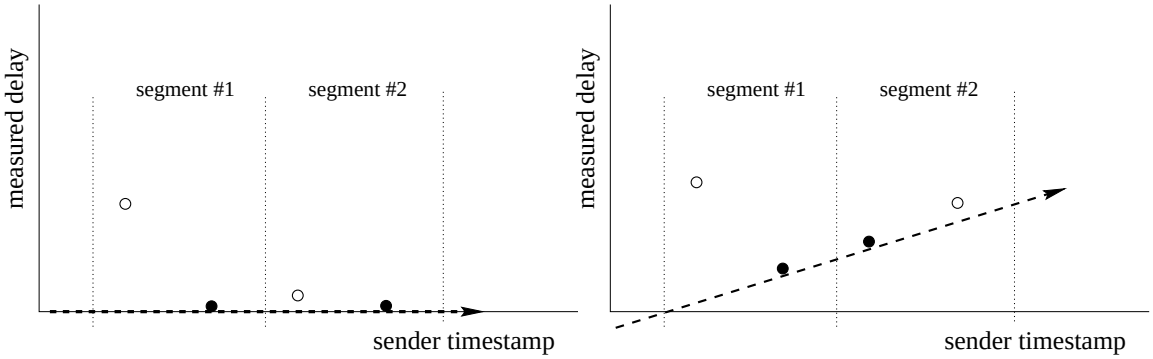
7.3.2 Other algorithms

It is clear that the linear regression algorithm satisfies Property 3. For the piecewise minimum algorithm, we use Figure 4 to demonstrate how a minimum-based algorithm does not exhibit Property 3. In the figure, the x-axis represents the sender timestamp, and the y-axis represents the measured delay. We assume that there are four delay measurements available, that and they are partitioned into two segments, each with two measurements. The minima of the segments are marked as black in the figure. When there is no clock skew, as shown in the left-hand side, the second and the fourth measurements are the minima of the segments, and the estimated skew has the slope of the horizontal solid line that lines with the x-axis. If the clock skew is bigger than 0, as shown in the right-hand side, the measured delay will increase proportional to their

α -1	$\mu = 10msec$				$\mu = 100msec$			
	Linear programming		Paxson		Linear programming		Paxson	
	mean	var.	mean	var.	mean	var.	mean	var.
0.01	0.01	0.9666e-5	0.01	0.0373e-3	0.01	0.8405e-4	0.01	0.0002
0.1	0.1	0.9666e-5	0.1	0.1073e-3	0.1	0.8405e-4	0.1001	0.0004
4	4	0.9666e-5	4.0001	0.3945e-3	4	0.8405e-4	4.0001	0.002

Table 1: Simulation results to test Property 3

corresponding sender timestamp, as the arrow curves indicate in the figure. The increase in measured delay due to a clock skew is a function of the sender timestamp, but not of the end-to-end delay as stated in (6). As the skew gets larger, the increase due to the clock skew becomes the dominant part of a measured delay, and the minimum of a segment is more likely to be found near the beginning of the segment. It is illustrated in the right-hand side, as the second and third measurements are picked as minima when the skew is above 0 and the estimated skew has the slope of the dotted arrow line. Depending on the magnitude of the skew, a minimum-based algorithm uses different minima, and clearly the differences between the estimate and the actual skew in these two cases are not the same.



when there is no skew when the skew is $\alpha \neq 1$

Figure 4: Example of minima that change depending on the magnitude of skew

Since Paxson's algorithm employs a robust line fitting technique on top of local minima, we choose to simulate Paxson's algorithm to examine its robustness. In the simulation, the number of packets is 600, and α changes from 0.01 to 0.1 and 4. The end-to-end delay consistent with C_r, $\bar{d}_i$, is assumed to have an exponential distribution with the means, $\mu = 10msec$ and $\mu = 100msec$. The purpose of the simulation is to show the variability of the difference between the estimate and the actul skew over a range of clock skews. Thus we use the same set of $\bar{d}_i$ for all three values of α, and calculate the sample mean and the variance of the estimates. As shown in Section 7.3.1, the difference between the actual skew and the estimate does not change in the linear programming algorithm case, and thus the sample variance of the estimates from the algorithm remains the same for the three values of α in the simulation. The sample variance of Paxson's algorithm, however, increases as the skew increases. This illustrates that the difference between the actual skew and the estimate of Paxson's algorithm grows as the skew grows, and thus the algorithm does not satisfy Property 3.

Trace	Date	Type	Source	Destination	Interval	Time	Duration
1	14Nov97	uni	im.cs.umass.edu	anhur.sics.se	80ms	12:50	4hr 10min
2	20Nov97	multi	eraser.cs.umass.edu	maria.wustl.edu	160ms	15:54	10hr
3	21Nov97	uni	im.cs.umass.edu	anhur.sics.se	160ms	0:03	5hr 54min

Table 2: Trace Descriptions

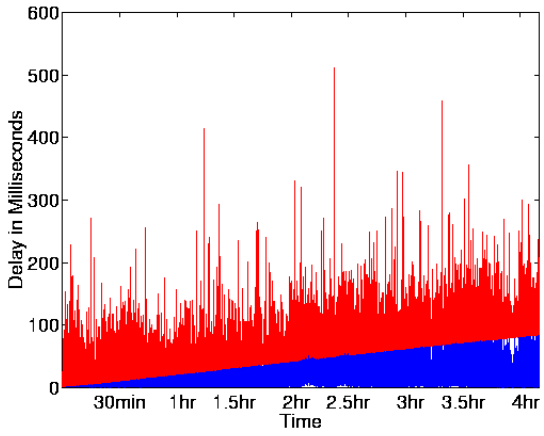


Figure 5: Linear Programming Algorithm

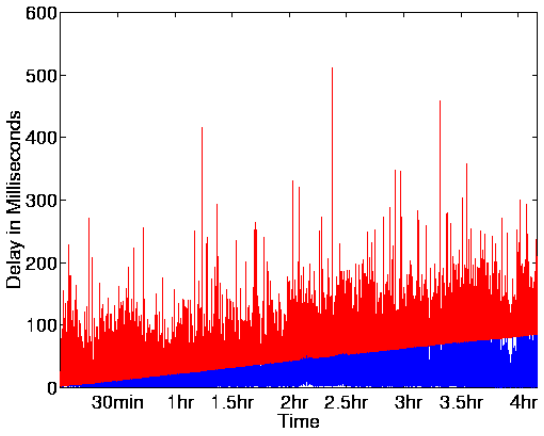


Figure 6: Paxson's Algorithm

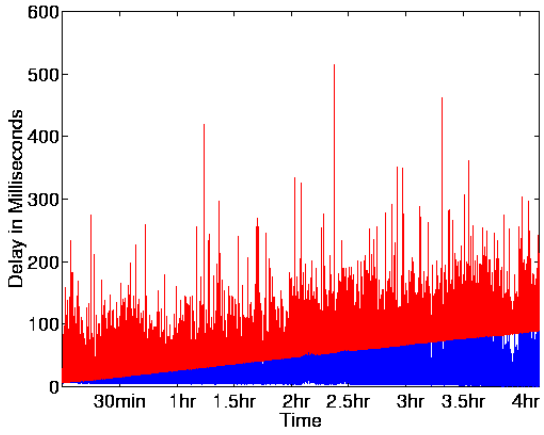


Figure 7: Linear Regression Algorithm

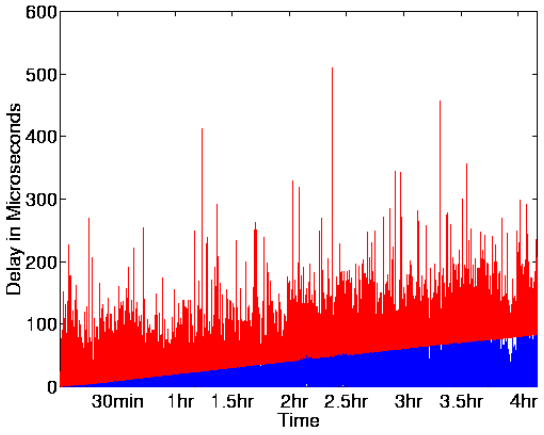


Figure 8: Piecewise Minimum Algorithm

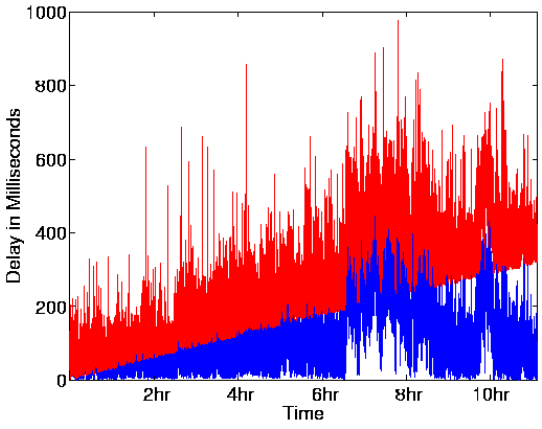


Figure 9: Linear Programming Algorithm

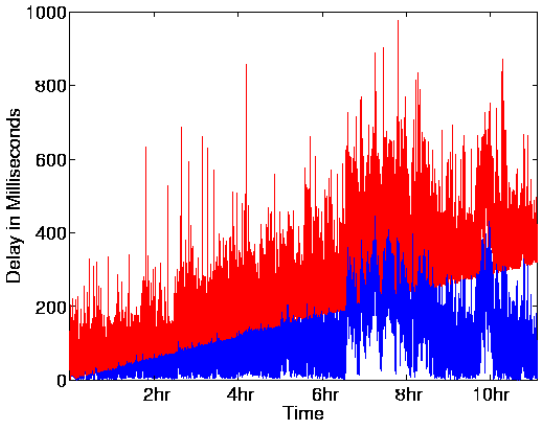


Figure 10: Paxson's Algorithm

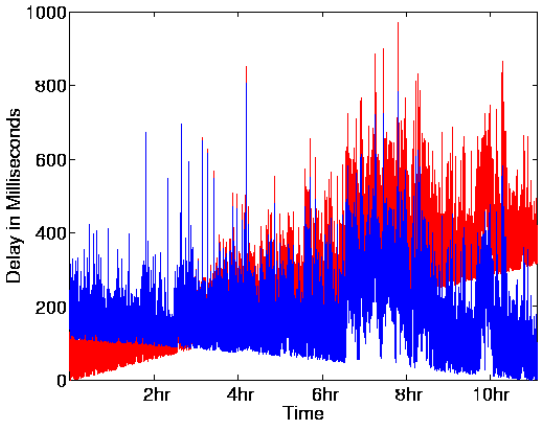


Figure 11: Linear Regression Algorithm

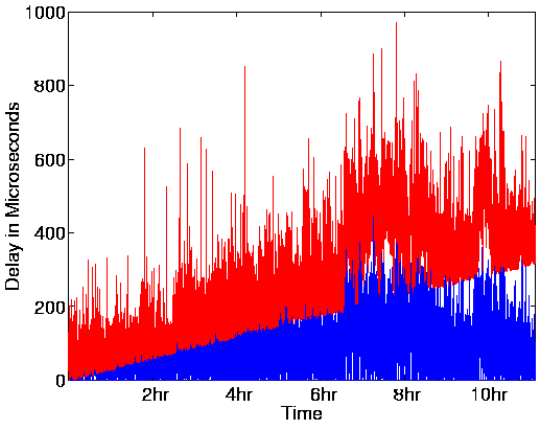


Figure 12: Piecewise Minimum Algorithm

7.4 Measurement

In this section we use Internet delay measurements to see how each algorithm performs when applied to actual measurements. We collected several traces of delay and loss measurements over the Internet and MBone [Eri94] between November 14, 1997, and December 21, 1997. Table 2 provides a brief description of the traces. The clocks of the end-hosts were not synchronized in all traces. We used constant-length UDP packets whose payloads consisted of a sequence number and a timestamp, and they were sent out at periodic intervals.

Figures 5 to 8 are from Trace 1 in Table 2. On the x-axis the sender timestamp, $\tilde{t}_i^s$ is plotted, and on the y-axis, the delay before and after the skew estimation and removal, $\tilde{d}_i$ and $\tilde{d}_i - (\hat{\alpha} - 1)\hat{t}_i^s$, are plotted.*

The gray foreground is the delay before the skew is estimated and removed, and the black background is the delay after the skew is removed. Even though the linear skew trend is conspicuous, the frequency ratio between the two clocks is small, and it is hard to verify visually which algorithm is better among the three algorithms with the exception of the linear regression algorithm, which is the only one whose adjusted delay is not as close as other algorithms to the x-axis between 0 and 2.5 hours.

Figures 9 to 12 are from Trace 2 in Table 2. The same is on the x and y axes, as in Figures 5 to 8. The linear skew trend is apparent as in Trace 1, but the delay behavior changes significantly in the later half of the trace, and more losses are detected. This is a multicast packet delay measurement, and the overall loss rate of the trace is very high: 42%. The losses are more pronounced as the jagged bottom of the gray plot in the second half of the trace. Considering the extraordinary high loss rate of the trace, we think that the clock skew was constant, but due to heavy congestion inside the network the queueing delay has increased significantly over an extended period of time, and is shown in measurements.

In Figure 11 The linear regression algorithm fails miserably in estimating a skew trend. The large delayss between 6 and 8 hours on the x-axis produce outliers which have a significant impact on the linear regression. After the skew is filtered out, the delay has a decreasing trend, which is the opposite of the original increasing trend. Since the linear programming and Paxson's algorithms come up with one estimate for the constant skew, the resulting delays from both algorithms are close to the x-axis, while keeping the increased delay trend intact. The piecewise minimum algorithm calculates too high a minimum over the increased delay period, and ends up interpreting the increased delay trend as a skew. The result is that the effect of network congestion on delay is removed along with the skew.

Figures 5 to 12 visually demonstrate the relative performance of each algorithm. The linear programming and Paxson's algorithms estimate the skew accurately in the presence of different levels of network congestion. In contrast, the linear regression and piecewise minimum algorithms perform poorly when the network is heavily congested, and the delay fluctuates significantly. In order to investigate the relative performance more precisely, we simulate the the linear programming and Paxson's algorithms with synthetic delay, and compare the sample mean and variance of the estimate in the next section.

7.5 Simulation

The purpose of the simulation is to examine the average performance of a skew estimation algorithm in terms of the sample mean and variance of the estimate. We have performed two sets of simulations for each of the linear programming and Paxson's algorithms. The first set of simulations (referred to as Set 1) uses a periodic inter-packet departure time of 20 milliseconds; the second set (Set 2) uses a periodic inter-packet departure time of 1 second. The other parameters of the simulation are the number of packets, the mean delay (μ), and the skew. The number of packets is either 600 or 3000. In Set 1, 600 packets correspond to 12 seconds and 3000 to 1 minute; in Set 2, to 10 minutes and 1 hour, respectively. We assume an exponential

*Here we actually plot $\tilde{d}_i - \min_i \tilde{d}_i$ and $\tilde{d}_i - (\hat{\alpha} - 1)\hat{t}_i^s - \min_i \tilde{d}_i$ to plot the delays before and after the skew estimation and removal in the same range of values on the y-axis.

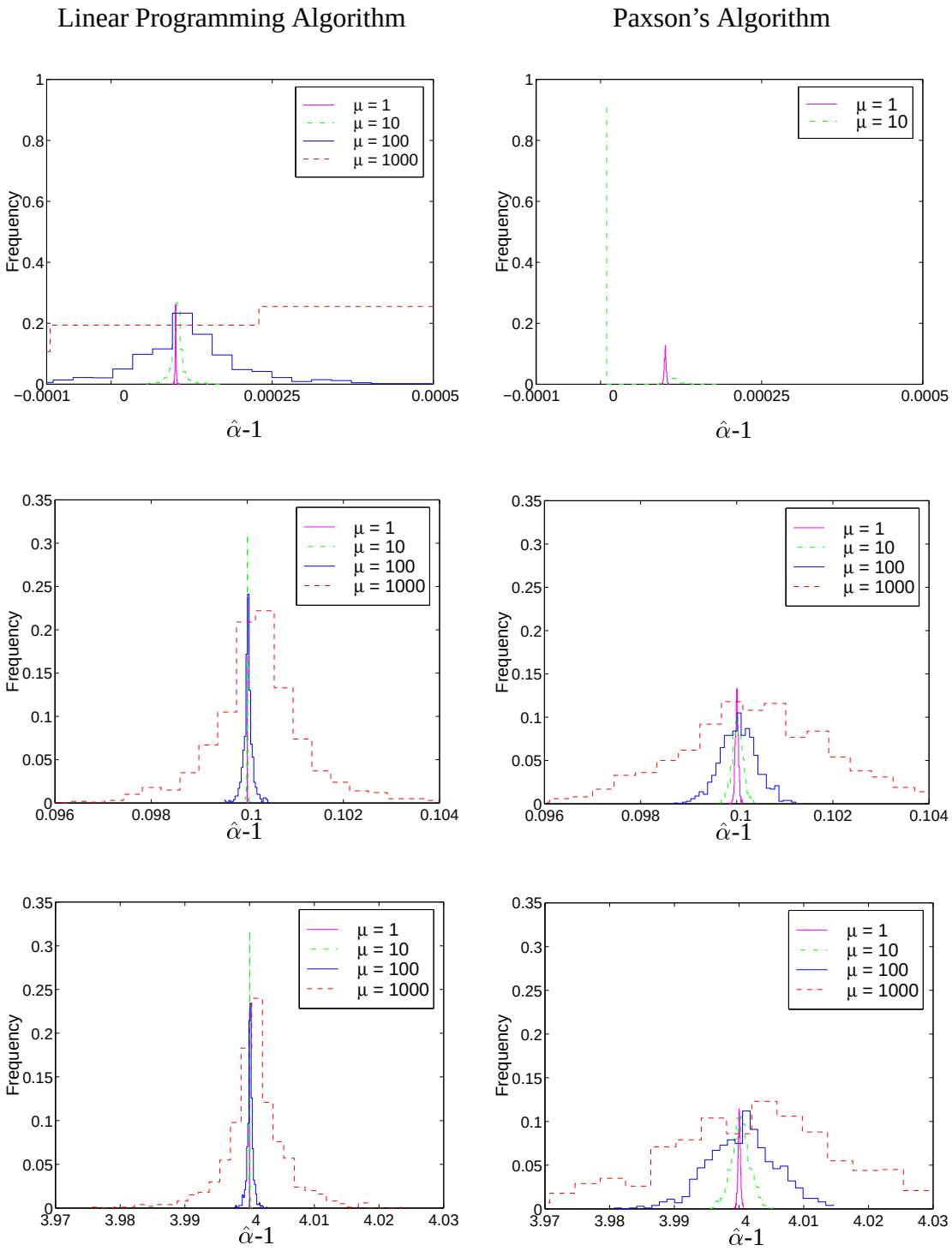


Figure 13: Set 1 - Histograms of $\hat{\alpha} - 1$ when the number of packets is 600

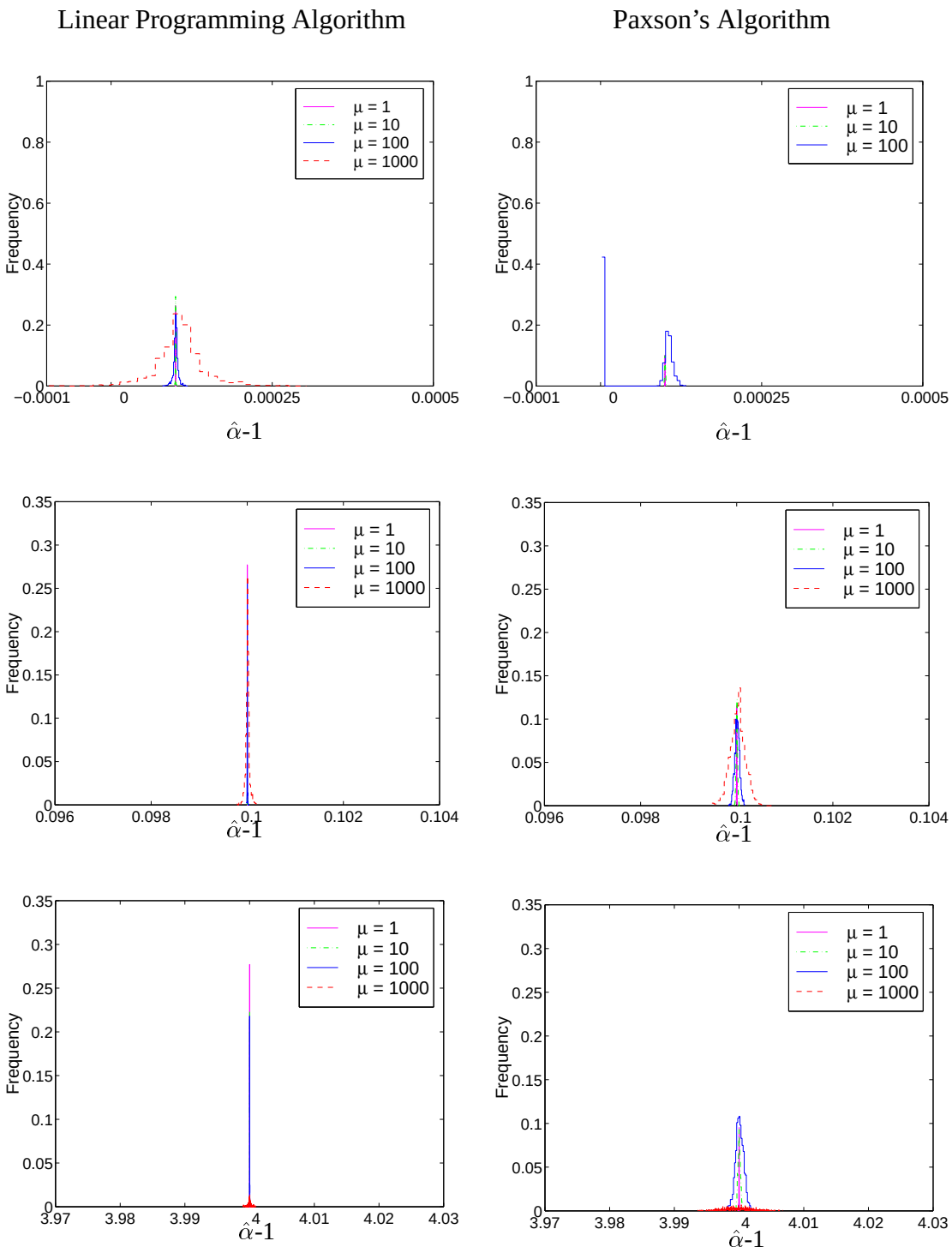


Figure 14: Set 1 - Histograms of $\hat{\alpha} - 1$ when the number of packets is 3000

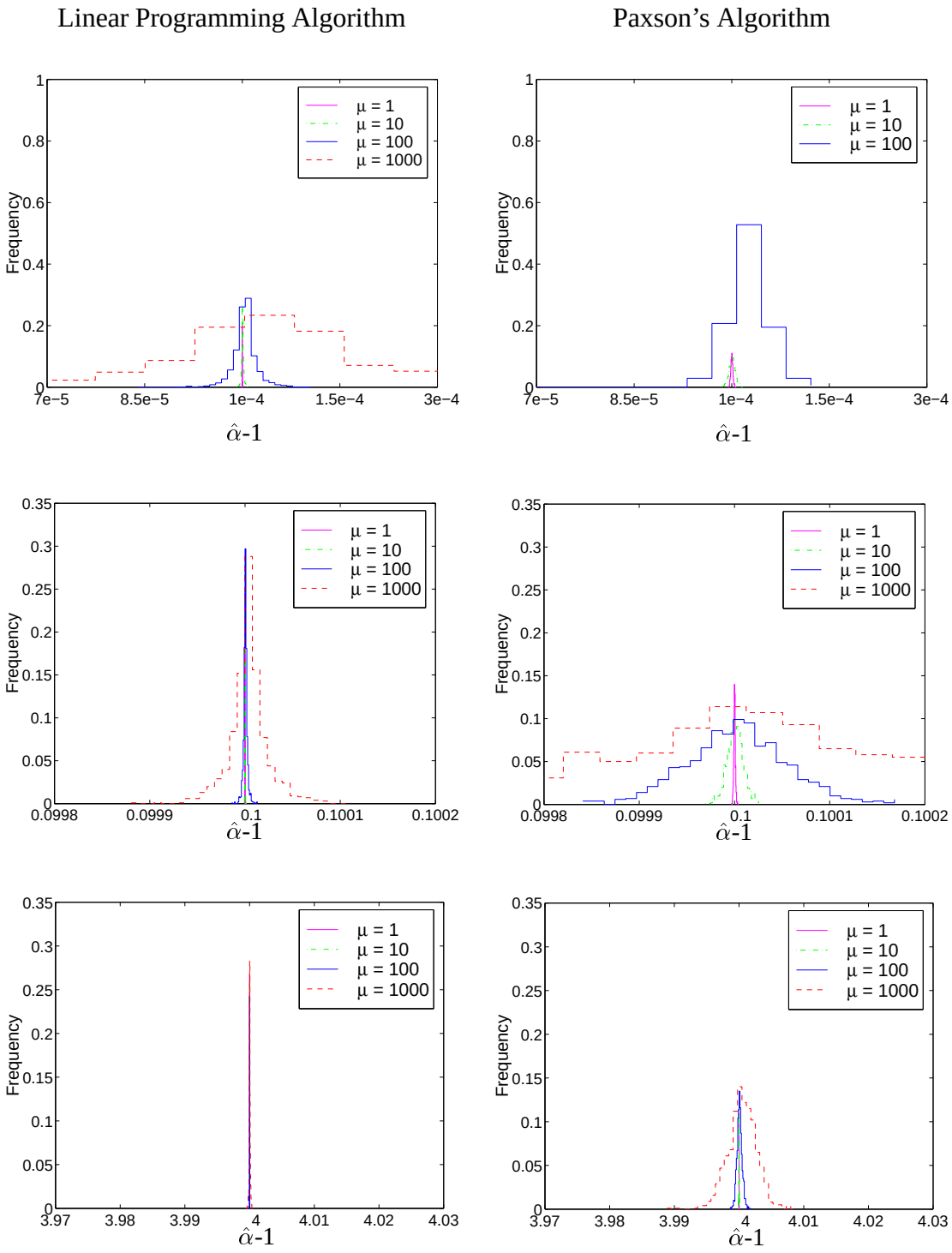


Figure 15: Set 2 - Histograms of $\hat{\alpha} - 1$ when the number of packets is 600

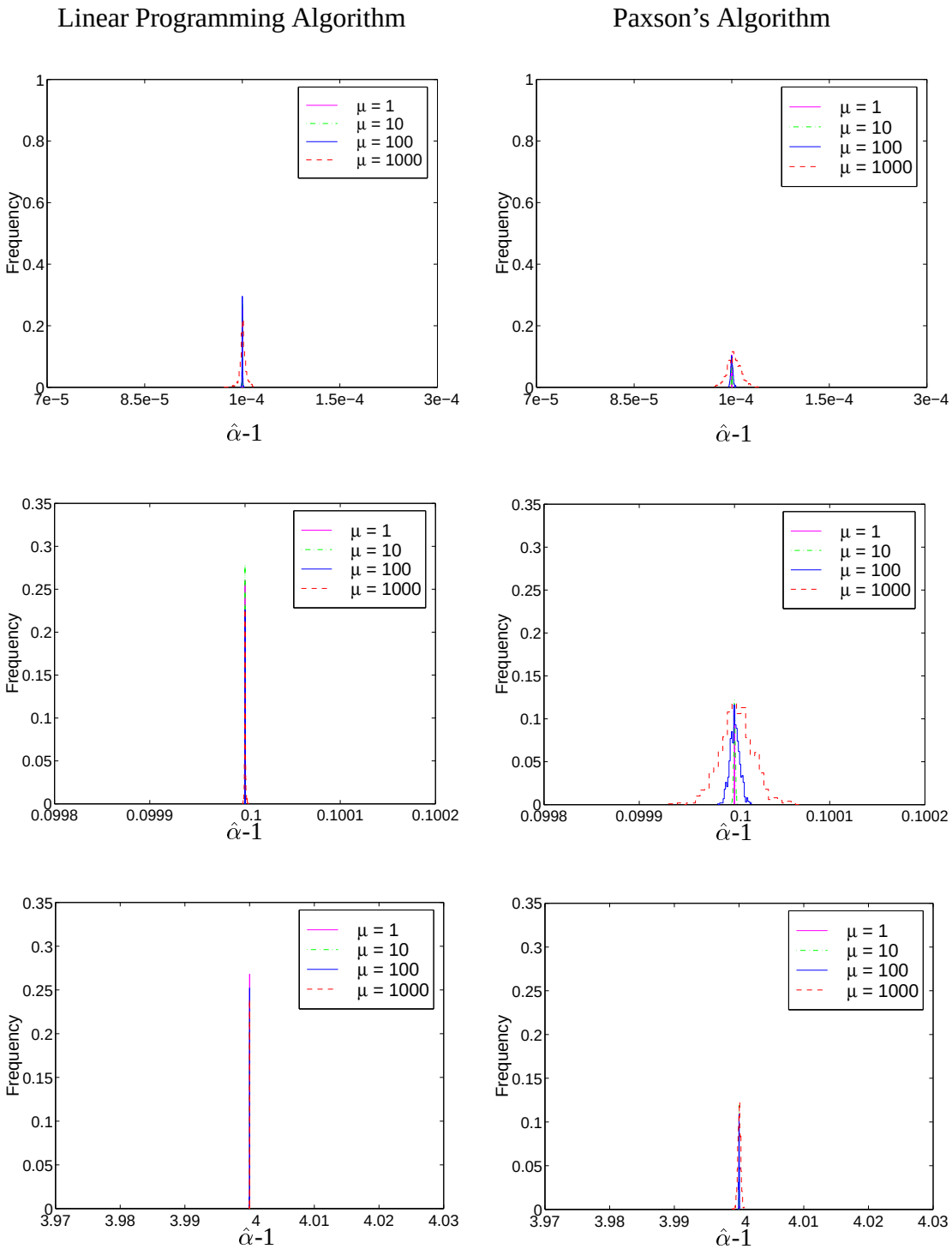


Figure 16: Set 2 - Histograms of $\hat{\alpha} - 1$ when the number of packets is 3000

distribution for end-to-end delays in our simulation and vary the mean (μ) of the exponential distribution from 1 millisecond to 10, 100, and 1000 milliseconds for both Sets 1 and 2. We also vary the magnitude of the clock ratio(α) from 1.0001 to 1.1 and 5. For a set of fixed values of the number of packets, mean delay, and skew, 1000 runs were executed, and the histograms of the skew estimates from the 1000 runs are plotted in Figures 13 to 16. The x-axis plots $\hat{\alpha} - 1$, and the y-axis, the frequency. In all the graphs, histograms have a fixed bin size of 30; the greater distance between the minimum and maximum of the estimates is, the wider the bin is.

Figures 13 and 14 plot the results of the linear programming and Paxson's algorithms from Set 1, when the number of packets is 600 and 3000, respectively. Figures 15 and 16 from Set 2. The first rows of Figures 13 to 16 is for $\alpha - 1 = 0.0001$, the second rows for $\alpha - 1 = 0.1$, and the last rows for $\alpha - 1 = 4$. The ranges on the x-axis and y-axis are fixed for a given α in a set to make it easy to compare how the histograms spread horizontally between the linear programming and Paxson's algorithms.

In both Sets 1 and 2, the histograms when the number of packets is 600 are more widespread than when the number of packets is 3000. This is because the estimate gets more accurate as more delay measurements are available.

Most histograms are symmetric around, and their estimates are very close to the true α values with a sample variance less than $\pm 4\%$ of α. However, as α gets closer to 1, the distribution of estimates starts to diverge from a symmetrical shape in Paxson's algorithm. In the upper-right graph of Figure 13 where the number of packets is 600 and $\alpha - 1 = 0.0001$, the histograms for $\mu = 100$ and $\mu = 1000$ are not plotted because their frequencies are 1 at $\hat{\alpha} - 1 = 0$. The same is true for $\mu = 1000$ when the number of packets is 3000 in Set 1 (Figure 14), and for $\mu = 1000$ when the number of packets is 600 in Set 2 (Figure 15). This is because of Step 3 in Paxson's algorithm. If the number of cumulative minima is not significant enough, Paxson's algorithm assumes no skew, and outputs $\hat{\alpha} = 1$. The simulation results show that it is hard to pass Step 3 and the algorithm is biased against a small skew, when the average delay is relatively large, and the measurements span over a short period of time.

The histograms of the linear programming algorithm are consistently less spread out than those of Paxson's algorithm for the given range of parameter values. We have shown through simulation that the estimate of our linear programming algorithm is unbiased against a small skew, and is likely to have less variance than that of Paxson's.

8 Further Discussion

8.1 Non-zero Clock Drift

In the delay traces, we have observed mostly linear skew trends, and in some cases non-linear trends. Trace 3 in Table 2 is one such case. Figure 17 displays $\tilde{t}_i^s$ on the x-axis and $\tilde{d}_i$ on the y-axis from Trace 3. The slope of the delays is changing over time in the first hour of the trace. The drift, as defined in Section 3.1, is non-zero. It attests that a slowly varying drift is present in the first hour of the trace, and then a linear trend sets in and lasts until the end of the trace. Figure 18 is a 100 times magnified view of the first 3.5 minutes in Figure 17. The slowly varying drift in a large time scale of hours is not noticeable in the magnified graph. Instead in this small time scale of less than 10 minutes, we only see a constant skew trend.

This implies that we can apply the constant skew estimation algorithms in a piecewise manner on a smaller time scale. How to detect a non-zero drift, and to determine the proper size to partition a long trace are important issues, and need further study. Another issue concerning the piecewise application of the skew estimation algorithms is to conserve the continuity of end points between partitions. We think that by carefully applying the current algorithms we can avoid the discontinuity in delay between partitions after the skew and drift are removed.

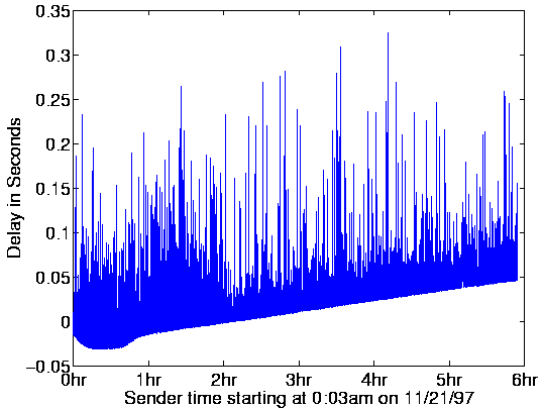


Figure 17: Non-zero Drift

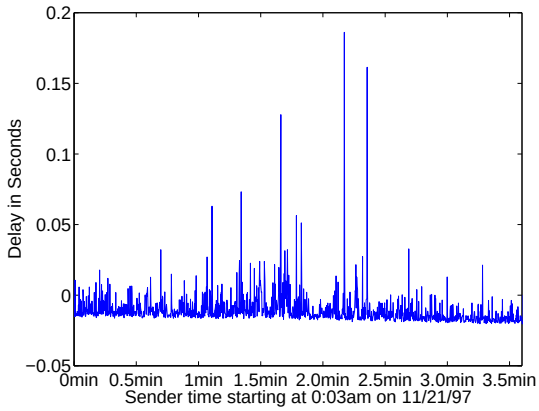


Figure 18: Non-zero Drift: Magnified View

8.2 Detection of clock adjustments

When a clock adjusts its time, the adjustment is likely to show in delay measurements. Paxson uses two-way measurements to detect such a clock adjustment[Pax97, Pax98]. It is harder to distinguish a genuine clock time adjustment from the delay fluctuation due to network congestion in one-way delay measurements, and needs further study.

9 Conclusion

In this paper, we have presented a framework for understanding the systematic errors introduced in one-way network delay measurements by unsynchronized clocks, and discuss several properties desirable of a skew estimation algorithm. We have developed an linear-programming-based algorithm, and compared it with three other existing algorithms. The linear regression and piecewise minimum algorithms demonstrated a poor performance over traces of Internet delay measurements. We generated synthetic delay measurements, and analyzed the sample mean and variance of the estimates of our and Paxson's algorithms. The results show that the estimate of the linear programming algorithm is likely to be unbiased and have less variance. In conclusion, the linear programming algorithm addresses all the desirable properties, and is simple, fast, and robust.

Acknowledgements

We would like to thank Maya Yajnik for helping us with the data collection; Steve Pink at Swedish Institute of Computer Science, and Chuck Cranor at Washington University in St. Louis for letting us collect data at their sites; Jean-Chrysostome Bolot for the suggestion of the piecewise minimum algorithm; Christopher Raphael for pointing us to the linear programming approach; and Robbie Moll for giving us his intuition behind the complexity of our algorithm.

References

[DH95] S. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) specification. RFC 1883, Internet Engineering Task Force, December 1995.

[DS95] Sudhir Dixit and Paul Skelly. MPEG-2 over ATM or video dialtone networks: issues and strategies. *IEEE Network*, 9(5):30–40, September-October 1995.

[Dye83] M. E. Dyer. Linear algorithm for two- and three-variable linear programs. *SIAM Journal on Computing*, 13:31–45, 1983.

[Eri94] Hans Eriksson. MBONE:the multicast backbone. *Communications of ACM*, 37(8), August 1994.

[HMT83] D. Hoaglin, F. Mosteller, and J. Tukey, editors. *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, 1983.

[Meg83] N. Megiddo. Linear time algorithm for linear programming in r^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, November 1983.

[Mil92a] D. Mills. Modelling and analysis of computer network clocks. Technical Report 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.

[Mil92b] D. Mills. Network time protocol(version 3): Specification, implementation and anlysis. Technical report, Network Information Center, SRI International, Menlo Park, CA, March 1992.

[MKT98] Sue B. Moon, Jim Kurose, and Don Towsley. Packet audio playout delay adjustment: Performance bounds and algorithms. *ACM/Springer Multimedia Systems*, 5:17–28, January 1998.

[PAMM98] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP performance metrics. RFC 2330, Internet Engineering Task Force, May 1998.

[Pax97] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997.

[Pax98] Vern Paxson. On calibrating measurements of packet transit times. In *Proceedings of SIGMETRICS '98*, Madison, Wisconsin, June 1998.

[RKTS94] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proceeding of INFOCOM '94*, 1994.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, Internet Engineering Task Force, jan 1996.

A Linear programming algorithm

```
1.  PROCEDURE: IN(d̃_i, t̃^s_i, N ) OUT( α̂, β̂)
2.  //  line(slope, y-intercept) returns a line.
3.  //  x(line1, line2) and y(line1, line2) return x- and y-coordinates of intersection.
4.     n_1 = 1,  n_2 = 2,  k = 2
5.     FOR  i = 3 to N
6.        FOR  j = k downto 2
7.           IF  x(line(t̃^s_i, −d̃_i), line(t̃^s_{n_j}, −d̃_{n_j})) > x(line(t̃^s_{n_j}, −d̃_{n_j}), line(t̃^s_{n_{j−1}}, −d̃_{n_{j−1}}))
8.              BREAK;
9.           ENDIF
10.       ENDFOR
11.       // Note that j is 1 when the FOR loop in line 6 runs to the end.
12.       k = j + 1, n_k = i
13.    ENDFOR
14.    opt_s = Σ_i t̃^s_i/N
15.    FOR  i = 1 to k − 1
16.       IF  (t̃^s_{n_i} < opt_s) AND (opt_s < t̃^s_{n_{i+1}})
17.          α̂ − 1 = x(line(t̃^s_{n_i}, −d̃_{n_i}), line(t̃^s_{n_{i+1}}, −d̃_{n_{i+1}})),
18.          β̂ = y(line(t̃^s_{n_i}, −d̃_{n_i}), line(t̃^s_{n_{i+1}}, −d̃_{n_{i+1}}))
19.          BREAK;
20.       ENDIF
21.    ENDFOR
22. ENDPROCEDURE
```

B Paxson's algorithm

```
1.  PROCEDURE: IN(d̃_i, t̃^s_i, N) OUT(α̂)
2.  //  slope(p1, p2) returns the slope of the line
3.  //  R(n, k) is from [Pax98]
4.  //  threshold is either 10^-3 or 10^-6.
5.     M = ⌊√N⌋
6.     FOR  i = 1  to  M
7.        m_i = min_{(i-1)M<j<=iM} d̃_j,
8.        n_i = arg_j min_{(i-1)M<j<=iM} d̃_j
9.     ENDFOR
10.    IF  N − M^2 > M/2
11.       m_{i+1} = min_{(M^2+1)<=j<=N} d̃_j,
12.       n_{i+1} = arg_j min_{(M^2+1)<=j<=N} d̃_j,
13.       M = M + 1
14.    ENDIF
15.    G_s = median_{1<=i,j<=k} slope((t̃^s_{n_i}, m_{n_i}), (t̃^s_{n_j}, m_{n_j}))
16.    //  Here we assume that G_s is negative and thus the trend is decreasing.
17.    k = 1
18.    cm_1 = m_1
19.    FOR  i = 2  to  M
20.       IF  m_i < min_{1<=j<=(i-1)} m_j
21.          k = k + 1, cm_k = m_i, n_k = i
22.       ENDIF
23.    ENDFOR
24.    IF  R(M, k) > threshold
25.       RETURN(1)
26.    ENDIF
27.    α̂ = median_{1<=i,j<=k} slope((t̃^s_{n_i}, cm_{n_i}), (t̃^s_{n_j}, cm_{n_j}))
28. ENDPROCEDURE
```