On Calibrating Measurements of Packet Transit Times

Vern Paxson
Network Research Group
Lawrence Berkeley National Laboratory*
University of California, Berkeley
vern@ee.lbl.gov

LBNL-41535

Abstract

We discuss the problem of detecting errors in measurements of the total delay experienced by packets transmitted through a wide-area network. We assume that we have measurements of the transmission times of a group of packets sent from an originating host, A, and a corresponding set of measurements of their arrival times at their destination host, B, recorded by two separate clocks. We also assume that we have a similar series of measurements of packets sent from B to A (as might occur when recording a TCP connection), but we do not assume that the clock at A is synchronized with the clock at B, nor that they run at the same frequency. We develop robust algorithms for detecting abrupt adjustments to either clock, and for estimating the relative skew between the clocks. By analyzing a large set of measurements of Internet TCP connections, we find that both clock adjustments and relative skew are sufficiently common that failing to detect them can lead to potentially large errors when analyzing packet transit times. We further find that synchronizing clocks using a network time protocol such as NTP does not free them from such errors.

1 Introduction

In this paper we tackle the problem of how to calibrate transit times measured for packets traveling through a network. We assume that we have a series of pairs of timings, recording each packet's departure time from its sender and arrival time at its receiver, but that the clocks used at the sender and receiver to generate these timestamps are not necessarily accurate: they may not keep true time, they may be subject to abrupt adjustments, and they may run at different rates.

Calibrating transit times might at first blush appear to be a fairly minor measurement problem, but in fact it is potentially central to the accuracy of a number of wide-area network measurement techniques. The timing structure of packets transmitted through a network is very rich: by carefully analyzing this structure one can infer fundamental network properties such as delay, bottleneck link speed, available bandwidth, queueing levels, and even hop-by-hop link speed [Ke91, Bo93, Mu94, CC96, Pa97a, Ja97]. These measurements are usually made using "echo" techniques, in which packets sent to a given target result in the target returning replies back to the sender. The analysis is then made on the timing structure of the replies.

Echo-based techniques, however, suffer from a fundamental problem: they unavoidably conflate properties of the network path in the forward direction (which perturbs the sender's original packets, and hence the times at which the target generates its replies), with the properties of the reverse direction. Consequently, these measurements are subject to considerable inaccuracy. Furthermore, a large-scale study of Internet routing found that paths through the Internet are often *asymmetric*, meaning that the series of routers visited in the two directions often differ [Pa96]. Subsequent work discusses other asymmetries (such as link speeds and queueing levels), and argues strongly for "receiver-based" measurement, in which packet receivers cooperate with packet senders in order to accurately measure network traffic [Pa97a].

Accurate receiver-based measurement, however, depends on accurate comparisons of timestamps produced by a clock at the packet sender with those produced at the receiver. It is easy to assume that to ensure accuracy we merely require synchronization between these clocks; but, while a considerable body of work has addressed the problem of synchronizing clocks to true time (see especially the work of Mills [Mi92a, Mi92b, Mi95]), these algorithms maintain good time over time scales of hours to days. They do *not* assure synchronization on the small time scales of individual network connections (a point we develop in § 7). Consequently, the problem of calibrating the timestamps produced by pairs of network clocks remains interesting and important.

If undetected, clock adjustments and rate mismatches can introduce significant measurement errors. For example, if the sender's clock runs slower than the receiver's clock, then the series of one-way transit times (OTTs) that we compute from their timestamps will show a systematic increase across the measurement interval. It is easy to mistake this increase for a genuine increase in networking delays due to a gradual buildup of queues at a router along the network path. Similarly, a clock adjustment, if undetected, can lead to completely erroneous conclusions that the network suffered from sustained periods of high delay.

To develop our algorithms we used the data we gathered for the Internet packet dynamics study mentioned above [Pa97a]. We recorded two datasets, each consisting of traces of TCP transfers conducted at random between a number sites around the Internet. For each transfer, packet arrivals and departures were recorded at both the sender and the receiver using the `tcpdump` utility [JLM89]. The clocks used at the different sites were not necessarily synchronized.

* A shorter version of this paper appears in the Proceedings of SIGMETRICS '98. This work was supported by the Director, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division of the United States Department of Energy under Contract No. DE-AC03-76SF00098.

We term the sender and receiver traces collectively as a "trace pair." Transfers entailed the sender transmitting 100 KB of data to the receiver. Because of the use of TCP, this results in a stream of large data packets flowing from the sender to the receiver, and a smaller stream of acknowledgement ("ack") packets flowing in the other direction, all of which were recorded.

The first dataset, $\mathcal{N}_1$, recorded at the end of 1994, consists of 2,335 trace pairs between 25 sites. The second, $\mathcal{N}_2$, recorded at the end of 1995, consists of 15,492 trace pairs between 31 sites. (For brevity, we do not list the sites here, but will use the same names and font as in [Pa97a]—e.g., "`austr`".)

We wrote a program, `tcpanaly`, for automating much of the analysis of the trace pairs, and [Pa97b] discusses a number of packet filter measurement errors detected by it. Part of the development of `tcpanaly` included devising and implementing the clock calibration algorithms we discuss in this paper.

Limitations of the study. There are a number of important limitations of our study that must be kept in mind. The first is that the data we had available for analysis had been previously recorded, and did not include any clock information (such as whether the clocks were synchronized using NTP, nor any logs of clock adjustments by the operating systems). Because we were confined to post facto analysis, we were unable to evalute the accuracy of our algorithms in any absolute sense. Until the algorithms can be evaluated in a controlled fashion, they can at best only be regarded as promising but unproven.

The post facto analysis also means that we could not design our measurement traffic to best support the problem of calibrating the packet timings. Instead, we had to deal with TCP bulk transfer traffic, which often introduces its own timing distortions along the data transfer path by contributing to queueing. Consequently, we must deal with noise issues that we could otherwise avoid.

Another limitation is that we found we needed to introduce a number of heuristics into the algorithms. We believe for the most part that doing so is unavoidable, because the goal of the heuristics is to deal with noise induced on the packet transit timings by network conditions, and there is no known method for removing such noise.

Finally, one might argue that inexpensive, high precision timing synchronization devices, such as GPS units, obviate the need for calibration techniques such as those we develop. However, even though these units are now relatively cheap, it is not clear that we can yet presume their ubiquity, because: their cost remains non-negligible; they cannot always be deployed due constraints on antenna placement; and many sites might instead use NTP to synchronize most of their machines to a few GPS-endowed machines. We also argue in our summary that, even given a directly-attached GPS unit, checking the ultimate clock readings derived from it remains prudent.

We begin our discussion by defining in § 2 basic terminology for describing different clock attributes. In § 3 we introduce "relative" counterparts of these terms, for discussing potential disagreements between two network clocks. We next conduct an assessment of relative clock accuracy (§ 4), before turning to the development of methods for detecting clock adjustments (§ 5) and relative clock skew (§ 6). As mentioned above, clock adjustments and skew can introduce large, artificial network "dynamics," so it is important to detect and remove these effects.

We finish in § 7 with a look at how well a clock's synchronization correlates with stable clock behavior (lack of adjustments and of skew). We find that, unfortunately, a high degree of synchronization between two clocks does not necessarily mean that the clocks are free of relative errors.

Finally, the topics in this paper are discussed in greater depth in [Pa97c].

2 Basic clock terminology

In this section we define basic terminology for discussing the characteristics of the clocks used in our study. The Network Time Protocol (NTP; [Mi92a]) defines a nomenclature for discussing clock characteristics, which we will use as appropriate. It is important to note, however, that the main goal of NTP is to provide accurate timekeeping over fairly long time scales, such as minutes to days, while for our purposes we are concerned with much shorter-term accuracy, namely between the beginning of a network transfer and its end. This difference in goals sometimes leads to different definitions of terminology, as discussed below.

Resolution. A clock's *resolution* is the smallest unit by which the clock's time is updated (a "tick"). It gives a lower bound on the clock's uncertainty. Note that we define resolution relative to the clock's reported time and not to true time, so, for example, a resolution of 10 msec only means that the clock updates its notion of time in 0.01 second increments, not that this is the true amount of time between updates.

We estimate a clock's resolution by inspecting the differences between successive packet timestamps. In general, we take the smallest positive difference as an upper bound on the clock's resolution. However, we must not confuse "monotonicity increments" with true clock advances. These increments are added by some system clocks when the clock is read multiple times within the same tick, so that instead of reporting that zero time has elapsed, they always report monotone-increasing timestamps. These artificial increments are generally the smallest representable timestamp advance—1 μsec for Unix systems such as those in our study. So to robustly estimate resolution, we need to disregard very small apparent clock increments. Finally, we note that this approach for estimating clock resolution produces at best an upper bound on the clock's true resolution, because it may happen that the packet filter never receives back-to-back packets separated by only one tick. For measurement purposes, this inaccuracy is often acceptable, because the extra error introduced is conservative in the sense that it only widens the uncertainties associated with any subsequent timing analysis.

Offset. We define a clock's *offset* at a particular moment as the difference between the time reported by the clock and the "true" time as defined by national standards. If the clock reports a time T_c and the true time is T_t, then the clock's offset is $T_c - T_t$.

Accuracy. We will refer to a clock as *accurate* at a particular moment if the clock's offset is zero, and more generally a clock's *accuracy* is how close the absolute value of the offset is to zero. For NTP, accuracy also includes a notion of the frequency of the clock; for our purposes, we split out this notion into that of *skew*, because we define accuracy in terms of a single moment in time rather than over an interval of time.

Skew. A clock's *skew* at a particular moment is the frequency difference (first derivative of its offset with respect to true time) between the clock and national standards.

Drift. As noted in [Mi92a], real clocks exhibit some variation in skew. That is, the second derivative of the clock's offset with respect to true time is generally non-zero. [Mi92a] defines this quantity as the clock's *drift*. We in general will only talk about this notion in terms of clock *adjustments*, during which the clock's time is rapidly altered, because during the small time scales of interest for our study, only large drift values have discernible effects.[1]

[1] We will see in § 6 that, for the time scale of a single TCP connection in our study, relative clock skew is nearly always very close to linear, indicating near-zero relative drift over small time scales.

3 Terminology for comparing clocks

In this section we develop terminology for discussing differences between two clocks producing timestamps. The definitions are, for the most part, analogous to those in § 2, except that, instead of comparing a single clock against true time, we are comparing one clock against another.

We first introduce the meta-notation of a subscript "s" denoting time measured at the packet *sender*, and "r" denoting time at the packet *receiver*. Let C_s and C_r refer to the clocks at the sender and receiver, with R_s and R_r their respective resolutions.

We define C_r's offset relative to C_s at a particular true time T as $T_r - T_s$, that is, the instantaneous difference between the readings of C_r and C_s at time T. For convenience we will sometimes refer to this as C_r's relative offset at time T, with C_s implicitly being the clock to which C_r is compared. We discuss assessing the relative offset of one clock to another in § 4.

Similarly, C_r's relative skew is the first derivative of C_r's relative offset with respect to true time. Since we do not assume an independent means of measuring true time, we can only estimate C_r's relative skew in terms of time as measured by either C_s or C_r. See § 6 for further discussion.

If C_r is accurate relative to C_s (their relative offset is zero), then we will refer to the pair of clocks as "synchronized." Note that clocks can be highly synchronized yet arbitrarily inaccurate in terms of how well they tell true time. For network measurement, often what is most important are *differences* in time as computed by comparing the timestamps from two different clocks. The process of computing the difference removes any error due to clock inaccuracies with respect to true time; but it is crucial that the differences themselves reflect good approximations to differences in true time.

For *resolution*, what we care about is not "relative resolution" but *joint resolution*, which we define as $R_{s,r} \equiv R_s + R_r$. This definition reflects the fact that, when comparing timestamps from C_s with those from C_r, the corresponding uncertainties must be *added* to properly propagate the resulting total uncertainty.

4 Assessing relative clock offset

In this section we discuss how to estimate the relative offset between two network clocks. The closer the offset is to zero, the greater the relative clock accuracy (degree of synchronization).

An important point is that for analyzing network dynamics, estimating relative offset accurately generally is *not* crucial, because the dynamics mostly concern *differences* in transit times rather than absolute transit times. For our purposes, we only need to do estimate relative offsets in order to construct legible plots of the two-way flow of packets, and to qualitatively investigate the relationship between large relative offset and other clock problems such as relative skew. Accordingly, we are satisfied with the method developed in this section even though it is not highly accurate.

4.1 Method for assessing relative offset

We now develop an algorithm for estimating the relative offset $\Delta C_{r,s}$ of a clock C_r at host r with respect to a clock C_s at host s. We assume that both clocks have zero skew with respect to true time. We also assume that the one-way transit time (OTT) across the network from s to r and also from r to s is ΔT (we return to this assumption of symmetry shortly). Note that we do *not* assume that we know ΔT itself.

Suppose a packet is sent from host s at time s_1 (with respect to C_s) and arrives at host r at time r_1, and that a second packet sent in the opposite direction is measured to depart at r_2 and arrive at s_2. Then we have:

$$r_1 = s_1 + \Delta T + \Delta C_{r,s} \quad (1)$$

$$s_2 = r_2 + \Delta T - \Delta C_{r,s} \quad (2)$$

Subtracting the second equation from the first then gives us:

$$\Delta C_{r,s} = \frac{(r_1 - s_1) - (s_2 - r_2)}{2}. \quad (3)$$

That is, from the raw, measured timestamps of the two packets alone we can estimate $\Delta C_{r,s}$, even if we don't know ΔT.

The accuracy of Eqn 3 depends, however, on how closely the OTTs of the two packets fit with the assumption that they are equal. In order to minimize variations in the OTTs due to extraneous network delays such as queueing, we select for our packets those with the *minimal* values (over all of the packets) of $(r_1 - s_1)$ and $(s_2 - r_2)$. Selecting minimal values works well because (most) network-induced noise is *additive* and *positive* (§ 5.2), so minimal values tend to have the least noise.

Even after reducing inequalities due to additional network delays, we still are not on firm ground assuming that we can locate two packets with the same propagation time. Previous studies have found that Internet routes often exhibit significant asymmetries [Pa96], so even in the absence of noise, packets sent in opposite directions along a path may experience considerably different delays. (See also Claffy et al. for discussion of measuring one-way transit times using synchronized clocks [CPB93].) Furthermore, Eqn 3 is necessarily inaccurate in the presence of relative clock skew, since then there is simply no fixed relative offset.

However, since we only try to achieve a rough rectification of the relative offset between the two clocks, we find these remaining inaccuracies acceptable.

4.2 Results of assessing relative offset

Using the methodology developed in § 4.1, we evaluated the relative clock offsets in $\mathcal{N}_1$ and $\mathcal{N}_2$ to see what sort of variation they exhibited. Our goal is to identify groups of closely-synchronized clocks, as we want to determine the degree to which these clocks are less plagued by inaccuracies than less well-synchronized clocks (§ 7). A single computation of $\Delta C_{r,s}$ does not tell anything about the absolute accuracy of either C_r or C_s, but we would expect that many computations of different $\Delta C_{r_i,s_j}$'s will reveal clusterings among the truly accurate clocks, and a large spread among the inaccurate clocks.

Note that in the presence of relative skew, the relative clock offset is not well-defined. However, if we find a pair of clocks that frequently enjoy a low relative offset, then it is plausible that they do not generally suffer significant relative skew, as otherwise their readings would tend to drift apart and they would not be able to preserve their low relative offset.

We proceed by clustering host clocks based on the median of the magnitude of their relative clock offset, over all the transfers in which they participated. We use the median offset in order to isolate hosts that consistently had large relative offsets, instead of those that only occasionally had large offsets, since the latter could be skewed by unfortunately-frequent pairing of a host with an accurate clock together with a host with a poor clock. We use the median of the absolute value of the offset rather than the median of the offset itself as a way of detecting clocks that often "swing" from being too slow to too fast.

We first inspect the median magnitudes of each host's relative clock offset. For both datasets, the same clock emerges as a clear outlier, being typically 5–15 minutes different from the other clock. We next remove the connections involving this outlier and recompute the medians, repeating this process until we converge on a set of clocks that have small median offsets relative to one another. For $\mathcal{N}_1$, this process removes 8 clocks as outliers. After eliminating these clocks, the remainder all have median offsets < 1.25 sec. We

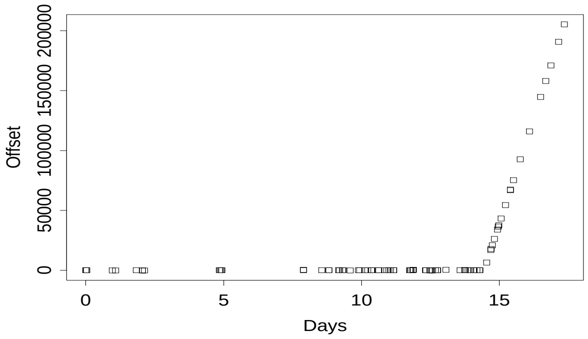

Figure 1: Evolution of `austr`'s relative clock offset over the course of $\mathcal{N}_1$

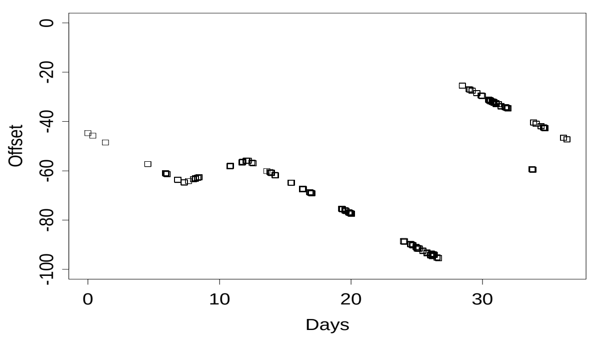

Figure 2: Evolution of `lbli`'s relative clock offset over the course of $\mathcal{N}_2$

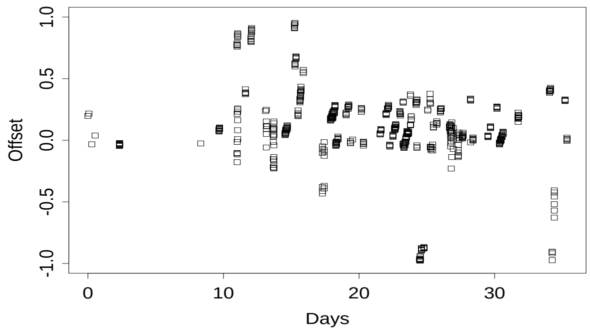

Figure 3: Evolution of `umont`'s relative clock offset over the course of $\mathcal{N}_2$

consider this group of 17 clocks as *closely synchronized*. We can continue the process to find a core group of 5 *highly synchronized* clocks, all with median offsets < 10 msec between one another.

For $\mathcal{N}_2$, removing 7 outliers leaves a group of 24 closely synchronized clocks, all with median offsets below 250 msec. Eliminating six more of these leaves a group of 18 clocks with median offsets below 50 msec. We can further winnow the group down to a final set of 10 highly synchronized hosts, all of which have median offsets between each other of less than 10 msec. This group includes hosts on both coasts of North America as well as two in Europe, indicating synchronization well below that of the propagation time between the hosts–very good, and around the accuracy limit for NTP reported in [Mi92b], even though we are performing a cruder estimate of accuracy (and of relative accuracy rather than absolute accuracy).

We will make use of these different groups of closely-synchronized and highly-synchronized hosts in § 7 when we test whether close synchronization tends to correlate with low relative clock skew.

We finish with a look at how a host's relative offset evolves over the course of an experimental run. The evolution is interesting because it provides a large-scale look at how clock accuracy changes. Our interest here is phenomenological—to develop an appreciation for clock inaccuracies and an awareness of how they occur.

To assess offset evolution, for each host we constructed a plot with the relative offsets (in seconds) computed for those connections for which it served as the data source on the y-axis, versus the time of the connection (days since the beginning of the experiment) on the x-axis. Positive values indicate the host's clock was running behind the receiver's clock, negative that is was running ahead.

Figure 1 shows such a plot for the `austr` tracing host's clock over the course of the $\mathcal{N}_1$ experimental run. Up until the 14th day, it kept good time, but after that point its clock came unglued and ran very slowly, such that the clocks of the other hosts to which it transferred data ran further and further ahead of it (hence, higher and higher offsets). Surprisingly, this is one of the clocks identified above as *highly synchronized*! That assessment, however, was based on *median* relative offset, which filters out the aberrant behavior. We look at this phenomenon further in § 6.6.

Figure 2 shows the evolution of `lbli`'s clock during $\mathcal{N}_2$. While overall the clock has a clear persistent skew, the skew is reversed around day 8, perhaps in an effort to correct the clock's inaccuracy (or perhaps just due to a temperature fluctuation). But the effort ends a few days later and the original skew returns. However, around day 27 the clock's relative offset jumps by over a minute, reflecting a different sort of correction. (This host synchronizes its clock upon reboot.)

Figure 3 presents our last example of interesting clock offset evolution, for another $\mathcal{N}_2$ clock. What is striking here are the presence of offset "towers" that, over the course of hours, slowly elevate the relative offset from nearly zero to several hundred milliseconds. Apparently what is happening is that the clock has a fairly hefty intrinsic skew, but NTP synchronization is detecting this and periodically resetting the clock as it strays too far.

5 Detecting clock adjustments

As shown quite strikingly in Figure 2, computer clocks are sometimes subject to abrupt adjustments in which the clock's notion of the current time is changed, either gradually or instantaneously. Gradual change is produced by artificially altering the clock's skew, so that it slowly shifts its offset towards the target. Instantaneous change is produced by simply loading a new value into the clock register.

Backward clock adjustments, in which a clock is set to a value it already registered in the past, can sometimes be easily detected *if the adjustment is large*, by the presence of a pair of timestamps T_1 and T_2 for which $T_2 < T_1$ even though T_2 was recorded after T_1. In this section we tackle the harder problem of clock adjustments (both forward and backward) that are *not* apparent by trivial inspection of the timestamp sequences.

5.1 Detecting adjustments graphically

Suppose we have a trace pair between s and r. One simple way to detect whether a clock adjustment occurred during the trace is to plot both the OTTs for the packets from s to r and those in the reverse direction. (Packets that are dropped by the network have no OTT associated with them and are omitted from the plot.)

Figure 4 shows such a plot made for a connection from `sdsc` to `usc` in $\mathcal{N}_1$. The solid black squares indicate the OTT for data packets sent from the sender to the receiver, and the hollow squares reflect the OTTs of the acknowledgement packets sent from the receiver to the sender (note that these are significantly smaller than the data packets). The OTTs have been adjusted using Eqn 3 to

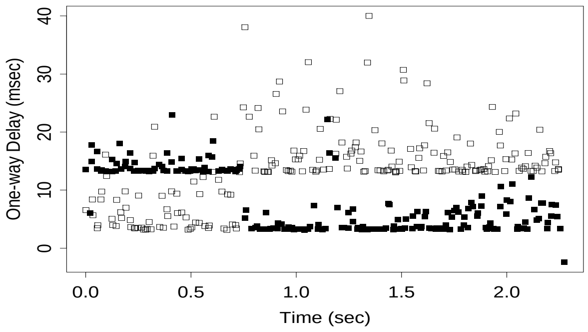

Figure 4: OTT-pair plot illustrating a clock adjustment (sender packets are filled, receiver packets are hollow)

approximately synchronize the two clocks. (In this case, the approximation does not work particularly well, since there is more than one clock offset to estimate!)

The figure shows a striking level-shift occurring for the sender's OTTs around time $T = 0.7$ seconds, a fall of about 10 msec. Furthermore, the OTTs in the opposite direction show an equal and *opposite* change. This equal and opposite change is a crucial aspect of the plot, as it is the signature of a clock adjustment. If the shift were due to a change in network path properties (for example, a route change), then in general we would expect that either (1) it would occur in only one direction, or (2) if it occurred in both directions due to a coupled effect, it would have the same sign.

For a networking change to result in an equal-but-opposite level shift, some resource needs to have been shifted between the two directions of the network path, and furthermore the resource needs to affect the transit times of the small acks equally with those of the large data packets. It is difficult to see what sort of networking change could do this. The change, however, makes perfect sense if, at around time $T = 0.7$ seconds, `sdsc`'s clock was set ahead 10 msec, or `usc`'s clock was set back 10 msec. In either of these cases, the difference in the timestamps for packets sent from `sdsc` to `usc`, i.e., the quantity $r_1 - s_1$ per Eqn 1, will decrease by 10 msec, and similarly $s_2 - r_2$ (per Eqn 2) will *increase* by 10 msec. This is exactly the behavior shown in the plot.

5.2 Removing noise from OTT measurements

Two other points concerning Figure 4 merit attention. The first is the presence of a few unusually small sender packet OTTs, one of about 7 msec around $T = 0$, and the other of around -3 msec around $T = 2.3$. Both of these reflect sender packets that did not carry any data (the SYN and FIN connection management packets). These travel through the network more quickly than full-sized data packets. Hence our techniques need to be careful to not weigh their OTT values the same as those for full-sized packets.

The second important point shown in the plot is the large *variation* in OTTs, both for the full-sized sender packets and the smaller receiver packets. For example, note that the OTTs of both some of the acks before the adjustment, and some the data packets after the adjustment, are larger than many of the OTTs on the other side of the adjustment. This variation is the first suggestion that we will require robust algorithms in order to not be fooled by noise when analyzing OTT data. The eye quite readily picks out the twin level shifts in this plot, but doing so algorithmically requires care to screen out noise such as these large OTT values.

OTTs often exhibit considerable network-induced noise in terms of deviation of a given OTT from the value expected if the network were unloaded. The noise, however, has one crucial property that often makes it tractable: barring a significant change in the network path (such as a route change), the noise always takes the form of an *additive*, *positive* increase. This means that, given a set of OTT measurements, we can often hope to find those with very little network-induced noise by looking at the smallest values in the set.

We used this property of OTT noise in § 4.1 above when we picked *minimal* values of $(r_1 - s_1)$ and $(s_2 - r_2)$ to use when estimating the relative clock offset. We will use it again when developing methods to detect clock adjustments and skew. For these latter, what is interesting are *trends* in how the OTT values (with noise removed) change over the course of the connection. Thus, we cannot simply de-noise the OTT values by selecting the global minimum, or we will obliterate the trend. Instead we divide the series of OTT values up into intervals and de-noise each interval by selecting the minimum value observed during the interval. The question then becomes which intervals to use.

One natural way of devising intervals is to allocate them so that each has the same number of packets. Another is to choose them so that they each span the same amount of time. For assessing trends in OTT values over time, the latter seems to be the natural choice. But using fixed-time intervals has a fundamental problem. Sometimes a connection's activity primarily occurs during only a small portion of the connection's total duration, with the rest of the time mostly inactive due to lengthy retransmission timeout lulls.

To address this difficulty, we combine the two approaches by choosing both a packet-count interval, I_p, and a duration interval, I_t. We then advance through the OTT timings and group timings into a single interval whenever we have either encountered I_p packets, or we have reached a point I_t from the beginning of the interval. At this point, if we have any packets at all, we take their minimum as the de-noised OTT value for the interval, and we begin a new interval by resetting the packet count and setting the start of the interval to coincide with the next OTT measurement.

A final issue is how to pick I_p and I_t. For a set of n OTT measurements spanning an interval ΔT, we used:

$$I_p = \lfloor \sqrt{n} \rfloor, \qquad I_t = \Delta T / \sqrt{n}.$$

Using these choices means that the number of de-noised OTT values scales as the square-root of the total number of values. This struck us as a good compromise between preserving sufficient detail without using too fine a resolution (which could mean we do not effectively remove noise). Furthermore, we anticipate subsequently applying a number of robust algorithms to the de-noised values, some of which have running times of $O(n^2)$ or higher. For these, if we present them with only $O(\sqrt{n})$ values, then the total running time will remain $O(n)$ or only slightly higher, which is important for performing fast automatic analysis.

We will refer to a measured series of OTT values as x_t, and denote the de-noised series derived from x_t as $\check{x}_t$. For each $\check{x}_t$, the index t corresponds to the same index as where in the interval we found the (first) minimal value of x_t. This is an important point—if we instead adjusted the index to reflect, say, the middle of the interval, then we might introduce inaccuracies in the trends. The key idea is that the "best" (least noisy) value of x_t during the interval occurred at a particular t, and we want to take that point and discard all the others in the interval.

Figure 5 shows the results of applying this de-noising method to the measurements plotted in Figure 4.

5.3 An algorithm for detecting adjustments

We now turn to attempting to detect adjustments algorithmically (though we will be forced to also introduce heuristics, for reasons discussed below). The central notion we will use is that of the *signature* of the OTTs in the two directions showing equal but opposite level shifts.

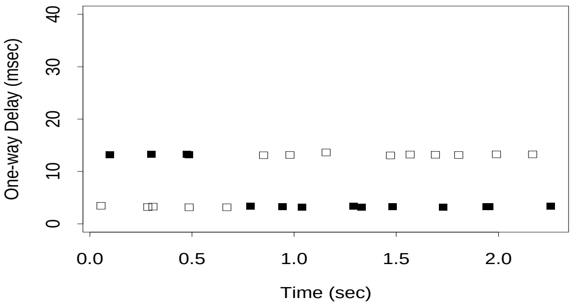

Figure 5: Same measurements after de-noising pair-plot

Identifying pivots. The foundation of our approach lies in identifying *pivots*: points in time before which the OTTs all lie predominantly above or below all the OTTs after the given point in time. In Figure 4, the pivot we aim to identify occurs around $T = 0.7$ sec. We now develop a heuristic for identifying pivots in the series of OTTs for packets sent in a single direction (from s to r or vice versa). We then will analyze the pivots identified in both directions to test for a clock adjustment.

Let $\check{x}_t$ be a series of de-noised OTT values occurring at times t, ordered by the time index t. Let $\check{x}_{t_i}$ be the same series numbered from $i = 1 \ldots n$, where t_i is the ith measurement time. We define a *pivot partition* of $\check{x}_t$ as a partition of $\check{x}_t$ into two disjoint sets, $\check{x}'_t$ and $\check{x}''_t$, for which the maximum of one set is less than the minimum of the other. Without loss of generality, let $\check{x}'_t$ be the "larger" of the two sets, i.e., its minimum is larger than the maximum of $\check{x}''_t$.

We further require that the time intervals spanned by $\check{x}'_t$ and $\check{x}''_t$ are disjoint, namely either the largest i in $\check{x}'_{t_i}$ is less than the smallest j in $\check{x}''_{t_j}$, or vice versa.

We term the pivot partition *positive* if the measurements $\check{x}'_t$ occurred *after* those in $\check{x}''_t$, and *negative* otherwise.

Geometrically, this definition corresponds to being able to draw horizontal and vertical lines on a plot like that in Figure 5 such that either all of the points lie in the first and third quadrants formed by the lines (if positive), or in the second and fourth quadrants (negative).

It is important to note that a given series $\check{x}_t$ may have more than one pivot partition. For example, if $\check{x}_t$ is strictly decreasing, then every value of t gives rise to a pivot partition. In addition, any time the largest or smallest value of $\check{x}_t$ occurs at the lowest value of t, i.e., $\check{x}_{t_1}$, then there is a pivot partition that isolates that one value versus placing all the other values in the other partition set. Generally, this is not a pivot partition of interest.

We proceed as follows. First, we determine whether to search for a positive or negative pivot by inspecting whether $\check{x}_{t_1}$ is less than or greater than $\check{x}_{t_n}$. From here on, we assume without loss of generality that we wish to detect a positive pivot, such as the one exhibited by the receiver packets (hollow squares) in Figure 4.

We search through the measurements to find the point k where $\min(\check{x}_{t_{k+1}}, \check{x}_{t_{k+2}}) - \max(\check{x}_{t_{k-1}}, \check{x}_{t_k})$ is largest. Conceptually, we are looking for the intervals that have the greatest difference between them in the same direction as the pivot; we spread the differencing over the additional intervals on either side to combat the problem of the intervals right at the pivot misleading us due to noise. Note that this spreading operation also means that we cannot detect a pivot that occurs right at the beginning or end of a connection (§ 5.6).

k is now the candidate pivot (actually, the potential pivot occurs at a point in time between measurement k and measurement $k+1$). We then inspect the points $\leq k$ to find χ_k, the largest point before the candidate pivot, and likewise those $> k$ to find χ_{k+1}, the smallest after the candidate. If χ_k is less than χ_{k+1}, then we conclude that $[k, k+1]$ does indeed straddle a pivot; otherwise, we conclude they do not.

If we find a pivot partition, then we define its magnitude M as the absolute value of the difference between the median of the points after the pivot with the median of those before. We also associate a pivot width, $W = t_{k+1} - t_k$.

Identifying adjustment signatures. We now turn to identifying the signature of a clock adjustment for the clocks of two hosts, s and r. The method we developed is not entirely satisfying, as it uses some heuristics in order to accommodate residual noise in the OTT measurements, while attempting to not mistake genuine networking effects for a clock adjustment. However, the method appears to work well in practice (see § 5.4). We note, though, that the method assumes that clock adjustments are relatively rare events: rare enough that our traces are likely to exhibit at most one adjustment, and that the likelihood of *both* of the clocks we are comparing exhibiting an adjustment during the trace is negligible. This also appears to generally hold (again, see § 5.4).

Suppose we have two sets of de-noised OTT measurements, $\check{s}_t$ and $\check{r}_t$. If either of $\check{s}_t$ or $\check{r}_t$ does *not* exhibit a pivot, or if the pivots are both positive or negative, then we conclude there was not any clock adjustment. Otherwise, let M_s, W_s, M_r, and W_r be the magnitudes and widths of the corresponding pivots. We next check whether the pivots *overlap*. Let s_1 and s_2 denote the packets bracketing $\check{s}_t$'s pivot region, and likewise for r_1 and r_2. Let s_1^s denote the time at which s_1 was sent from s (according to s's clock), and s_1^r the time at which it arrived at r (according to r's clock). With analogous definitions for the other packets, we then conclude that the pivots overlap if either of the following holds:

$$s_1^r < r_2^r + \delta t \quad \text{and} \quad s_2^r + \delta t > r_1^r,$$

$$\text{or} \quad r_1^s < s_2^s + \delta t \quad \text{and} \quad r_2^s + \delta t > s_1^s,$$

where δt is the allowed "slop," which we set to:

$$\delta t = \frac{\max(W_s, W_r)}{2}. \tag{4}$$

The idea behind the slop is to help detect other-than-instantaneous adjustments (illustrated below).

If the pivots do not overlap, then we conclude there was no adjustment. If they do, we then next look at the magnitudes of the pivots. If either magnitude is less than the larger of twice the joint clock resolution $R_{s,r}$ (§ 3), or 2 msec (an arbitrary value to weed out fairly insignificant adjustments), then we declare the pivot "insignificant" and ignore it.

Finally, we check whether M_s and M_r are within a factor of two of each other. If not, then we term the pivot a "disparity pivot," meaning that it may be due to unusual networking dynamics (§ 5.6). If the two agree within a factor of two (which experience has shown is a good cut-off point), then we conclude that the trace pair exhibits a clock adjustment with a magnitude of about $\frac{M_s + M_r}{2}$.

5.4 Checking the algorithm's accuracy

We now turn to the important question of *How do we know the algorithm actually works?* Since we are restricted to post-facto analysis, we need to develop other means for detecting likely clock adjustments, and use them to gauge the algorithm's accuracy.

We can divide our accuracy concerns into two types: *false positives*, in which the algorithm claims a clock adjustment occurred when in fact one did not, and *false negatives*, in which it fails to detect that an adjustment actually did occur.

Since the algorithm only flags adjustments in a relatively small number of traces (§ 5.5), we can deal with the possiblity of false positives by manually inspecting each of these using a plot like in

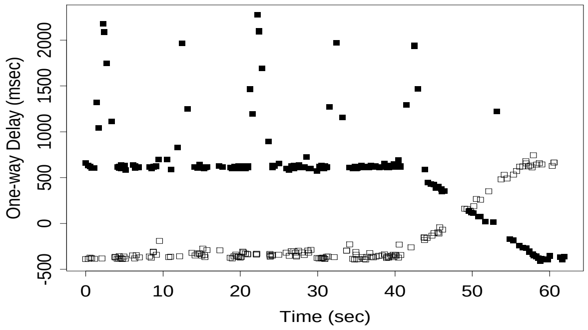

Figure 6: Clock adjustment via temporary skew

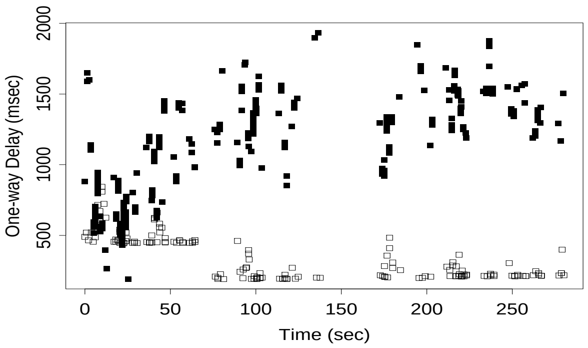

Figure 7: Likely clock adjustment masked by network delays

Figure 4 to determine whether we find compelling evidence that an adjustment really did occur. The process of doing so led to some of the finer points of the algorithm, such as rejecting "disparity pivots." After these additions, we find virtually no apparent false positives (though who knows how many we are missing because their presence is not visually compelling).

The possiblity of false negatives is more difficult to address. Since we have too many traces to inspect by hand (though we did apply random sampling to hand-inspect a large number of traces), we developed two other heuristics for identifying clock adjustments. The first is to compute the minimum round-trip time (RTT) that could be derived from differences between the timestamps for any pair of packets between the two hosts. If this was significantly lower than the minimum observed round-trip time (using a single clock), and especially if it was ever non-positive, then `tcpanaly` flags the trace as requiring manual inspection. The second is to compute the cross-correlation between the denoised OTT times in the two directions, and then to flag traces with strong negative correlations. The use of these heuristics also led to refinements in the detection algorithm, such as spreading out the pivot differencing over multiple intervals when searching for candidate pivots, and allowing "slop" per Eqn 4. After these additions, we find very few false negatives (see § 5.6 for examples).

5.5 Results of checking for adjustments

`tcpanaly` uses the method given in § 5.3 to check each trace pair it analyzes for clock adjustments. Doing so, we found 36 trace pairs in $\mathcal{N}_1$ out of 2,335 (1.5%) that exhibited apparent clock adjustments, and 128 out of 15,492 in $\mathcal{N}_2$ (0.8%). While these proportions are fairly low, they are high enough to argue that a large-scale measurement study for which accurate timestamps are important needs to take into account the possibility of clock adjustments. Furthermore, *the adjustments are only detectable due to the use of a pair of clocks*. If a study uses timestamps from only one measurement endpoint, then checking the timestamps for clock adjustments becomes much more difficult.

The median adjustments were on the order of 10–20 msec, the mean around 100 msec, and the maxima close to 1 sec. These magnitudes are unfortunately small enough to sometimes not be glaringly obvious, but large enough to be comparable to wide-area packet transit times, so they can introduce quite large analysis errors if undetected.

While clock adjustments are usually abrupt, this is not always the case. The adjustment-detection method found some clock adjustments that occurred due to a short period of altered clock frequency (i.e., temporary skew). Figure 6 shows a striking example. Here, around time $T = 40$ sec the sender's clock began running more quickly than the receiver's, leading to lower sender OTTs and higher receiver OTTs. Less than 20 seconds later, the frequencies were again equal, but the relative offsets between the clocks shifted by nearly 1 sec in the process.

5.6 Problems with detection method

The method given in § 5.3 appears to work well in practice, at least in terms of the checking discussed above. However, it does sometimes fail to detect clock adjustments. In this section we look at some cases where we identified this happening.

Failure to detect adjustment via skew. In Figure 6 we illustrated how sometimes a clock adjustment can occur due to temporary skew. However, in such cases there are multiple pivots in each direction (any location along the skew line is a pivot), and sometimes, due to noise, the two pivots located by the method do not overlap, and the possibility of an adjustment is rejected. In general, this sort of failure will only occur with adjustments using temporary skew; abrupt adjustments have sharply defined pivots. (This example was detected due to a non-positive minimum RTT, as discussed in § 5.4.)

Excessive network-induced delay. Figure 7 shows a case where the reverse path exhibits a clear level shift around $T = 70$ sec, with a magnitude of about 250 msec, but the corresponding shift on the forward path is less clear because it is accompanied by an increase in networking delays, too. In that direction, `tcpanaly` assesses the magnitude of the shift as about 730 msec. Since this is more than twice the magnitude in the other direction, `tcpanaly` rejects the possibility of a clock adjustment.

`tcpanaly` flags a trace pair like this as having a "disparity pivot," namely common pivots that have too great a difference in their magnitudes to be considered a clock adjustment. Disparity pivots are quite rare (only 61 in $\mathcal{N}_2$). We inspected each one and found that only the one shown above was a plausible clock adjustment. The rest appear simply due to unfortuitous patterns of noise.

Adjustment too close to connection edge. Since our method for identifying pivots (§ 5.3) will not accept a pivot right at the beginning or at the end of a connection, `tcpanaly` naturally will miss this sort of adjustment should it occur.

Multiple adjustments. The development of the clock adjustment detection algorithm presumes that there is a single clock adjustment to be detected. Sometimes a trace pair suffers from more than one adjustment, and the algorithm either only detects one of them, or fails to detect any of them. The latter is particularly likely if there are two adjustments in opposite directions. Figure 8 shows a striking example of a trace pair with two adjustments, both effected using temporary skew. (This example was likewise detected due to a non-positive minimum RTT; the strong negative correlation test also detects it.)

Clock "hiccups." Related to the multiple adjustments discussed above are clock "hiccups," in which one of the clocks in a trace pair momentarily either ceases to advance or advances very

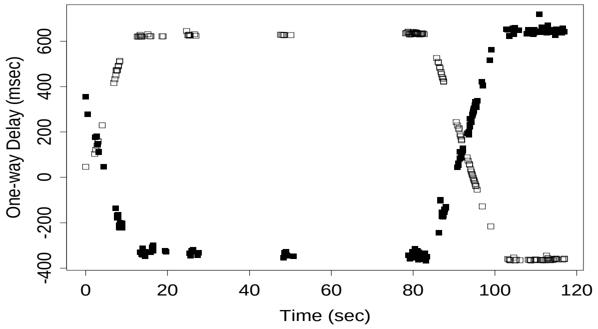

Figure 8: Double clock adjustment via temporary skew

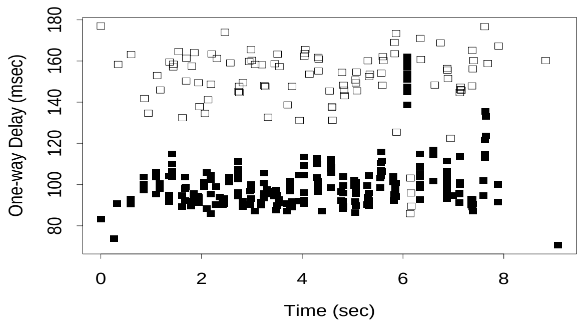

Figure 9: Clock adjustment "hiccup"

quickly. Figure 9 shows an example, occurring at time $T = 6$ sec. It is possible that this example is actually due to surprising network dynamics, as the 4 acks with lowered OTTs come right after the only packet reordering event in the trace. (While a clock glitch can change the value of OTTs, it *cannot* reorder packets on the wire! But see [Pa97b] for measurement errors that can indeed reorder packets.) It is difficult to see what networking mechanism could lead to the data packets in the opposite direction simultaneously experiencing increased delay.

6 Assessing relative clock skew

Errors in relative clock skew, which often introduce inaccuracies on the order of perhaps a few seconds a day, might seem trivial and perhaps not worth the effort of characterizing. For purposes of keeping fairly good absolute time, this is true, but for purposes of assessing network dynamics, it is not.

To illustrate why skew is a crucial concern, consider evaluating OTTs between two hosts s and r, for which r's clock runs 0.01% faster than s's. That is, over the course of a day, r's clock will gain about 9 seconds relative to s's clock, not a particularly large error for many purposes. If, however, we are computing OTTs between s and r, then over the course of only 10 minutes r's clock will gain 60 msec over s's clock. *If we assume that variations in OTT reflect queueing delays in the network, then this minor clock drift could lead to a large false interpretation of growing congestion.* For example, if s sends 512 byte packets to r and the bandwidth of the path between them is T1 (1.544 Mbps), then a true 60 msec increase in delay reflects the equivalent of an additional 23 packets' worth of queueing. Thus, quite "minor" skew differences between the two endpoint clocks can lead to quite large, erroneous assessments of queueing delay.

The first issue for detecting skew is to identify a skew "signature" similar to that for clock adjustments shown in Figure 4.

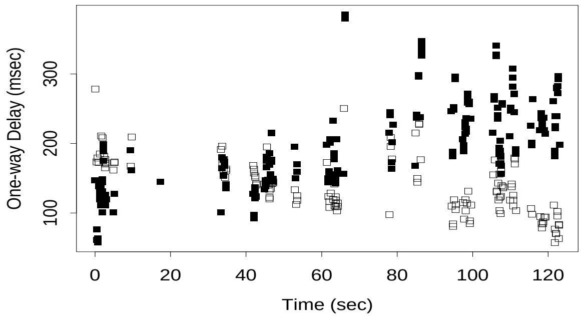

Figure 10: An OTT pair plot showing relative clock skew

Figure 10 shows an OTT pair plot that exhibits a clear skew signature: the OTTs in one direction show a steady overall increase, while those in the opposite direction show a steady decrease. Both changes have a magnitude of about 120 msec over the 2 minute course of the connection, consistent with the receiver's clock advancing about 0.1% faster than the sender's clock. It is difficult to see what sort of network dynamics could introduce such a true combined inflation and deflation of OTTs over a two-minute period, so we conclude that the OTT pair plot shows strong evidence of relative clock skew.

We now turn to developing robust algorithms for detecting and removing relative clock skew.

6.1 Defining canonical sender/receiver skew

We begin by defining exactly what quantity it is that we wish to estimate. First, we assume that the skew trends we identify will be *linear*. While we might possibly encounter non-linear skew, we did not find any clear examples of such in $\mathcal{N}_1$ or $\mathcal{N}_2$. For linear skew, we can summarize the skew using a single value that reflects the excess rate at which one clock advances compared to the other.

To avoid ambiguity (in terms of which clock we are comparing to which), we will always quantify how C_r, the receiver's clock, advances with respect to C_s. Suppose C_r runs a factor η faster than C_s, by which we mean that, if C_s reports that an interval ΔT has elapsed, then C_r will have reported the same interval as having length $\eta\Delta T$.

The algorithms we develop are based on how OTT measurements expand or shrink with respect to time. It is important to recognize that the phrase "with respect to time" does *not* mean "with respect to true time," since we have no way of measuring true time. Instead, it means "with respect to the clock at the packet originator."

When discussing a linear trend in the measured OTTs of the packets sent by host s, we will quantify the trend in terms of G_s, the growth in the OTTs of the packets sent by s. Suppose packet p_1 is sent at time T_s^1, according to C_s, and arrives at time T_r^1, according to C_r. Likewise, suppose packet p_2 is sent at T_s^2 and arrives at T_r^2. Suppose further that the transit times of the packets are identical (no network-induced noise), so the only variations in their OTTs are due to clock skew.

The measured OTTs for the two packets are:

$$\phi_1 = T_r^1 - T_s^1, \qquad \phi_2 = T_r^2 - T_s^2.$$

As G_s quantifies the linear growth in measured OTTs over time, we have:

$$\phi_2 = \phi_1 + G_s(T_s^2 - T_s^1).$$

In the absence of relative skew between C_r and C_s, $G_s = G_r = 0.0$, where G_r quantifies the growth in OTTs of packets sent by r. If C_r

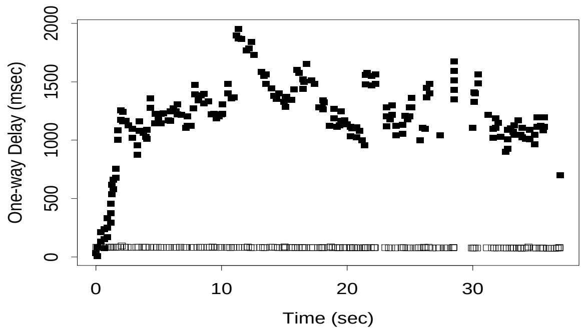

Figure 11: Clock skew obscured by network delays

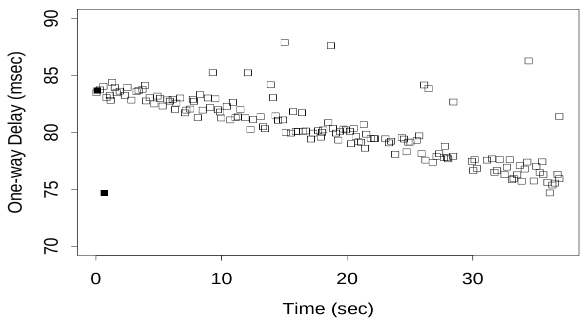

Figure 12: Enlargement of reverse path

runs faster than C_s, then the packets sent by s will exhibit *increasing* OTTs and those sent by r will exhibit *decreasing* OTTs, so we will have $G_s > 0$ and $G_r < 0$. Naturally, the reverse holds if C_r runs slower than C_s.

It can be shown that:

$$G_s = \eta - 1 \quad (5)$$

$$G_r = \frac{1}{\eta} - 1 \quad (6)$$

$$= \frac{1}{G_s + 1} - 1. \quad (7)$$

For $\eta = 1 + \epsilon$, where $|\epsilon| \ll 1$, we have:

$$G_s = \epsilon, \qquad G_r = -\frac{\epsilon}{1+\epsilon} \approx -\epsilon.$$

Because clock skews are often only a few parts per thousand or ten thousand, we are usually in this regime (but see § 6.6 below). Consequently, an easy inaccuracy to introduce is to assume that:

$$G_s = -G_r,$$

(i.e., the slopes are equal but opposite), since this often appears to be the case when inspecting OTT pair plots. To ensure full accuracy, we instead take care to always use Eqns 5 and 6 to express relative clock skew in terms of η, or Eqn 7 to translate G_r to G_s. We will refer to values of G_s and G_r that are consistent with respect to Eqn 7 as "equivalent but opposite trends."

6.2 Difficulties with noise

One particular problem with testing for clock skew is that, due to queueing fluctuations, one direction of a path can have such highly variable OTTs that these completely mask the smaller-scale trend of OTT increase or decrease due to skew, even after de-noising. Figure 11 shows an example, in which congestion on the forward path completely obscures the relative clock skew, which is apparent from the enlargement of the return path shown in Figure 12. Such noise most often obscures the forward path (presumably due to extra queueing induced by the data packets), but it can also obscure the reverse path. Thus, we cannot always rely on the signature of *dual* equivalent-but-opposite OTT trends; sometimes we must settle instead for simply a compelling trend in one direction.

Furthermore, network-induced noise also scuttles what might seem the most straightforward approach to detecting skew, namely fitting a line to the de-noised OTT measurements, $\check{s}_t$ and $\check{r}_t$ (§ 5.2). Even using de-noised measurements, least-squares fitting fails to provide solid skew detection, because residual noise in $\check{s}_t$ and $\check{r}_t$ makes it too difficult to reliably distinguish between a skewing trend and coincidental opposite queueing trends. All it takes is one period of elevated queueing at either end of a connection to throw off the fit.

Unfortunately, the same also occurs using robust fitting techniques, such as estimating the line's slope as the median of all of the pairwise slopes between the individual de-noised measurements [HMT83]. The difficulty lies in both false positives and false negatives generated due to queueing fluctuations. Clearly, we need an even more robust technique.

6.3 A test based on cumulative minima

Eventually we recognized that the most salient feature of relative clock skew is not simply the overall trend (slope) of the OTT measurements, but the fact that the smallest such measurements continually increase or decrease. This observation suggests the following statistical test, the strength of which is that it is nearly immune to transient increases in OTT measurements due to queueing buildups.

Suppose we have n observations X_{t_i}, $1 \le i \le n$, where t_i is the time of the observation and X_{t_i} is the value of the observation. We assume that the t_i's are monotone increasing, and that the X_{t_i} are distinct. Further, we assume without loss of generality that we wish to test for a negative trend in X_{t_i}. We discuss applying the same test for a positive trend in § 6.4 below.

Consider the indicator:

$$I_{t_j} = \begin{cases} 1, & \text{if } X_{t_j} < \min_{i<j} X_{t_i}, \text{ or if } j = 1, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

That is, I_{t_j} is 1 if X_{t_j} represents a new "cumulative minimum" if we inspect X_{t_i} from 1 up to j (but not all the way up to n), and 0 if there is an earlier X_{t_i} that is less than X_{t_j}.

If the X_{t_i} are independent, then:

$$P[I_{t_j} = 1] = 1/j,$$

because the probability that any particular X_{t_i} out of j observations is the minimum of the group is simply $1/j$.

Consider now the function:

$$M_j = \sum_{i=1}^{j} I_{t_i},$$

which is the number of cumulative minima seen as we inspect X_{t_i} from the first value up to the jth value. The key observation we make is that, in the absence of a negative trend, the distribution of M_j will tend to be close to that for independent X_{t_i}; that is, we will find a few cumulative minima but not a great number; while, in the presence of a negative trend, we should find many cumulative minima, since the X_{t_i} tend to get smaller and smaller.

Suppose we find $M_n = k$, that is, the X_{t_i} exhibit k cumulative minima. We wish to compute the probability that we would have observed this many or more minima, given the independence assumption. If we find the probability sufficiently low, we will reject

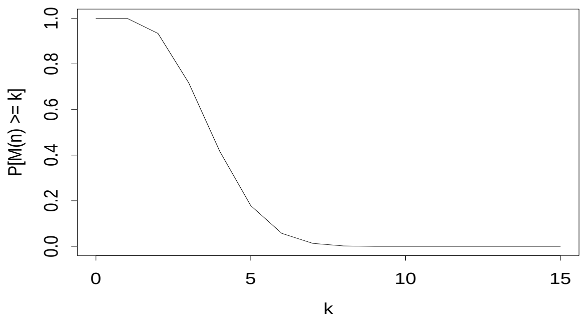

Figure 13: Distribution of $R(n, k)$ for $n = 15$

the null hypothesis that the X_{t_i} are independent. In its place we will accept the tentative hypothesis (which we will further test in § 6.5) that the X_{t_i} exhibit a negative trend.

Let $R(n, k) = P[M_n \geq k]$. Given $0 \leq k \leq n$, we can compute $R(n, k)$ recursively, as follows:

$$R(n,k) = \begin{cases} 1, & \text{if } k = 0, \\ 1/n!, & \text{if } k = n, \text{ and} \\ \frac{R(n-1,k-1)+(n-1)R(n-1,k)}{n} & \text{if } k < n. \end{cases} \quad (8)$$

The first case is the degenerate one that grounds the recursive definition: the probability that there are at least 0 cumulative minima is always 1. The second case corresponds to every single X_{t_i} being a cumulative minimum. This only occurs if the X_{t_i}'s are sorted in descending order, which, if they are independent, has probability $1/n!$.

The last case corresponds to conditioning on whether X_{t_n} is a cumulative minimum or not. For independent X_{t_i}, it will be a cumulative minimum with probability $1/n$, and not with probability $(n-1)/n$.

Figure 13 shows the distribution of $R(n, k)$ for $n = 15$. The key feature of the distribution that makes it a powerful test for a negative trend is the rapid fall-off in probability above a certain point, in this case around $k = 8$. Because if the X_{t_i}'s do indeed have a negative trend we should find k quite close to n, this means we can readily distinguish between the case of a negative trend and that of no trend, without requiring that *all* of the X_{t_i} be increasingly negative. Thus, we can accommodate considerable noise.

6.4 Applying the test to a positive trend

The test developed in § 6.3 for detecting a negative trend can also be applied to detecting a positive trend, with one subtlety. At first blush one might think that, to do so, one simply uses maxima in lieu of minima. This works in principle, but fails when applied to OTT sequences, because of the positive additive nature of OTT noise (§ 5.2). That is, the maxima will be often dominated by the noisiest OTT values, rather than by OTT values that slowly rise due to skew, so the noise will obscure any positive trend due to clock skew. This remains a problem even after de-noising, since all it takes is a single period of elevated OTT values, long enough to span an entire de-noising interval, to pollute the de-noised values with what will in some cases be a global maximum. When searching for a negative trend, such an interval will, on the other hand, simply not include a cumulative minimum; but it will not prevent the test from finding other minima due to clock skew.

There is a simple fix for this problem, though: we apply the cumulative minima test to $Y_{t_j} = X_{t_{n-j+1}}$, which is simply X_{t_i} viewed in reverse. The reversal converts a positive trend in X_{t_i} to a negative trend in Y_{t_j}, which the cumulative minima algorithm then readily detects.

6.5 Identifying skew trends

With the cumulative minima test we finally have a robust algorithm for detecting trends. These trends, however, might not be due to clock skew but to networking effects, so we need to develop further *heuristic* checks to correctly detect linear skew.

Suppose we have two sequences of de-noised OTT measurements, $\check{s}_t$ and $\check{r}_t$, corresponding as usual to the full-sized data packets sent from the connection sender to the receiver, and the acks sent back from the receiver to the data sender. For each sequence, we first determine whether it is a *skew candidate* as follows.

Let u_t denote the given sequence. Let $R_u(n, k)$ be the probability that the sequence u_t matches the null hypothesis of no trend (independence) given by Eqn 8. We consider u_t a skew candidate if either:

1. $R_u(n, k) < 10^{-6}$ and u_t is either $\check{r}_t$, or u_t is $\check{s}_t$ and its trend is negative. This latter test is because queueing buildup due to the data packets sent along the forward path can often produce a strong positive trend; or

2. $R_u(n, k) < 10^{-3}$ and u_t is *tightly clustered* around the "trend line," which is computed using a robust linear fit (per the algorithm discussed above) to just the (denoised) timings corresponding to the cumulative minima or maxima.

   The goal here is to allow for a skew candidate if the u_t points fit quite closely to a (linear) trend, even though their cumulative minima probability is not so small. This can happen, for example, if we do not have a large number of points in u_t.

   Note that the limit of 10^{-3} precludes assuming a skew candidate if there are fewer than 7 points, since $1/6! \approx 1.4 \cdot 10^{-3}$ (but see below).

   It remains to define "tightly clustered." To do so, we compute the inter-quartile range (75th percentile minus 25th percentile) of the distance between the u_t and the trend line. If it is less than or equal to the larger of the joint clock resolution, $R_{s,r}$, or 1 msec, then a large number of the de-noised OTTs lie very closely to a pure linear trend.

We next determine whether either $\check{s}_t$ or $\check{r}_t$ is compelling enough by itself to accept as evidence of a skew trend; or if the pair form a *joint* skew candidate, to be investigated further; or if there is insufficient evidence for a skew trend. To do so, we first consider which of them is individually a skew candidate, as follows:

1. If neither is a candidate, then we check to see whether $\max(R_s(n, k), R_r(n, k)) \leq 10^{-2}$. If so, then the joint probability that both have no trend (or, more precisely, are fully independent) is $\leq 10^{-4}$, which we consider sufficiently low to consider them as joint skew candidates and proceed as discussed below. If either probability exceeds 10^{-2}, then we reject the trace pair as a candidate for exhibiting a skew trend.

2. If $\check{r}_t$ is a skew candidate but $\check{s}_t$ is not, then we accept $\check{r}_t$ as reflecting clock skew quantified using the corresponding G_r. We do so because sometimes we have no hope of detecting a skew trend in $\check{s}_t$ due to queueing buildup, as illustrated in Figure 11 and Figure 12.

3. If $\check{s}_t$ is a skew candidate but $\check{r}_t$ is not, then we check the direction of $\check{s}_t$'s trend. If it is negative, then this goes against the networking tendency for a positive trend induced by the queueing of the data packets along the forward path, and we accept $\check{s}_t$ as reflecting clock skew quantified using G_s.

   If the trend is positive, we must proceed carefully to screen out a false skew trend due to queueing. First, we require

$$\sigma^2_{\check{s}_t} \leq \sigma^2_{\check{r}_t},$$

that is, the variance of the de-noised OTT values along the forward path is less than that in the reverse path. If this is not the case, then we reject the trace pair as a candidate for exhibiting a skew trend.

Next we split $\check{s}_t$ into two halves, $\check{s}_{t_1}$ and $\check{s}_{t_2}$, with the division coming at $\lfloor \frac{n}{2} \rfloor$ if s_t has n values. If $R(n, k)$ for either half exceeds 10^{-2}, or if the trends for the two halves do not agree in direction, then we also reject the possibility of a skew trend.

If $\check{s}_t$ passes these tests, then we consider $\check{s}_{t_1}$ and $\check{s}_{t_2}$ as comprising a joint skew candidate. We reverse $\check{s}_{t_2}$ so it now has the opposite trend of $\check{s}_{t_1}$, and proceed as discussed below.

4. If both $\check{s}_t$ and $\check{r}_t$ are skew candidates, then we consider them together a joint skew candidate.

If the above procedure yields a joint skew candidate, we then evaluate the candidate as follows:

1. If both candidates have the same trend direction, then we reject the possibility of a skew trend.
2. If not, then we translate the first candidate's skew quantification into terms of the second candidate using Eqn 7. Let G_1 and G_2 be the corresponding skew quantifications (one of which has been translated, so they can be directly compared). If
$$|G_1 - G_2| > \frac{G_1 + G_2}{2},$$
that is, the difference between the two exceeds their average, then we reject the pair as having too much variation in their slopes for them to be trustworthy indicators of skew. Otherwise, we accept the pair as indicative of a skew quantified as $G = \frac{G_1+G_2}{2}$.

6.6 Results of checking for skew

`tcpanaly` uses the method given in § 6.5 to check each trace pair it analyzes for clock skew. As we did for detecting clock adjustments, we gauged its accuracy by visually inspecting many of the skews it found (to detect false positives), and also (for false negatives) by hand-inspecting randomly chosen traces, as well as those with strong, negative cross-correlations in their OTTs or excessively low minimum RTTs (per § 5.4). These last, as for clock adjustments, often occur in the presence of significant clock skew. Making these checks led to a number of the heuristics outlined above, and we now find the algorithm appears reliable, at least in terms of plausible skew trends we can detect visually.

The method indicates that 295 trace pairs in $\mathcal{N}_1$ out of 2,335 (13%) exhibited clock skews, and 487 out of 15,492 did so in $\mathcal{N}_2$ (3%). These proportions are high enough to argue for considerable caution when comparing timestamps from two different clocks.

In both $\mathcal{N}_1$ and $\mathcal{N}_2$, about three-quarters of the skews were detected on the basis of $\check{r}_t$ alone, not particularly surprising since often a skew trend in $\check{s}_t$ will be lost in the OTT variations due to queueing induced by the data packets. (We could avoid this problem if we could choose the particulars of our measurement traffic, rather than analyzing TCP bulk transfer traffic.) The largest skew in $\mathcal{N}_1$ was a whopping $\eta = 5.5$, meaning that one clock ran *more than five times faster than the other*! Figure 14 shows how skew like this appears in an OTT pair plot. (Note that the reverse path starts a time $T = -4$ sec because `tcpanaly` could not figure out any sort of useful relative clock offset.) In the forward direction, the connection's elapsed time was only 2 sec, but in the reverse direction it took 10 sec!

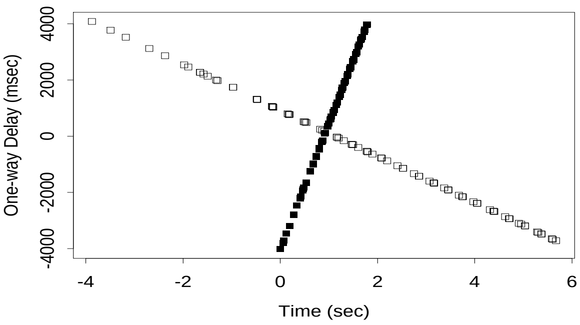

Figure 14: Example of extreme clock skew

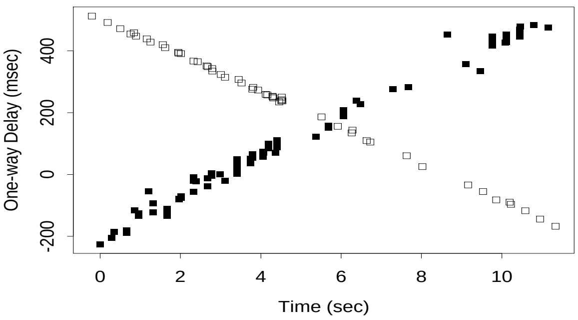

Figure 15: Strong relative clock skew of 6%

This example is more than just an amusing curiosity. It occurred not once but 43 times in $\mathcal{N}_1$ (see Figure 1). We note, however, that this clock (which corresponds to the `austr` site) was one of the ones identified in § 4.2 as being *highly* synchronized with a number of the other sites, indicating care was being taken to keep accurate time with it (presumably using NTP). Thus, this clock's behavior is a compelling argument that *just because a clock is believed to be well-synchronized does not render it immune from extreme error!*

Aside from `austr`'s clock, the next largest skew we observed in $\mathcal{N}_1$ was $\eta = 0.991$, a frequency difference of about 0.9%. This led to an OTT change of about 70 msec during an 8 sec connection. All in all, after removing connections involving `austr`, in $\mathcal{N}_1$ the median skew had a magnitude of about 0.023%, and the mean 0.035%. These are small, but not negligible.

In $\mathcal{N}_2$, the prevalence of trace pairs exhibiting skew was significantly lower (3% versus 13%), perhaps due to the use among the participating sites of newer hardware with more reliable clocks. After removing one site that either had a very broken clock or very unusual network dynamics (we were unable to determine which; perhaps it was both), the largest skews we observed were on the order of 6%. Figure 15 shows an example. The pattern is quite striking, and clearly could lead to grossly inaccurate conclusions about network dynamics if undetected. Note that both sites involved in this connection were among those identified as closely synchronized in $\mathcal{N}_2$ (§ 4.2), again emphasizing that clocks that are *in general* well-synchronized can still exhibit very large errors.

After removing these connections, the median skew magnitude of the remainder in $\mathcal{N}_2$ is about 0.011%, and the mean around 0.016%. These are a factor of two smaller than those in $\mathcal{N}_1$, but still not completely negligible for assessing queueing in longer-lived connections.

Dataset	Relative offset	Likelihood of adjustment
$\mathcal{N}_1$	< 1 sec	1.4 %
$\mathcal{N}_1$	≥ 1 sec	1.6 %
$\mathcal{N}_2$	< 1 sec	0.75 %
$\mathcal{N}_2$	≥ 1 sec	0.95 %

Table 1: Relationship between relative clock accuracy and clock adjustments

6.7 Removing relative skew

As discussed in the previous section, a non-negligible proportion of the trace pairs in our study suffer from relative clock skew. We would like to remove this skew so we can then reliably include those traces in subsequent analysis of network dynamics. Fortunately, the skew almost always appears well-described as linear, which means it is straight-forward to remove it.

To remove skew of magnitude η, we simply modify all the timestamps t_i^r generated by C_r using:

$$t_i^{r\prime} = t_i^r + G_r(t_i^r - t_0^r), \tag{9}$$

where G_r is given by Eqn 6 and t_0^r is the first timestamp generated by C_r.

A key point is that applying Eqn 9 does *not* necessarily rectify C_r's skew with respect to *true time*. It only rectifies it with respect to C_s. It could be that the correct action to take in terms of true skew removal is to apply an analogous transformation to C_s's timestamps *instead*. We have no way of knowing which clock is in error, but by Eqn 9 we can still make the two sets of timestamps consistent, and eliminate artificial trends in the network delays we compute, even if some absolute skew remains.

After `tcpanaly` removes relative skew, it re-analyzes the clock. If it still detects relative skew, then either its initial assessment that the trace pair had relative skew was wrong, or the skew was not linear. It flags this case separately, and also then refrains from any further timing analysis. Thus, re-analysis provides a self-consistency test for the soundness of our skew detection. This test failed less than 2% of the time.

7 Clock synchronization vs. stability

We finish our study with an investigation into the question of whether highly-synchronized clocks tend to be free of problems such as adjustments and skew. We will term clocks free of such problems as "stable."

We might hope that highly-synchronized clocks would also be stable, because freedom from such problems would tend to greatly aid a clock in maintaining synchronization. On the other hand, if good synchronization is maintained by frequently adjusting an errant clock to match an external notion of accurate time, then such clocks might be *more* likely to exhibit adjustments or skew, and hence be less stable than other clocks.

The issue is an important one because it is quite cheap to determine whether a remote clock's offset is close to that of a local clock (§ 4.1). If relative accuracy is a good indicator that the remote clock is stable, then we can quickly determine that we can rely on the soundness of the timestamps generated by the remote clock, without having to go through all the effort of the methods developed in this paper for detecting adjustments and skew. Such a quick determination could prove invaluable for a transport protocol that needs to decide whether it can trust the timing feedback information being returned from a remote peer.

Dataset	Relative offset	Likelihood of skew
$\mathcal{N}_1$	< 0.01 sec	0.95%
$\mathcal{N}_1$	< 0.1 sec	5.6%
$\mathcal{N}_1$	< 1 sec	13 %
$\mathcal{N}_1$	≥ 1 sec	12 %
$\mathcal{N}_2$	< 0.001 sec	1.3 %
$\mathcal{N}_2$	< 0.01 sec	0.88 %
$\mathcal{N}_2$	< 0.1 sec	1.3 %
$\mathcal{N}_2$	< 1 sec	1.8 %
$\mathcal{N}_2$	≥ 1 sec	5.3 %

Table 2: Relationship between relative clock accuracy and clock skew

Table 1 shows the relationship between relative clock accuracy and the likelihood of observing a clock adjustment. We see that closely synchronized clocks, i.e., those with a relative offset under 1 sec, are only slightly less likely to exhibit a clock adjustment than less closely synchronized clocks. Thus, relative clock accuracy is not a good predictor of the absence of clock adjustments.

Table 2 shows the relationship between relative clock accuracy and the likelihood of observing relative clock skew. For $\mathcal{N}_1$, clock synchronization only provides an advantage if the clocks are highly synchronized, with a relative offset under 100 msec and preferably under 10 msec. For $\mathcal{N}_2$, however, synchronization of under 1 sec provides a definite advantage in predicting a lower likelihood of skew, though much better synchronization provides little additional predictive power. For both $\mathcal{N}_1$ and $\mathcal{N}_2$, not even very close synchronization reduces the likelihood of encountering clock skew to a negligible level (i.e., appreciably lower than 1%).

We conclude that relative clock accuracy provides no benefit in assuring that clock adjustments will be unlikely, and some benefit in assuring that clock skew is less likely, but not to such a degree that we can ignore the possibility of clock skew when analyzing more than a handful of measurements.

In addition, we conjecture that the closely-synchronized hosts in our study are most likely synchronized using NTP. If so, then the use of NTP does *not* reduce the likelihood of clock adjustments introducing systematic errors when measuring packet transit times, and reduces but does not eliminate the likelihood of clock skew introducing systematic errors. This finding does *not* mean that NTP fails to keep good time. Rather, the timescales on which it does so significantly exceed those of our connections. NTP keeps good time on large time scales precisely by altering clock behavior on small time scales.

8 Summary

The problem of comparing timestamps between unsynchronized clocks might at first appear relatively minor. But, as we developed in the introduction, it actually has significant impact on the accuracy of wide-area network measurement. If we can compare such timestamps reliably, then we can use "receiver-based" measurement in order to directly measure the properties along one direction of a network path, rather than unavoidably conflating these properties with those along the reverse path, as happens with "echo-based" measurement.

Unsynchronized clocks are subject to at least two types of errors: clock adjustments, in which one of the clocks rapidly changes its current setting, and relative clock skew, in which one clock runs faster than the other. If undetected, both of these can introduce measurement artifacts that can masquerade as changes in delay due to genuine networking effects. In this paper we have undertaken to develop robust algorithms for detecting both adjustments and relative skew, even in the presence of significant noise in the timing measurements. While our algorithms require some heuristic tuning to minimize inaccuracies in terms of false positives and false negatives, with this tuning in place we find that they appear reliable, as best as we can judge without a source of independent calibration.

In summary, prudent large-scale measurement and analysis of packet timings should include algorithms such as these as self-consistency checks to detect possible systematic errors, even in the presence of synchronization via algorithms such as NTP, which we find does not render clocks immune from errors (§ 7). We further argue that even pairs of clocks using a more direct external synchronization source such as GPS should be subjected to such checks, as a means of assuring that no timing errors have crept in between the original, highly accurate time source, and the packet timestamps ultimately produced by the inevitably imperfect computer clocks.

9 Acknowledgements

This work greatly benefited from discussions with Domenico Ferrari, Sally Floyd, Van Jacobson, Mike Luby, Greg Minshall, John Rice, and the comments of the anonymous referees. My heartfelt thanks.

References

[Bo93] J-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. SIGCOMM '93*, pp. 289-298, Sep. 1993.

[CC96] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," *Performance Evaluation*, Vol. 27-8, pp. 297-318, Oct. 1996.

[CPB93] K. Claffy, G. Polyzos and H-W. Braun, "Measurement Considerations for Assessing Unidirectional Latencies," *Internetworking: Research and Experience*, 4 (3), pp. 121-132, Sep. 1993.

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

[JLM89] V. Jacobson, C. Leres, and S. McCanne, `tcpdump`, available via anonymous ftp to ftp.ee.lbl.gov, Jun. 1989.

[Ja97] V. Jacobson, "pathchar — a tool to infer characteristics of Internet paths," ftp://ftp.ee.lbl.gov/pathchar/msri-talk.ps.gz, Apr. 1997.

[Ke91] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. SIGCOMM '91*, pp. 3-15, Sep. 1991.

[Mi92a] D. Mills, "Network Time Protocol (Version 3): Specification, Implementation and Analysis," RFC 1305, Network Information Center, SRI International, Menlo Park, CA, Mar. 1992.

[Mi92b] D. Mills, "Modelling and Analysis of Computer Network Clocks," Technical Report 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.

[Mi95] D. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks," *IEEE/ACM Transactions on Networking*, 3(3), pp. 245-254, Jun. 1995.

[Mu94] A. Mukherjee, "On the Dynamics and Significance of Low Frequency Components of Internet Load," *Internetworking: Research and Experience*, Vol. 5, pp. 163-205, Dec. 1994.

[Pa96] V. Paxson, "End-to-End Routing Behavior in the Internet," *Proc. SIGCOMM '96*, pp. 25-38, Aug. 1996.

[Pa97a] V. Paxson, "End-to-End Internet Packet Dynamics," *Proc. SIGCOMM '97*, Sep. 1997.

[Pa97b] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," *Proc. SIGCOMM '97*, Sep. 1997.

[Pa97c] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. dissertation, University of California, Berkeley, April 1997.