Clock Synchronization Algorithms for Network Measurements

Li Zhang, Zhen Liu and Cathy Honghui Xia

Abstract—**Packet delay traces are important measurements for analyzing end-to-end performance and for designing traffic control algorithms in computer networks. Due to the fact that the clocks at end systems are usually not synchronized and running at different speeds, these measurements can be quite inaccurate. We propose several algorithms to estimate and remove the relative clock skews from delay measurements based on the computation of convex hulls. Compared with existing techniques such as linear regression and linear programming, the convex-hull approach provides better insight and allows us to handle more error metrics. We obtain algorithms which are linear in the number of measurement points for the case with no clock resets. For the more challenging case with clock resets, i.e., the clocks are reset to some reference times during the measurement period, we develop linear algorithms to identify the clock resets, and derive the best clock skew lines. We extend this analysis to environments in which at least one of the clocks is controlled by NTP. These algorithms can greatly improve the accuracy of the measurements, and can be used both online and offline. They can also be extended for active clock synchronization, to replace or further improve NTP. Numerical experiments are presented to demonstrate the robustness of the algorithms.**

I. Introduction

Packet delay traces are important measurements for analyzing end-to-end performance and for designing traffic control algorithms in computer networks. These measurement data can help in decision making in traffic routing, capacity planning, application tuning, alarm detection and network fault detection, etc. These delay traces can be obtained either by monitoring tools or by active probing. In either case, time stamps of packets are collected at the source and the destination. The difference between the two timestamps of the same packet is the *measured delay* for the end-to-end network delay experienced by that packet. If the two host clocks are perfectly synchronized, then the *measured delay* is the true delay. However, in real measurements, the two host clocks are usually not synchronized. In particular, the two clocks may run at different speeds. This difference in speed is called the *clock skew*. It is therefore possible for the receiver to receive a packet from the "future", resulting in a negative delay according to the measurement. The *measured delay* in this case can be very different from the *true delay*.

In this paper, we address the problem of estimating and removing the relative clock skews from delay measurements. The problem becomes more challenging and complicated in the case that the clocks may be reset through system calls such as `rdate`. Such resets are typically performed at a very coarse level through the `cron` daemon, e.g. a couple of times a day. Without prior knowledge of the reset times, we need to "detect" them from the data, and obtain the "correct" delay measurements. Another type of resets is velocity adjustments through the use of Network Time Protocol (NTP) [3]. Such velocity adjustments are usually performed at a finer time granularity.

Although we have no prior knowledge about the offset, about skew between the two clocks, and about the reset times of either clock, there is still a lot of information contained in the data that allows us to make reasonable estimates of the clock skew. Assume we have a collection of measurement data, $\Omega := \{v_i = (t_i, d_i) : i = 1, \ldots, N\}$, where t_i is the time the packet was sent according to sender's clock, and d_i is the *measured delay*. We plot these data in the 2-D plane using t_i as the x coordinate and d_i as the y coordinate. From the plot we observe that all the points are supported by a straight line and that this straight line has a non-zero slope (Figure 1). The interpretation of this phenomenon, if the two clocks are perfectly synchronized, would be that there is a steady trend for the delay to grow (or decay) as time progresses. This is very unlikely to happen. It is therefore reasonable to attribute such a trend to clock skew. After removing such clock skew, the resulting delay can then be used as a true measurement of the network condition. It is also possible for temporary internet congestion to cause the delay measurements to increase for a period of time. This is illustrated by the lower plot in Figure 1 with simulated data.

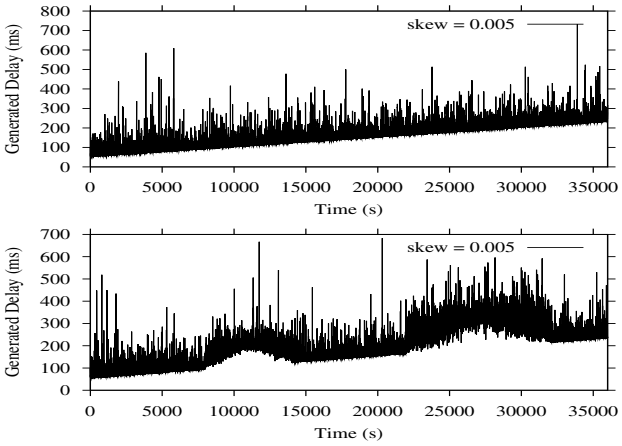

Fig. 1. Delay Measurements (generated)

In the literature, several solutions have been proposed to address this problem of clock skew correction. In [5], Paxson provides an algorithm using the median line fitting technique. This algorithm, however, provides a poor estimate of the slope of the trend when the data is highly variable. The linear regression algorithm is discussed in both [4] and [5]. It does not work well due to the nature of the delay measurements. Specifically, the linear regression algorithm works well when the distributions of the data are normal. The network delay measurements, however, rarely satisfy this condition. Furthermore, temporary network congestion would cause significant amount of deviation for the skew slope estimations. The piecewise minimum is discussed in [4]. It provides a series of skew segments. These

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598; {zhangli,zhenl,cathyx}@us.ibm.com

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

segments are very unlikely to be a straight line, or to have the same slope. It therefore does not provide the correct skew estimate. In cite[moon], Moon, Skelly and Towsley formulate the problem as a linear program and solve it using standard algorithms in [1]. This approach is closest to our current solution.

These algorithms, however, do not apply to the cases with clock resets or velocity adjustments. Our approach is based on the computation of convex hulls. Compared with existing techniques such as linear regression and linear programming, the convex-hull approach provides better insight and allows us to handle more error metrics. We obtain algorithms which are linear in the number of measurement points when there are no clock resets. For the more challenging cases with clock resets, i.e., the cases where the clocks are reset to some reference times when the delays traces are recorded, we develop linear algorithms to identify the clock resets, and then derive the best clock skew lines. We extend this analysis to environments in which at least one of the clocks is controlled by NTP. These algorithms can greatly improve the accuracy of the measurements, and they can be used both online and offline. The online feature of our approach allows us to use the algorithms for online clock synchronization. Furthermore, our approach has the advantage that it provides analytical solutions for different objective functions. Numerical experiments are also presented to demonstrate the robustness of the algorithms.

The presentation of the paper is organized as follows. We first define the notation and the problem under consideration. We then describe in Section III our convex-hull approach for the case without clock resets. In Section IV, we extend the technique and propose an algorithm for the case with clock resets. In Section V, we describe an algorithm for the case of NTP. The issue of online use of the convex-hull approach is discussed in Section VI. In Section VII, we present experimental results obtained under various configurations. Finally, concluding remarks are provided in Section VIII.

II. General Problem Formulation

Based on the observations made in the last section, namely, the supporting straight line for clock skew and the abrupt shift of the delay level for clock reset, we can obtain a mathematical formulation for the clock skew problem.

Assume we have the collection of measurements, $\Omega = \{v_i = (t_i, d_i) : i = 1, \ldots, N\}$. If we do not consider clock resets, the problem is to find a linear function which is below all the points in Ω, and is closest to Ω in some sense. There are many possible metrics for determining how close a line is to a set of points. We will discuss three metrics that mimic our observation in the last section, will illustrate their different properties and show how to solve the problem for these metrics.

With clock resets, the problem is then to find a piecewise linear function, with each piece having the same slope, such that all the points in Ω are above the function, and such that this piecewise linear function is closest to Ω under some objective function. Note that each piece should have the same slope since the difference in speed (or the rate of the skewness) of two given clocks is fixed. In real systems, one would not expect frequent clock resets.

We assume that all the t_i's are initially sorted in increasing order, which will usually be the case. This assumption allows us to develop linear time algorithms to solve the problem.

III. Convex Hull Approach for Clock Skew Estimation

We first focus on the simpler case with no clock reset. Suppose the line for clock skew is $L := \{(x, y) | y = \alpha x + \beta\}$. The restriction for all the points in Ω to be above this line can be expressed as

$$\alpha t_i + \beta \le d_i. \tag{1}$$

Among all lines that satisfy this condition we would like to choose the one that is the closest to Ω.

A. Objective Functions

We consider three metrics that can be used as the objective function for the optimization problem described above. These three examples are simple enough, yet capture the key ideas behind our intuition. We use them as example metrics to illustrate how our approach works. There are certainly other objective functions that one could use to solve the problem.

(1) Minimize the sum of the vertical distances between the points and the line.

The objective function is then

$$obj_1 := \sum_{i=1}^{N} (d_i - \alpha t_i - \beta) = \sum_{i=1}^{N} d_i - \sum_{i=1}^{N} t_i \, \alpha - N\beta \tag{2}$$

This is the the objective function used by Moon, Skelly and Towsley [4] in their linear programming formulation. They implemented an $O(N)$ algorithm from [1], [2] that takes advantage of the fact that all the t_i's in Ω are already sorted.

(2) Minimize the area between the curve and the line.

To obtain this objective function we can sum over the area between the line $y = \alpha x + \beta$ and the line segment between every two consecutive points in Ω. This gives

$$\begin{aligned} obj_2 &:= \sum_{i=1}^{N-1} (d_i - \alpha t_i - \beta + d_{i+1} - \alpha t_{i+1} - \beta) \frac{(t_{i+1} - t_i)}{2} \\ &= \sum_{i=1}^{N-1} \frac{(d_i + d_{i+1})(t_{i+1} - t_i)}{2} - \frac{t_N^2 - t_1^2}{2} \alpha - (t_N - t_1)\beta \end{aligned} \tag{3}$$

This is the objective function that we are going to focus on. In the special case that the sender is sending out the packet regularly, i.e., $t_{i+1} - t_i = c$, we have

$$\frac{N-1}{N} obj_1 - \frac{1}{c} obj_2 = \frac{d_1 + d_N}{2} - \frac{d_1 + \cdots + d_N}{N}.$$

Since $N, d_1, \ldots, d_N$, and c are fixed constants with given data, the two objective functions obj_1 and obj_2 are equivalent.

(3) Maximize the number of points on the line.

This objective function is different from the previous two in the sense that it is not linear in the variables α and β. We can write it using the indicator function

$$obj_3 := \sum_{i=1}^{N} 1_{\{d_i = \alpha t_i + \beta\}}. \tag{4}$$

In spite of being nonlinear, we can still solve the optimization problem of maximizing this objective function in time $O(N)$ by using our approach in the next section.

These three objective functions characterize different aspects of our observation of making the skew line as close to the points as possible. Each of them works well under certain circumstances, and performs poorly for some other cases. Intuitively, they give different weights for individual points in evaluating the distance between a set of points and a line. We will develop linear time algorithms for these three objective functions in the next section by computing the convex hull of Ω.

B. Convex Hull Approach

To solve the optimization problems we first take a closer look at the constraints in (1), which says that all the points in Ω are above the straight line $L = \{(x, y) | y = \alpha x + \beta\}$. From the theory of convex polytopes [6] we know that this is equivalent to saying the convex hull of Ω,

$$\text{co}\,(\Omega) := \{\, x \mid x = \sum_i \lambda_i v_i,\ \lambda_i \geq 0,\ \sum_i \lambda_i = 1,\ v_i \in \Omega \,\},$$

is above L. The convex hull of N points is a polytope enclosed by piecewise linear functions. In this case, it is enough to make sure the lower boundary of co (Ω) is above L. Taking advantage of the fact that the t_i's of all the points in Ω are sorted in increasing order we can find an algorithm that needs at most $2N$ operations to find the lower boundary of co (Ω).

The significance of the convex hull is that the "closest" line L to Ω will touch Ω at some point. If L does not touch Ω, we can always shift it up so that it is "closer" to Ω. Therefore, no matter what objective function one uses, the optimal straight line L will be below the convex hull co (Ω) and touch it at some point. Furthermore, it is easy to show that at least one of the touching points is in Ω. This is because all the vertices of co (Ω) are points in Ω, and the "closest" line to Ω touches co (Ω) at one or more of its vertices.

This special property of the convex hull is the key to the algorithms that we develop in the next section. And it plays a vital role in developing algorithms for other possible objective functions as well.

C. Algorithms

We will first present the key algorithm for finding co (Ω), and then show how to make use of this algorithm to find the optimal straight line with respect to the three objective functions.

C.1 Convex Hull

Given $\Omega = \{v_i = (t_i, d_i) : i = 1, \ldots, N\}$, with $t_1 \leq t_2 \leq \ldots \leq t_N$. We first find the lower boundary of co (Ω). It is well known that this lower boundary is composed of line segments whose end points are in Ω. We use a stack to keep track of these points.

The algorithm examines the points in Ω from left to right. For each point, it determines whether to push it into the stack right away or to pop some points out of the stack and push this point in. At the end of the algorithm, all the points in the stack are the vertices of the lower boundary of co (Ω). For our convenience, we will use $line(v, w)$ to denote the straight line connecting the two points v and w.

Algorithm Convex_Hull_L:

(1) Initialize: push v_1; push v_2;
(2) For $i = 3$ to N
  If (v_i above $line(top, next\_to\_top)$) push v_i;
  Else
    While (v_i below $line(top, next\_to\_top)$
    and $stack\_size > 1$)
      pop;
    push v_i;
(3) End

It is easy to see that when the algorithm stops, all the points in $stack$ are in Ω and the line segments of the consecutive points in $stack$ are in co (Ω). Furthermore, all the points in Ω are above these line segments. This is because each point v_i is pushed into the stack when it is first seen. It is popped out of the stack only when it is above the line segment between two other points in Ω. Therefore, the line segment of all the consecutive points in the stack is the lower boundary of co (Ω).

In the algorithm, for every comparison either a new point gets pushed into the stack, or a point in the stack gets popped out. Each point in Ω is pushed into the stack exactly once, and popped out at most once. Therefore, there are at most $2N$ push and pop operations before the algorithm stops.

We further remark that straightforward modification of algorithm Convex_Hull_L by reversing the role of above and below provides an algorithm named Convex_Hull_U, which gives the upper boundary of the convex hull. Combining algorithms Convex_Hull_L and Convex_Hull_U we can find the convex hull of Ω in linear time. We only need algorithm Convex_Hull_L in this paper.

C.2 obj_1 and obj_2

We first obtain the lower boundary of co (Ω) using algorithm Convex_Hull_L. We show next that the section of the lower boundary that covers the point $\sum_i t_i/N$ provides the optimal solution to obj_1, and that the section of the lower boundary that covers the middle point, $(t_1 + t_N)/2$ provides the optimal solution to obj_2. To establish these results, let us first present some structural properties of the problem.

For any fixed α, and given c_1, c_2, c_3, we define

$$\begin{aligned} f(\alpha) \quad = \quad & \min_{\beta} \quad c_1 - c_2\alpha - c_3\beta \\ & \text{s.t.} \quad t_i\alpha + \beta \leq d_i, \quad i = 1, \ldots, N. \end{aligned}$$

We can then take the minimum over α and obtain:

Proposition 1: The minimization over α, β can be sequentialized, i.e.,

$$\begin{aligned} \min_{\alpha} \ f(\alpha) \quad = \quad & \min_{\alpha,\beta} \quad c_1 - c_2\alpha - c_3\beta \\ & \text{s.t.} \quad t_i\alpha + \beta \leq d_i, \quad i = 1, \ldots, N. \end{aligned}$$

We now assume $c_3 > 0$. It is easy to check that obj_1 and obj_2 both satisfy this condition. By taking a closer look at the function $f(\alpha)$, we can prove the following key structural property.

Theorem 2: $f(\alpha)$ is convex in α.

Proof: From the constraints, we have $\beta \le d_i - t_i\alpha, \quad i = 1, \ldots, N$. Since $c_3 > 0$,

$$\begin{aligned} f(\alpha) &= c_1 - c_2\alpha - c_3 \min_i\{d_i - t_i\alpha\} \\ &= c_1 - c_2\alpha + c_3 \max_i\{t_i\alpha - d_i\} \end{aligned} \tag{5}$$

The result follows since $\max()$ is a convex function. □

Geometrically, for fixed α, we are simply shifting the line with slope α to touch the lower boundary of co (Ω) at an extreme point. The value of the objective function with respect to this line gives $f(\alpha)$.

Since $f(\cdot)$ is convex, a local minimal solution must be globally minimal. It suffices to find a local minimum of $f(\cdot)$. For $\alpha_1 < \alpha_2$, suppose $f(\alpha_1)$ and $f(\alpha_2)$ achieve the maximum in (5) at the same point, say (t_{i_0}, d_{i_0}). Note that this point must be an extreme point of co (Ω), as illustrated in Figure 2.

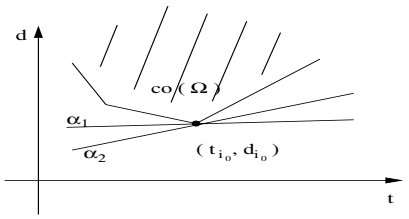

Fig. 2. Finding the vertex for the optimal skew line

We then have

$$f(\alpha_k) = c_1 - c_2\alpha_k + c_3(t_{i_0}\alpha_k - d_{i_0}), \qquad k = 1, 2.$$

It follows that $f(\alpha_1) - f(\alpha_2) = (c_3 t_{i_0} - c_2)(\alpha_1 - \alpha_2)$. Therefore,

$$f(\alpha_1) < f(\alpha_2) \iff t_{i_0} > c_2/c_3.$$

This shows that $f(\alpha)$ is decreasing when t_{i_0} is smaller than c_2/c_3, and increasing when t_{i_0} is larger than c_2/c_3. Hence the optimal solution is achieved by the lower boundary of co (Ω) that covers the point c_2/c_3. For obj_1, from (2), $c_2/c_3 = \sum_i t_i/N$. For obj_2, from (3), $c_2/c_3 = (t_1 + t_N)/2$. We can therefore conclude:

Theorem 3: The optimal solution for the distance objective obj_1 is the section of the lower boundary of co (Ω) that covers the point $\sum_i t_i/N$. The optimal solution for the area objective obj_2 is the section of the lower boundary of co (Ω) that covers the point $(t_1 + t_N)/2$. The overall time for finding the optimal solution for both obj_1 and obj_2 are of order $O(N)$.

C.3 obj_3

After obtaining the lower boundary of co (Ω), we can walk through all the points and count how many points in Ω are on each section of the boundary. Notice that this counting procedure can be combined with algorithm Convex_Hull_L so that we can count the numbers on the fly. Either way, the complexity is $O(N)$.

Theorem 4: The section in the lower boundary with the most points in Ω is the optimal solution under obj_3, and it can be obtained in time $O(N)$.

IV. Clock Skew Corrections With Clock Resets

We now consider the general problem with clock resets. In this section, we focus only on those clock resets that perform instantaneous time adjustments. The type of smooth velocity adjustments used in NTP will be discussed in the next section.

As mentioned in the previous section, in real measurements we do not expect to have many such instantaneous clock resets. We focus on the problem with a fixed number of resets. We assume that there is no change in clock speeds before and after clock resets. Therefore, the skew lines before and after clock resets should have the same slope. We can use the three objective functions described in the previous section to measure the goodness of a skew slope.

In the event of clock resets, we would observe a supporting straight line for a duration of time and then an abrupt shift of the delay level followed by another supporting straight line of the same slope. This abrupt shift of the delay level is likely due to the clock reset, though it could be for other reasons, such as the failure of a router, resulting in different routing of the packets.

We can therefore base our analysis on these characteristics of the data to determine the clock skew and obtain the correct measurement of the end-to-end delay between the two hosts.

We start by considering the case with one clock reset during the entire measurement, and then extend the approach to the general case with a bounded number of clock resets.

A. One Clock Reset

Suppose there is only one clock reset during the entire trace. Suppose the clock reset time is time t_{k+1}. We will then vary k to obtain the best t_{k+1} as the final solution for the clock reset time.

Suppose the supporting clock skew lines in the two sections (i.e., the sections before and after time t_{k+1}) are $L_1 := \{(x, y)|y = \alpha x + \beta_1, x \le t_k\}$, and $L_2 := \{(x, y)|y = \alpha x + \beta_2, x \ge t_{k+1}\}$. Let $\Omega_1 := \{v_i = (t_i, d_i) : i = 1, \ldots, k_1\}, \Omega_2 := \{v_i = (t_i, d_i) : i = k_1 + 1, \ldots, N\}$. We require all the points in Ω_1 and Ω_2 to be above lines L_1 and L_2, respectively. This restriction can be expressed as

$$\begin{aligned} \alpha t_i + \beta_1 &\le d_i, \qquad x \le t_k; \\ \alpha t_i + \beta_2 &\le d_i, \qquad x > t_k. \end{aligned}$$

Among all the lines that satisfy this condition we would like to choose the one that is closest to Ω_1 and Ω_2.

We next discuss the three different objective functions described earlier for measuring the closeness of the skew line to the set Ω.

(1) Minimize the sum of the vertical distances between the points and the line.

$$obj_1 := \sum_{i=1}^{N} d_i - \sum_{i=1}^{N} t_i\,\alpha - k\beta_1 - (N - k)\beta_2 \tag{6}$$

(2) Minimize the area between the curve and the line.

Summing over the area between the line $y = \alpha x + \beta$ and the line segment between every two consecutive points in Ω gives

$$obj_2 := \sum_{i=1}^{k-1} \frac{(d_i + d_{i+1})(t_{i+1} - t_i)}{2} + \sum_{i=k+1}^{N-1} \frac{(d_i + d_{i+1})(t_{i+1} - t_i)}{2}$$
$$- \frac{t_N^2 - t_{k+1}^2 + t_k^2 - t_1^2}{2}\,\alpha - (t_k - t_1)\beta_1 - (t_N - t_{k+1})\beta_2. \tag{7}$$

Unlike the case with no clock resets, even when the measurement points are equally spaced, i.e., $t_{i+1} - t_i = c$, obj_1 and obj_2 are no longer related by a linear equation. The optimal solution for the two objective functions, therefore, are not always the same.

(3) Maximize the number of points on the line.

Counting the number of points over the two sections, we have,

$$obj_3 := \sum_{i=1}^{k} 1_{\{d_i=\alpha t_i+\beta_1\}} + \sum_{i=k+1}^{N} 1_{\{d_i=\alpha t_i+\beta_2\}}. \quad (8)$$

In spite of being nonlinear, the maximization problem for this objective function can still be solved in time $O(N)$.

We next develop linear time algorithms for these three objective functions by computing the convex hull of the two sections of Ω.

For any fixed α, and given c_1, c_2, c_3, c_4, by first taking the minimum over β_1 and β_2, we define

$$\begin{aligned} f(\alpha) \quad = \quad & \min_{\beta_1,\beta_2} \quad c_1 - c_2\alpha - c_3\beta_1 - c_4\beta_2 \\ & \text{s.t.} \quad t_i\alpha + \beta_1 \le d_i, \quad i = 1, \ldots, k; \\ & \qquad t_i\alpha + \beta_2 \le d_i, \quad i = k+1, \ldots, N. \end{aligned}$$

We can then take the minimum over α and obtain:

Proposition 5: The minimization over α, β_1, β_2 can be sequentialized, i.e.,

$$\begin{aligned} \min_{\alpha} \ f(\alpha) \quad = \quad & \min_{\alpha,\beta_1,\beta_2} \quad c_1 - c_2\alpha - c_3\beta_1 - c_4\beta_2 \\ & \text{s.t.} \quad t_i\alpha + \beta_1 \le d_i, \quad i = 1, \ldots, k; \\ & \qquad t_i\alpha + \beta_2 \le d_i, \quad i = k+1, \ldots, N. \end{aligned}$$

We assume $c_3, c_4 > 0$, which is satisfied by both obj_1 and obj_2. By taking a closer look at the function $f(\alpha)$, we can prove the following key structural property.

Theorem 6: $f(\alpha)$ is convex in α.

Proof: From the constraints, we have

$$\begin{aligned} \beta_1 &\le d_i - t_i\alpha, && i = 1, \ldots, k; \\ \beta_2 &\le d_i - t_i\alpha, && i = k+1, \ldots, N. \end{aligned}$$

Since $c_3, c_4 > 0$,

$$\begin{aligned} f(\alpha) \ &= \ c_1 - c_2\alpha - c_3 \min_{1\le i\le k}\{d_i - t_i\alpha\} - c_4 \min_{k<i\le N}\{d_i - t_i\alpha\} \\ &= c_1 - c_2\alpha + c_3 \max_{1\le i\le k}\{t_i\alpha - d_i\} + c_4 \max_{k<i\le N}\{t_i\alpha - d_i\} \quad (9) \end{aligned}$$

The result follows because max is a convex function. □

Following the same approach as in Section III-C.2, since $f(\cdot)$ is convex, it suffices to find a local minimal solution. Suppose for $\alpha_1 < \alpha_2$, the points where the optimal solutions for $f(\alpha_1)$ and $f(\alpha_2)$ touch co (Ω), are the same extreme point. Assume this extreme point is (t_{i_1}, d_{i_1}) for $i_1 \le k$, and (t_{i_2}, d_{i_2}) for $i_2 > k$. This means that the maxima over i in (9) are achieved at i_1 and i_2, respectively. Therefore,

$$f(\alpha_j) = c_1 - c_2\alpha_j + c_3(t_{i_1}\alpha_j - d_{i_1}) + c_4(t_{i_2}\alpha_j - d_{i_2}), \quad j = 1, 2.$$

Then, $f(\alpha_1) - f(\alpha_2) = (c_3t_{i_1} + c_4t_{i_2} - c_2)(\alpha_1 - \alpha_2)$. Therefore,

$$f(\alpha_1) < f(\alpha_2) \iff c_3t_{i_1} + c_4t_{i_2} > c_2. \quad (10)$$

This shows that when $c_3t_{i_1} + c_4t_{i_2} \le c_2$, $f(\alpha)$ is decreasing. Conversely, if $c_3t_{i_1} + c_4t_{i_2} \ge c_2$, $f(\alpha)$ is increasing. Notice that t_{i_1} and t_{i_2} are increasing functions of α. Hence the optimal solution is achieved by the lower boundary of co (Ω) that satisfies $c_3t_{i_1} + c_4t_{i_2} = c_2$. For obj_1, from (6), condition (10) becomes

$$kt_{i_1} + (N-k)t_{i_2} = \sum_i t_i. \quad (11)$$

And for obj_2, from (7), condition (10) becomes

$$(t_k - t_1)t_{i_1} + (t_N - t_{k+1})t_{i_2} = (t_N^2 - t_{k+1}^2 + t_k^2 - t_1^2)/2. \quad (12)$$

Therefore, we have the following theorem:

Theorem 7: For the fixed clock reset at time t_{k+1}, the optimal solution for the distance objective, obj_1 is the section of the lower boundary of co (Ω) that satisfies condition (11). The optimal solution for the area objective, obj_2 is the section of the lower boundary of co (Ω) that satisfies condition (12). The overall time for finding the optimal solutions for both obj_1 and obj_2 are of order $O(N)$.

B. Multiple Clock Resets

The results in the previous section for the single clock reset can be easily extended to the case with multiple clock resets. Assume first that R clock reset times $\{ t_{k_1+1}, t_{k_2+1}, \ldots, t_{k_R+1} \}, (1 < k_1 < k_2 < \cdots < k_R < N-1)$ are given. These reset times divide all the points into $R+1$ sections,

$$\begin{aligned} \Omega_1 \quad &:= \quad \{v_i = (t_i, d_i) : i = 1, \ldots, k_1\}, \\ &\vdots \\ \Omega_{R+1} \quad &:= \quad \{v_i = (t_i, d_i) : i = k_R + 1, \ldots, N\}. \end{aligned} \quad (13)$$

We would like to find the best skew lines in the $R+1$ segments such that they have the same slope and are close to $\Omega_j, j = 1, \ldots, R+1$.

We first obtain the convex hull of each section $\Omega_j, j = 1, \ldots, R+1$. The optimal skew line must touch at least one point in each section. Assume $t_{i_1}, \ldots, t_{i_{R+1}}$ are the extreme points that the optimal skew lines touch in each section.

For the distance objective obj_1, condition (11) generalizes to

$$k_1t_{i_1} + (k_2 - k_1)t_{i_2} + \cdots + (N - k_R)t_{i_{R+1}} = \sum_i t_i. \quad (14)$$

For the area objective obj_2, condition (12) generalizes to

$$(t_{k_1} - t_1)t_{i_1} + (t_{k_2} - t_{k_1+1})t_{i_2} + \cdots + (t_N - t_{k_R+1})t_{i_{R+1}} = V/2, \quad (15)$$

where $V = t_{k_1}^2 - t_1^2 + t_{k_2}^2 - t_{k_1+1}^2 + \cdots + t_N^2 - t_{k_R+1}^2$. Further notice that the time of the touching vertices $t_{i_1}, \ldots, t_{i_{R+1}}$ increase as α increases. We therefore have the same result as Theorem 7 with the generalized conditions (14) and (15) for the given multiple clock resets.

Theorem 8: For the fixed clock resets at times $\{ t_{k_1+1}, \ldots, t_{k_R+1} \}, (1 < k_1 < \cdots < k_R < N-1)$, the optimal solution for the distance objective, obj_1 is the section of the lower boundary of co (Ω) that satisfies condition (14). The optimal solution for the area objective, obj_2 is the section of the lower boundary of co (Ω) that satisfies condition (15). The overall time for finding the optimal solutions for both obj_1 and obj_2 are of order $O(N)$.

Based on these results we provide the following algorithm to identify the optimal clock skew slope α for a given set of clock reset times, $\{ t_{k_1+1}, t_{k_2+1}, \ldots, t_{k_R+1} \}$. We set $k_0 = 0, k_{R+1} = N+1$, to simplify the notation. We first apply Algorithm Convex_Hull_L for each section $\Omega_j, j = 1, \ldots, R+1$, to obtain the lower convex hulls. Let $convex^j = (k_j + 1, \ldots, k_{j+1})$ be the indices of the vertices of the lower convex hull of Ω_j.

Algorithm Identify_Best_Alpha:

(1) Initialize:
$index[i] = k_i + 1, \quad i = 0, \ldots, R;$
$slope[i] = slope(v_{index[i]}, v_{index[i]+1}), \quad i = 0, \ldots, R;$
Set LHS and RHS;
(2) While ($LHS < RHS$)
$segment = \arg\min\{slope[0], \ldots, slope[R]\};$
$index[segment]$ = next_index_in $convex^{segment}$;
$\alpha = slope[segment]$;
Update $slope[segment]$;
Update LHS;
(3) Output slope α and indices $index[0], \ldots, index[R]$;
End.

For objective functions obj_1, or obj_2, LHS and RHS denote the left hand side and right hand side of equation (14), or of equation (15), respectively.

With the assumption that there is at most one clock reset every p units of time ($p = N/(R+1)$), we can search through every possible clock reset combination, and apply Algorithm Identify_Best_Alpha for each of the combinations. The clock skew slope is then the best solution among all the possible clock reset combinations. The overall algorithm can be summarized as follows:

Algorithm R-Resets:

(1) Loop through $k_i = \frac{(i-1)N}{p} + 1, \ldots, \frac{iN}{p}, \quad i = 1, \ldots, R;$
Define $\Omega_1, \ldots, \Omega_{R+1}$ according to (13);
Apply Algorithm Convex_Hull_L, obtain co (Ω_i), $i = 1 \ldots, R;$
Apply Algorithm Identify_Best_Alpha,
obtain best α for given $k_1, \ldots, k_R$;
Record current best solution and minimum objective value;
End loop;
(2) Output best slope α and clock reset times;
End.

If there are R clock resets, then there are R loops each with N/R possibilities. Algorithm Convex_Hull_L runs in time $O(N)$ and Algorithm Identify_Best_Alpha runs in time $O(NR)$. Therefore, the overall complexity of this algorithm is $O((N/R)^R NR)$.

C. Identifying Number and Time Epochs of Clock Resets

The major component in the complexity of the general R-Reset algorithm lies in the combinatorial search for all possible clock reset points. To further reduce the complexity of the algorithm, we study heuristic algorithms to identify the number of clock resets and where they occur.

We use a divide-and-conquer approach to identify the occurrences of clock resets. First, the whole data set is divided into intervals. These intervals should be wide enough so that the structural property of the delay measurements can be observed within each interval. In particular, one should observe a supporting straight line underneath the delay points in each interval. On the other hand, these intervals need to be narrow enough so that there is at most one clock reset within any three consecutive intervals. We then apply Algorithm Convex_Hull_L and Theorem 3 to identify the best skew lines within each interval. We compare the skew lines for two adjacent intervals by calculating the maximum distance between the two skew lines inside the two intervals. The two skew lines are considered to be the same if the maximum distance is smaller than some given tolerance level. For each interval, if the skew line is different from any of its two neighboring intervals then this interval is marked with the possibility of containing a clock reset. Because a clock reset can result in two consecutive marked intervals, we need to merge the adjacent marked intervals so that we can infer there is exactly one clock reset within each marked interval. We can then apply the linear search algorithm for one clock reset, i.e., Theorem 7, to identify the clock reset within each marked interval. We can also use the skew slopes in the (unmarked) intervals without clock resets to identify the best clock reset within the marked intervals. These two approaches have the same complexity and provide the same results in practice. The collection of all the resets within marked intervals are all the reset points.

Algorithm Divide_And_Conquer:

(1) Divide all the data into intervals of width w;
(2) For each interval, apply Algorithm Convex_Hull_L and Theorem 3 to identify best skew lines;
(3) For each interval,
- compare its skew line with neighbor skew lines;
- set marks for possible clock resets;
(4) Merge marked intervals to form intervals with exactly one clock reset;
(5) For each marked interval, identify the best clock reset by
- the linear search algorithm for one clock reset (Theorem 7);
- or linear search for one clock reset with slope given by some average of the slopes in the unmarked intervals;
(6) Merge the clock resets in all the marked intervals.

This algorithm has the advantage that it identifies the number of clock resets, instead of being supplied with the number of clock resets ahead of time. It agrees with the intuition from our visual observations for reset points. Furthermore, this algorithm has complexity $O(Nw)$, linear in N, which makes it very efficient. This algorithm provides the correct answer under the following assumptions. First, it assumes that clock resets do not happen very often and that one can identify the minimal distance between two clock resets. One also needs to specify a tolerance level for the comparison of two skew lines. This tolerance level can be interpreted as the accuracy of the clocks. These assumptions are fairly minimal, and hold true for most real system clocks, making the algorithm attractive.

Another approach to identify the clock resets is to consider the two one-way-delay data between the two machines. The supporting straight lines for the two one-way-delay points have symmetric slopes. When there is a clock reset, the delay points would shift up for one data direction and shift down of the other

data direction. By considering this observation and applying Algorithm Convex_Hull_L we can design a marching algorithm to identify the clock resets which runs in time $O(N)$. This algorithm, as well as the divide-and-conquer algorithm can be applied to address the clock skew problem with NTP. We defer the detailed description and discussion of this marching algorithm until Section V.

After identifying all the clock resets, one can then apply Algorithm Identify_Best_Alpha to find the global optimal solution for the skew lines with the clock resets given by the above heuristics.

Remark: We further observe that the above algorithms can be applied for the counting objective, obj_3, if we use a simple counter to record the number of points on the skew line as its slope increases. The complexity of the algorithm stays the same.

V. Clock Skew Corrections With Velocity Adjustments

The Network Time Protocol (NTP) [3] is used to synchronize the time of a computer to another server or reference time source, such as a radio or satellite receiver. Each computer can communicate with multiple peers and reference time sources. At every synchronization point, NTP determines if the clock setting needs to be adjusted, (possibly) adjusts the speed of the computer clock, and computes the time for the next synchronization. All the decisions and computations are with regard to previous synchronization points. Except at the first clock synchronization point, which is usually when the computer boots up, NTP only changes the speed of the clock. It does not reset the clock. The clock adjustment information is logged in a set of files stored on the computer.

In general, NTP can provide sub-millisecond accuracy on LANs, and low tens of millisecond accuracy on WANs [3]. The delay measurements between two computers depends on the clocks on both machines. These machines may be both running NTP. The status of the remote clock is usually unknown, which results in inaccurate measurements. We can improve the accuracy of such measurements in several ways. If the remote machine is not running NTP, we can first adjust the measurements according to clock corrections from the local NTP (or `cron`) log files. We then apply the clock reset algorithms in Sections III and IV to obtain the clock skew for the remote machine, and adjust the measurements. Since the local NTP (or `cron`) log files are updated online, this whole approach can be applied online as well.

In the event that the remote machine is running NTP, and the log files are not readily available from either machine, we need to estimate the combined effects of the velocity adjustments from both machines. In the sequel of this section, we discuss this situation.

A. Piecewise Linear Skew Lines

From the measurement data, one would observe a supporting straight line underneath the delay points, and then at a certain point the supporting line would change its slope. This is the piecewise linear skew lines for measurements with NTP versus the parallel linear skew lines in Section IV.

As before, we assume a finite collection of measurement points, $\Omega = \{v_i = (t_i, d_i) : i = 1, \ldots, N\}$. We further assume that the R clock reset times $\{ x_1, x_2, \ldots, x_R \}$ are given. Assume $t_{k_i-1} < x_i \leq t_{k_i}, \quad i = 1, \ldots, R$, where $1 < k_1 < k_2 < \cdots < k_R < N-1$). These reset times divide all the points into $R+1$ sections, $\Omega_1, \ldots, \Omega_{R+1}$. We would like to find the best piecewise linear supporting lines in the $R+1$ segments such that they change their slopes at the given reset times and that they are close to $\Omega_j, j = 1, \ldots, R+1$.

Similar to Section IV, this problem can be formulated as a linear program. Let $\{ (t_1, b_0), (x_1, b_1), \ldots, (x_R, b_R), (t_N, b_N) \}$ be the turning points for the piecewise linear supporting lines. Here, $b_0, b_1, \ldots, b_R, b_N$, are the variables we need to solve. For convenience, let $k_0 = 1, k_{R+1} = N, x_0 = t_1, x_{R+1} = t_N$. We require all the measurement points be above the line segments. This leads to the following set of linear constraints:
For all $i = k_j, \ldots, k_{j+1} - 1, j = 0, \ldots, R$:

$$d_i - b_j \quad \geq \quad \frac{b_{j+1} - b_j}{x_{j+1} - x_j}(t_i - x_j).$$

After rearranging the terms these constraints can be written in the following form:

$$\begin{aligned} \bar{c}_{01} b_0 + \bar{c}_{02} b_1 \quad & \leq \quad \bar{d}_0 \\ \ddots \quad & \vdots \\ \bar{c}_{R1} b_R + \bar{c}_{R2} b_{R+1} \quad & \leq \quad \bar{d}_R \end{aligned} \tag{16}$$

where $\bar{c}$ and $\bar{d}$ are easily obtained from the data in Ω. We can use both the distance (obj_1) and area (obj_2) objective functions defined in Sections III and IV to measure the closeness of the measurement points and the skew lines. These two objective are both linear functions of the variables $b_0, b_1, \ldots, b_R, b_N$. One can then apply the linear programming algorithms to find the optimal solution which minimizes obj_1 and obj_2 subject to the constraints in (16). We further remark that the special staircase constraints in (16) allow the decomposition techniques in linear programming to be readily applied. The decomposition algorithm, however, does not guarantee $O(N)$ complexity.

For the special case that there is only one turning point, this linear program can be solved in time $O(N)$. In this case, we would like to solve for b_0, b_1, b_N. We can express both b_0 and b_N in terms of b_1 and prove that the linear objective functions are convex in b_1. We can then apply the same techniques in Section IV to find a local minimum solution for b_1, by finding the convex hull for the points in each section and searching through b_1. This locally minimal solution must be globally minimal due to the convex property of the objective functions.

When there are more turning points, however, this approach does not always give the optimal solution. The key for this approach to work is that the optimal piecewise linear skew lines touch at least one measurement point within each of the $R+1$ sections. If this assumption holds, one can then apply the same search algorithm on b_0 to obtain a local optimum solution which must be globally optimal due to the convex property.

B. Identifying Number and Time Epochs of Velocity Adjustments

It remains to find the turning points in the piecewise linear skew lines. Because the skew line would have a different slope

when there is a turning point in the interval, Algorithm Divide_And_Conquer can be used to identify the sub-intervals with exactly one turning point. One can then apply the linear algorithm for one turning point in the previous section to identify the best turning point.

Another approach for identifying the turning points is to examine the two one-way delay data points. We can obtain the measurements for the one-way delay from the source machine to the destination machine and at the same time, the one-way delay from the destination machine back to the source machine. When there is a clock reset (turning point), one of the one-way delay measurements would turn up and the other one-way delay measurement would turn down. This is best illustrated in Figure 3 for cron and Figure 5 for NTP. This phenomenon would make the skew line for the upturning plot unchanged and the skew line for the down turning plot rotate down. The skew lines for these two one-way measurements should have symmetric slopes, i.e., the sum of the slopes should be close to zero. This marching algorithm would march over time and consider the measurement points from the two one-way delay points, one point at a time. It would update the convex hull and the best skew line for each set of the one-way delay points. When the sum of the slopes is small enough and incorporating the new point makes the sum farther away from origin, one would consider this new point a turning point.

Marching_Algorithm:

(1) Initialization:
- Adjust the initial time offsets;
- take one point from each one-way delay measurements;

(2) March over time. Select the next point until the end;
- apply Algorithm Convex_Hull_L and Theorem 3 update best skew lines for each one-way delay;

(3) Check condition for turning points
- If the sum of the slopes is small enough and the new point changes this sum away from origin, then
  + this new point is a turning point;
  + print out the previous section;
  + start a new section;

(4) Goto Step (2);

This algorithm identifies the number of clock resets, instead of being supplied with the number of clock resets ahead of time. The complexity of this algorithm is $O(N)$, which is very efficient. The algorithm agrees with the intuition from our visual observations for turning points by comparing two one-way delay measurements. Combining the two one-way delay measurements takes into account more information, and hence has potential to be more accurate. One also obtains the best skew lines in each section from this algorithm. We can certainly build features into this algorithm such as the minimal time between reset points and tolerance levels for skew slopes. In order for this algorithm to provide the correct answer, one needs to have fairly stable measurement data. The most attractive feature of this algorithm is that it is adaptive, and can be readily applied online due to its nature of marching over time.

One drawback of this algorithm is that it is quite sensitive to network congestion. It may produce false turning points. And the error during each step of the algorithm can propagate in later steps. One way to reduce the number of false turning points is by weakening the conditions for detecting the turning points. This approach, however, would lead to the late detection of the turning points, which causes larger errors to propagate into future estimates. One can resolve this issue by keeping a short history list of the last certain number of points. When the algorithm detects a turning point, it would use the best candidate in the history list as the true turning point to calculate the skew lines. The algorithm then would march on from this true turning point.

VI. Online Estimation and Correction

Many real time applications require the skew of the delay measurements to be corrected online. Being able to correct the clock skews online provides better flexibility and adaptivity for the applications. Correcting the clock skews online can also be used as a means for active clock synchronization. It can be an alternative to or be used in combination with other clock synchronization algorithms such as NTP.

We now study the techniques from the online perspective. Algorithm Convex_Hull_L scans through each measurement point in increasing order and builds a stack to store the lower convex hull of the previous points. It can obviously be applied online, as measurement points accumulate. Therefore, for the case with no clock resets, the algorithm can be applied in an online manner, with the same complexity.

Consider the case with possible clock resets. We assume there is at most one clock reset every p units of time, which is generally the case in real systems. We apply Algorithm R_Resets for an initial set of measurement data, up to time τ. We then fix the optimal clock reset times produced by the algorithm prior to time $\tau - p$. We label the last fixed clock reset time to be $t_{last\_fixed\_reset}$. As more measurement points become available, until time $t_{last\_fixed\_reset} + 2p$, we perform the algorithm allowing only one clock reset after $t_{last\_fixed\_reset}$. At the first measurement point after time $t_{last\_fixed\_reset} + 2p$, we perform the algorithm allowing two clock resets after $t_{last\_fixed\_reset}$. We then fix the first of the two clock resets and update $t_{last\_fixed\_reset}$. This procedure is repeated until some time in the future when part of the history data are discarded and the variables are updated to make the results more representative of the recent data.

For the case with NTP, as discussed in Section V, the marching algorithm is already an online algorithm, which is its attractive feature.

An advantage of applying the algorithms online is that we can have the current best clock skew estimate. Using the remote clock as a reference source, we can adjust the local clock according to the skew estimate at certain well defined synchronization points. These synchronization points can be defined according to the previous data, for example, when the skew correction reaches a certain threshold.

VII. Experiments

In this section we present an experimental study of the algorithms by applying them to real network delay measurements. We collected packet delay traces over the Internet as well as within the IBM firewall using the `ping` and `tcpdump` utility.

We used `ping` program to send icmp packets between two machines every second. On both machines we used the `tcpdump` tool to collect the icmp packet information, including the packet id and timestamp. We then have available the following time stamps for each round trip measurement by `ping`:

s_1: time that sender sent out the icmp request packet, according to sender's clock;
s_2: time that receiver received the icmp request packet, according to receiver's clock;
s_3: time that receiver sent out the icmp reply packet, according to receiver's clock;
s_4: time that sender received the icmp reply packet, according to sender's clock.

The differences of the time stamps $s_2 - s_1$ and $s_4 - s_3$ are the two one-way delay measurements. We collected these data every one to four seconds, over the duration of a couple of hours to one day, between machines in New York and Nice, and between New York and Beijing. We apply the algorithms in Sections IV and V to obtain the clock skew lines.

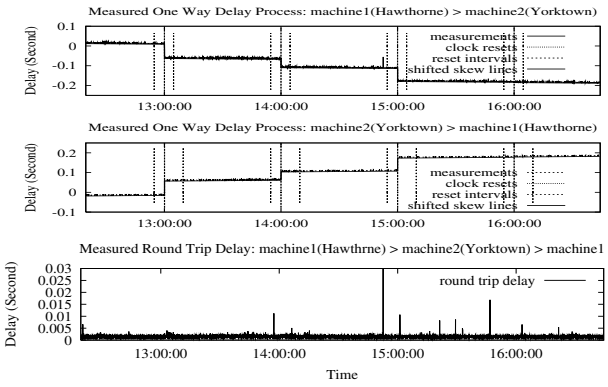

Fig. 3. Measurements with clock resets over LAN

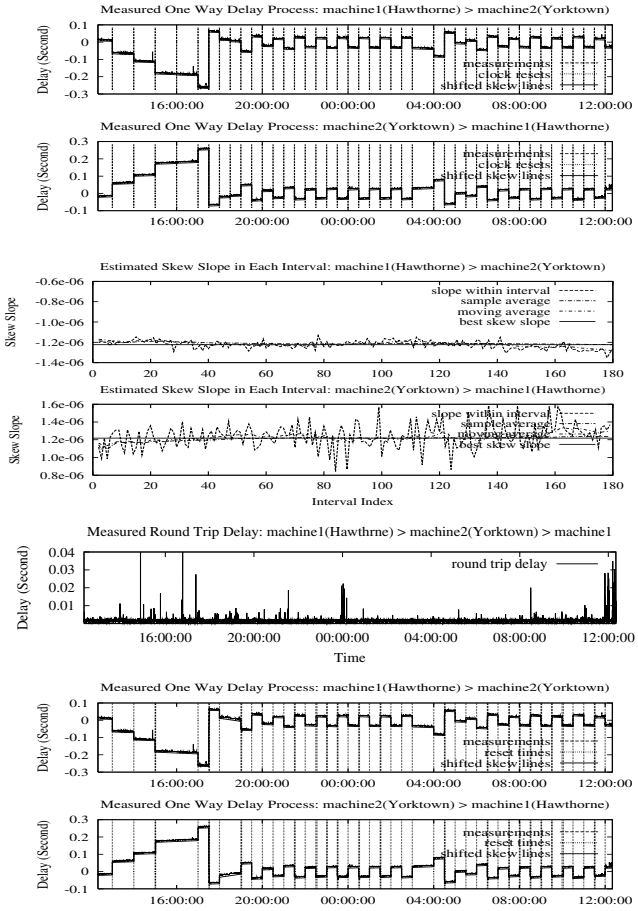

Fig. 4. Measurements with clock resets over LAN

Figure 3 presents the measurements between two closely located machines within 10 miles apart, connected by a corporate network. The top two plots show the two one-way delay measurements. The bottom plot shows the round trip delay between the two machines. The horizontal axis of all the plots are the time the packets were sent from the machines. One of the two machines resets its clock regularly with a standard clock server while the other machine does not have any time synchronization scheme. As can be seem from the one-way delay plots there are four clock resets all happening on the hour. At the time of the clock resets, the first one-way delay plot shifts down while the second one-way delay plot shifts up by an equal amount. We applied Algorithm Divide_and_Conquer to divide the whole time into 300 second sub-intervals and detected the intermediate intervals with exactly one clock reset. The longest dividing lines in the plot are the reset times that the algorithm detected, which are exactly the time the machine resets its clock. The algorithm was implemented in C, and finds the best skew lines within a second on a 333MHz workstation.

Figure 4 presents the measurements between the same two machines for the duration of 24 hours. Both of the machines reset their own clocks regularly. The top 2 plots show the delay data, intervals with resets and the clock reset times from the divide and conquer algorithm. The algorithm correctly detected all the clock resets and found the best skew lines. The third and fourth plots show the skew slopes for each interval with no resets, their running averages, exponentially weighted moving averages, and the global optimal skew slopes. Interestingly, we observe that the skew slopes are more stable on the forward path, with less than 10% deviations, while the deviation on reverse path is around 30%. This difference is probably due to the fact that the forward and reverse paths go through different links. The absolute values of the slopes are on the order of 10^{-6}, which means the two clocks can be synchronized to millisecond level accuracy if they adjust with respect to each other every hour. To solve the global optimal skew slopes by taking into account both one-way delay measurements such that the two skew slopes are opposite to each other, we can put together the two one-way delay data, one in the forward direction and one in the backward direction. This changes the skew slope of the backward data into opposite direction. We then apply the divide and conquer algorithm to solve for the global optimal skew slope. The fifth plot in Figure 4 shows the round trip delay data on the order of sub-milliseconds. We also applied the marching algorithm to this data set to find the reset points and the best skew slopes. The running time for the marching algorithm is comparable to the

divide and conquer algorithm. The bottom two plots in Figure 4 show the results. We notice that the marching algorithm detected most of the clock resets except the two small clock resets at time `16:00:00` and `18:30:00`, while the divide and conquer algorithm was able to detect them. In the case of clock resets, we further point out that although the marching algorithm can be applied adaptively and online, it does not give the global optimal solutions, in contrast to the divide and conquer algorithm.

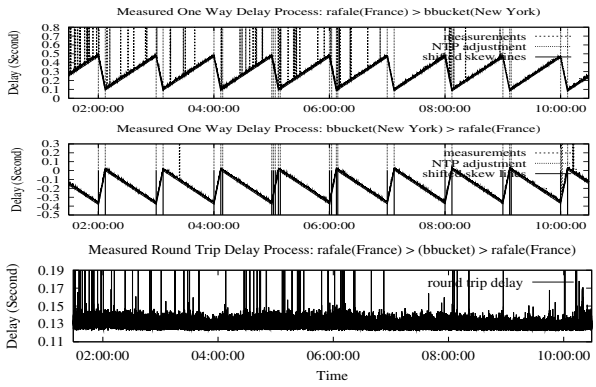

Fig. 5. Measurements with clock resets using `ntpdate` between New York and Nice

We also performed an experiment between two machines in New York and Nice. One machine uses `ntpdate` to adjusts its clock regularly and the other does not adjust its clock. The way this new version of `ntpdate` adjusts the computer clock is by changing the speed of the clock for a duration of time and then letting the clock run at its own speed. This is different from `rdate` in the sense that `rdate` simply resets the clock to the correct time and lets the clock run. The top two plots in Figure 5 show the two one-way delay measurements and the results of applying the marching algorithm. The algorithms identified all the true turning points and the correct skew lines, which was validated against the NTP logs on the machine. We notice that several false turning points were identified due to Internet congestion and the relatively large delay jitter. The skew slopes between these two machines are on the order of 10^{-4}.

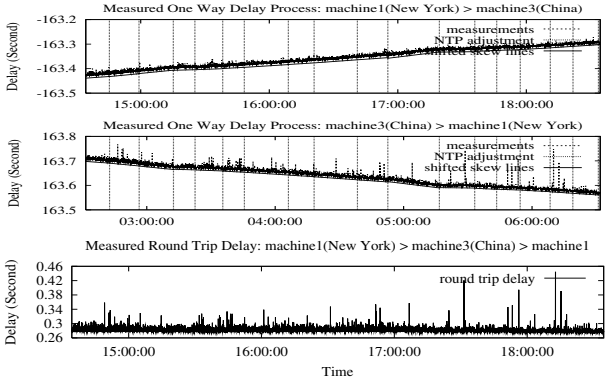

Fig. 6. Measurements with NTP between New York and Beijing

We further performed an experiment between two machines in New York and Beijing. One machine uses NTP to adjust its clock and the other does not adjust its clock. As in the previous example, Figure 6 shows the two one-way delay measurements and the round trip delay. The marching algorithm worked well and found the proper turning points and skew lines. The skew slopes between these two machines are on the order of 10^{-5}.

Remark: Although obj_1 and obj_2 are different technically, they provided the same solutions in all our experiments. This is because the data we collected are evenly distributed. obj_3 also provides the "correct" answer, together with some false answers. This is because it is unlikely for many measurement points to be on exactly the same line. obj_1 and obj_2 are therefore preferable choices.

Compared with previous work, for the case with no clock resets, our convex hull algorithm is essentially the same as the linear programming algorithm in [4]. The convex hull algorithm is more visual and intuitive. The linear regression algorithm in [5] is more sensitive to the network congestion as discussed in [4], and also as observed by our experiments for the data in Figure 1. For the case with clock resets and with velocity adjustments, the algorithms in [4], [5] would provide the wrong answer because they cannot account for the jumps and direction changes in the delay measurements.

VIII. Summary

To summarize, in this paper we studied algorithms for adjusting the delay measurements to obtain more accurate results. These algorithms, based on a convex hull approach, are intuitive and computationally efficient. Furthermore, these algorithms can be applied online in an adaptive manner. The most significant contribution of this study is that it solved the problem for the cases with clock resets and with velocity adjustments (such as NTP).

These algorithms have a wide range of applications, from computer systems to communication networks, from local area network to the Internet domain, from traffic routing to application tuning, from network management to QoS control. They can be used not only to improve the Internet measurements, but also to actively synchronize clocks.

Acknowledgments: The authors are grateful to Douglas Freimuth, Yunhee Jang, Hong Li, Naceur Malouch, David Olshefski, Jehan Sanmugaraja, and Kun Song, for their help in setting up the experiments.

References

[1] Dyer, M.E., Linear Time Algorithms for Two- and Three-Variable Linear Programs. *SIAM Journal on Computing,* **13** (1983), 31-45.

[2] Megiddo, N., Linear-Time Algorithms for Linear Programs in R^3 and Related Problems. *SIAM Journal on Computing,* **12(4)** (1983), 759-776.

[3] Mills, D., Network Time Protocol (Version 3) - Specification, Implementation and Analysis, RFC 1305, University of Delaware, March 1992.

[4] Moon, S.B., Skelly, P. and Towsley, D., Estimation and Removal of Clock Skew from Network Delay Measurements. In *Proceedings of the IEEE INFOCOM Conference on Computer Communications,* page 227-234, March 1999.

[5] Paxson, V., On Calibrating Measurements of Packet Transit Times. In *Proceedings of the ACM SIGMETRICS,* Madison, Wisconsin, June 1998.

[6] Rockafellar, R.T., Convex Analysis. Princeton Univ. Press, 1970.