Abstract

Network one-way transit time is one of the principal factors affecting network QoS. It is therefore important to measure it with sufficient accuracy. The measurements of network one-way transit time should not include host effects (time spent of the measurement hosts themselves) but be entirely attributable to delays incurred on the network. Furthermore, the measurements of network one-way transit time should not be affected by the relative offset and skew of the clocks of the measurement hosts. This paper proposes a methodology for such measurements. The accuracy achieved by the proposed method is sufficient to detect deteriorating network conditions.

1 Introduction.

While there are many factors contributing to network QoS, network transit times, and the related loss and jitter, are commonly used as the main predictor of network QoS. We shall be concerned with the end-to-end network round trip time which is the time it takes for a packet to travel from computer A to computer B and back to A, and with the end to end network one-way transit time which is the time it takes for a packet to travel from computer A to computer B.

When measuring network transit times, care should be taken not to include any delay which is not attributable to the network itself. For example, delays caused by latencies within the computers A and B should not be attributed to the network. In addition, since clocks from two different computers are involved in measuring one-way transit times, the relative clock offset and skew need to be taken carefully into account.

This paper makes two contributions. First we propose a method for the joint estimation of the clock skew and offset based on a sequence of timestamps collected in the course of packet round trips. The estimated clock skew and offset can be used to obtain one-way transit times. Second we propose a modification of the Linux operating system that enables the collection of transit times that can mostly be attributed to the network rather than to the host itself. The paper also presents a comparison of measurements made using the proposed system against measurements made using a commercial system designed to evaluate the performance of networked applications.

The problem of reconciling time measurements made by two different clocks is an important one and has been studied extensively. One of the most accurate methods to deal with the issue involves GPS hardware, but given the associated costs, many authors feel that efforts should be made on the basis of standard equipment and accordingly have proposed methods and algorithms to synchronize clocks. The synchronization of clocks over long periods may be achieved with NTP (Network Time Protocol – see [1]) but for the measurement of accurate packet level one-way transit times it is not adequate, precisely because NTP works by making occasional adjustments which aim at a long term accuracy and, in fact, interferes with the measurement process itself. See Pasztor [3] and Paxson [4] for further discussion of this point.


Although the estimation of clock skew is sufficient for many purposes, having sufficiently accurate measurements of one-way transit times can be very useful in attributing blame to possible failing network elements. For this reason, we propose a method that estimates both the clock offset and skew. The accuracy in clock offset estimation is limited by the speed at which the two computers can communicate. With the approach we propose, clock offset is estimated using the underlying network itself and may be compared to one half of a record round trip although we actually decouple the optimization in each of the two directions. The end result is an accuracy which compares favorably to normal network conditions.

The progression of time on one computer is a central concept in the paper. This progression is affected by many things including, natural variations within the tolerance of resonating crystals, temporal variations in voltage and or temperature, the aging of the crystals and, of course, manual or automatic interventions such as those accomplished by NTP. In this paper, we assume that a linear progression of the real time relative to measured time is a sufficient model over the period of measurement:

\[ t = \delta + \rho T \]  

where \( \delta \) and \( \rho \) represent the offset and the skew of the clock relative to the ideal clock. A priori, \( \delta \) and \( \rho \) are not known and \( \rho \) is normally very close to 1 (the resonating crystals oscillate at close to their nominal frequency and/or the best possible adjustment for whatever discrepancy between nominal and actual frequency has been made within the computer’s time keeping code). The parameter \( \delta \), on the other hand, can be very far from 0. The parameters \( \delta \) and \( \rho \) are specific to individual computers. For the rest of the paper, since the context requires more than one clock, one of them is assumed to be the ideal clock.

Manual or automatic interventions such as those accomplished by NTP would violate the above linear model, and we assume that there was no such intervention. Handling such interventions is a problem that is discussed in Zhang [5]. In particular, for clock resets, they propose a piecewise linear model and estimates both the number of pieces together with the slopes and intercepts. Both Paxson [4] and Zhang [5] have examples with real data exhibiting sudden discontinuities in the measured one-way transit times and suggest that the likely explanation is clock resets. That may very well be the case for their particular data, but in our experience network conditions are quite capable of producing complex patterns of discontinuities, particularly in situations where there is load balancing.

2 Kernel timestamps.

Two computers can be used to measure network transit times, one-way or round trip times. Consider the measurement of network round trip times. Computer sends a packet to computer B which simply returns the packet back to A. There are 4 timestamps associated with this scenario, S, U, V and T as show in figure 1. The network round trip time can be defined as \( R = (T - S) - (V - U) \). The basic ingredients to the collection of the above four timestamps would be operating system utilities such as time, send and recv, standard POSIX functions to read the time, to send and to receive a packet respectively. It should be noted, however, that as a packet is sent, there is a gap in time between when the time function is called and when the packet is actually on the wire and another gap, as a packet is received, between when a packet is received and when the time is noted. Figure 2 illustrates this elaboration, introducing kernel timestamps S', U', V'
Figure 2: Observed one-way delays

and T’ which represents the times at which the packet enters and leaves the wire. The gaps, between the timestamps and their kernel counterparts represent operating system latencies and should really not be attributed to the network. The network round trip time can be refined to

$$R' = (T' - S') - (V' - U'),$$

which amounts to the extraction of operating system latencies from the former definition of R.

In practice, without specialized hardware, it is not possible to measure S’, U’, V’ and T’, but we have modified the Linux operating system kernel in order to collect something as close as possible to them using standard hardware. The Linux operating system already has a facility to timestamp packets immediately as they are received, before they are delivered to the target application. The SIOCGSTAMP socket ioctl can be used to retrieve this timestamp, which is how we gather S and U. The Linux operating system has no corresponding facility to collect timestamps for outgoing packets, however. It is possible to modify an ethernet driver so that it will collect a timestamp just before putting the packet on the wire. Such an outgoing packet kernel timestamp could be stored in the packet itself or possibly stored in kernel space to be retrieved by the application at some later time. The practical difference between user land and kernel space timestamps will be illustrated in Section 4.

3 One-way Delay and Round Trip Time.

Assuming a linear relationship between measured and true time, clock offset is irrelevant to network round trip times. Indeed, since the measured departure and arrival times T₀ and T₁ are measured on the same computer, the true round trip time is

$$t_1 - t_0 = \rho(T_1 - T_0).$$

The estimation of the true round trip time can be done without estimating δ.

The measurement of one-way delay is not so simple because it involves two clocks on which time progresses differently. Let T₀ be the measured departure time (on A) and T₁ be the measured arrival time (on B). Assuming that A has a perfect clock, the true one-way delay is

$$t_1 - t_0 = (\delta + \rho T_1) - T_0$$

where δ and ρ are the offset and skew of the clock on B relative to A. Evidently, the estimation of the true one-way delay cannot be done without estimating both δ and ρ.

Assume that we have observed timestamps for a number of packets traveling from A to B and back to A. We present a method to estimate the skew and offset between the two clocks involved. As it will be illustrated below, the method may be applied to either user land or kernel space timestamps. With each of the packets, we associate four timestamps denoted S, U, V and T which are, in that order, departure from A, arrival at B, departure from B and arrival at A. For n packets, we end up with 4n timestamps, indexed by 1 ≤ i ≤ n. Without loss of generality, we will assume that the clock on computer A is perfect.

Note that the events departure from A, arrival at B, departure from B and arrival at A are certainly ordered in true time. Accordingly, we know that

$$S_i < \delta + \rho U_i < \delta + \rho V_i < T_i \quad 1 \leq i \leq n$$

or, equivalently,

$$\max_{1 \leq i \leq n} \{S_i - \rho U_i\} \leq \delta \leq \min_{1 \leq i \leq n} \{T_i - \rho V_i\}. \quad (5)$$

This defines a set $D$ of couples $(\rho, \delta)$ satisfying (4) which are all consistent with all of the observed values, that is, linear corrections of the time on B that are such that when they are applied, the known ordering of the events is satisfied. Thus, we define as estimates of ρ and δ any pair that belongs to $D$. The set $D$ will not be empty if the progression of time on B relative to A really is linear. It is
not hard to see that $D$ is a convex set and that a point in $D$ can easily be found by linear programming methods. Also note that if $(\rho, \delta') \in D$, and

$$\delta = \frac{1}{2} \left\{ \max_{1 \leq i \leq n} \{S_i - \rho U_i\} + \min_{1 \leq i \leq n} \{T_i - \rho V_i\} \right\}$$

(6)

then $(\rho, \delta) \in D$.

A simple minded (and commonly proposed) adjustment for clock offset can be made on the basis of a record round trip as follows:

$$\delta = U_i - S_i + \frac{1}{2} \left((T_i - S_i) - (V_i - U_i)\right)$$

(7)

where $i$ is chosen to minimize $(T_i - S_i) - (V_i - U_i)$ over $1 \leq i \leq n$. This simplification, can be derived from equation (6) by assuming that $\rho = 1$ and that the minimum and the maximum are reached at the same index $i$. The method we propose decouples the two optimizations resulting in the smallest possible set $D$.

### 4 Examples and discussion.

We now illustrate the above with a few examples. The examples are based on packets exchanged between single board computers manufactured by Axis with a CPU that runs at 100MHz. The computers are at various places in the Avaya network. The tests are performed while the computers were otherwise not busy. A few seconds before each test, we set the origin of time on all of the computers involved.

For the first example, the timestamps are based on 120 UDP packets of length 1 kilobytes that traveled between a pair of computers at the rate of 1 per second. Figure 4 shows the series of observed one-way delays, the series $S_i - U_i$ (bottom lines) and $T_i - V_i$ (top lines) for $1 \leq i \leq n$.

The outside lines correspond to user land timestamps while the pair of inside ones correspond to kernel space timestamps. The downward slope reveals that time progresses faster on computer B than on computer A (U’s and V’s are measured on computer B). The gap between the two lines represents the round trip times, $(T_i - V_i) - (S_i - U_i) = (T_i - S_i) - (V_i - U_i)$ for individual packets.

The gap between the kernel space timestamps series are substantially smaller than the gaps for their user land counterparts. This is because using kernel timestamps extracts operating system latencies from what would otherwise be attributed to the network. From the relative smoothness of the kernel timestamp series, we can also conclude that using kernel timestamps also extracts substantial (if not all) operating system jitter from what would otherwise be attributed to the network.

Figure 5 shows the two sets $D$ based on user land and kernel space timestamps, illustrating the reduction in uncertainty about $\rho$ and $\delta$ jointly the kernel timestamps represent. The set $D$ is the set of points that is at the intersection of the 240 half planes of relation (4). The set $D$ is below the straight lines $\delta + \rho V_i = T_i$ at the top right and above the straight lines $\delta + \rho U_i = S_i$ at the bottom left. Every straight line that fits between the two curves of Figure 4 corresponds to a point within the set $D$.

Our introduction states the proposed method is sufficient to detect deteriorating network conditions. Figure 6 illustrates this point. The top left frame displays a histogram of round trip times between the two computers. The top right and bottom left frames display the corresponding histograms of one-way transit times in each of the two directions. The three histograms use the same time divisions and scale. For this data, the projection of $D$ on the $y$ axis goes from -5.1 to -5.05 and we estimate $\delta$ to be -5.075 with an error of at most 0.025. The three histograms show a vertical line at $\rho = 0.025$ putting the uncertainty we have about $\delta$ in perspective comparing it to the delays measured on the network. The bottom right frame shows the corresponding series of observed one-way delays, the series $S_i - U_i$ (bottom lines) and $T_i - V_i$ (top lines) for $1 \leq i \leq n$.

Our method thus produces a partitioning of the round trip time into a pair of one-way delays with an accuracy that exceeds (because of the decoupled minimization) the best network round trip times. On a network where there is delay variation, this accuracy is necessarily sufficient to detect changing network conditions.

We have compared network one-way transit times measured with a well known commercial product to which we refer as Commercial below to those obtained with the measurement system we propose which we call Babel for
historical reasons. Our measurement system is based on
our modified Linux kernel and an application that estimates $\rho$ and $\delta$ as described above to derive network one-
way transit times.

We have performed comparisons under four experimental conditions. Each comparison confronts measurements made with Commercial and Babel. Figure 3 shows the layer 3 topology for the tests. The Commercial tests use endpoints A and B while the Babel tests use endpoints C and D so that each test goes through 2 hops at layer 3. Endpoints A and B are a 400MHz Pentium II desktop with 256 MB of memory and a 1.8 GHz Pentium IV laptop with 512 MB of memory. Endpoints C and D are a pair of single board computers (Developer Board LX) manufactured by Axis Communications (www.axis.com) based on a 100MHz RISC ETRAX cpu and with 8 MB of memory.

The size of packets (28 bytes) and their frequency (50 packets per second) during the test corresponds to a 5 minute long VoIP call with the G.729 Speech Codec. The forward and reverse network one-way transit times (15000 of them in each direction) are summarized into 600 forward and 600 reverse averages that correspond to a 1 second timing record.

The comparisons are designed to illustrate the advantages of kernel timestamping and to use joint estimates of $\rho$ and $\delta$ to derive network one-way transit times. In the first and third comparison, the measurement software is running on an otherwise quiet cpu; in the second and fourth comparison, it is competing against some cpu intensive process. In the first and second comparison, the tests are performed on an otherwise quiet network; in the third and fourth comparison, an intensive TCP stream is injected from E to F during the middle two minutes of the test.

Figure 7 shows the forward and reverse series under conditions 1 and 2 (no competing traffic) while Figure 8 shows the forward and reverse series under conditions 3 and 4 (intense competing traffic). Top and bottom frames correspond to Babel and Commercial respectively. The middle frames show ping round trip times which could serve as a bound for the sum of the two one-way delays. Left and right correspond to no cpu load and intense cpu load respectively.

Note that the frames of each Figure are on the same scale except for the Commercial - no competing traffic - with load frame which had to be scaled separately to avoid dwarfing the other frames. It is clear that cpu load has a drastic effect on Commercial to the point that it invalidates the reported one-way delay measurements, with or without the presence of competing traffic.

In contrast, the one-way delay measurements reported by Babel compare meaningfully to the ping round trip times and appear to be unaffected by cpu load.
Figure 4: Observed one-way delays
Figure 5: The set $D$
Figure 6: Partitioning round trip time.
Figure 7: Babel and Commercial (no competing traffic).
Figure 8: Babel and Commercial (intense competing traffic).
References


