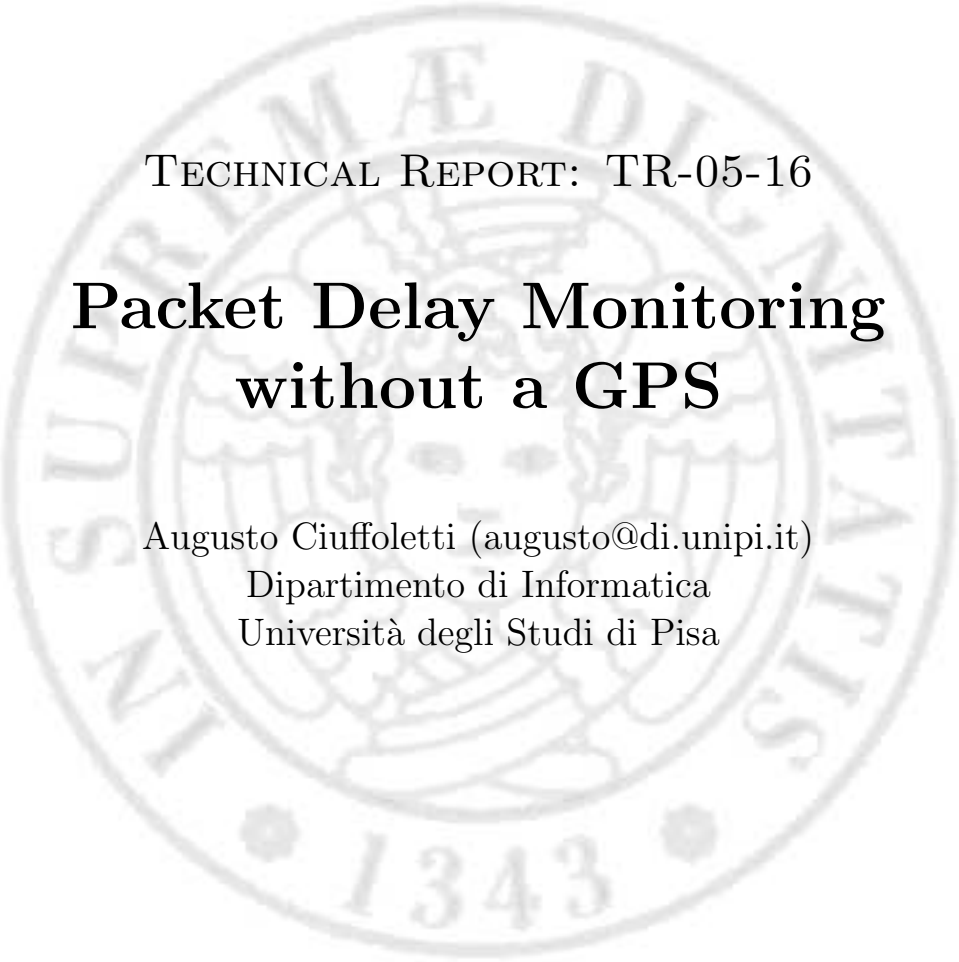
Università di Pisa

Dipartimento di Informatica



Technical Report: TR-05-16

Packet Delay Monitoring without a GPS

Augusto Ciuffoletti (augusto@di.unipi.it)
Dipartimento di Informatica
Università degli Studi di Pisa

June 1, 2005

Address: via F. Buonarroti 2, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Packet Delay Monitoring without a GPS

Augusto Ciuffoletti (augusto@di.unipi.it)
Dipartimento di Informatica
Università degli Studi di Pisa

June 1, 2005

Abstract

This paper illustrates a technique to characterize one-way packet delays. In the spirit of [9] such technique does not depend on external clock synchronization facilities: instead, it computes the relative skew of the two clocks, and uses this information to characterize one-way packet delays.

We focus on the applicability of such technique to a network monitoring environment, and perform an extensive test that includes second order clock frequency variations. We conclude that the approach is especially suited to estimate jitter, and other "local" characteristics of one-way delay variation. It can be helpful also to evaluate second order parameters of one-way delay, like standard deviation, when second order variations of clock frequency, usually due to temperature variations, are compensated by hardware.

This result makes *one-way delay variations* measurement widely accessible to Internet hosts, at a cost which is overall comparable with that of a `ping`.

1 Introduction

Packet delays are good indicators of the quality of a communication infrastructure: the presence of anomalous increments of such delays is a symptom of adverse events occurring along the route followed by packets. For instance, an upcoming congestion or a route change are associated to delay variations.

The most common technique to obtain an indication of communication quality based on packet delays is to send a request packet to a target, and wait for a reply: the difference between the send time of the request, and the receive time of the reply corresponds to the sum of the delays of the

two packets, plus some computational overhead. A series of such *roundtrip* measurements is useful to analyze the behavior of the path between the two hosts [6].

A relevant advantage of the *roundtrip* technique is the limited overhead, and the independence from an accurate timing. An evident drawback is that this technique mixes the observation of two distinct paths: the forward path, followed by the request, and the backward path, followed by the reply. The purpose of this paper is to investigate a technique that preserves as much as possible the advantages of the *roundtrip* approach, while allowing the characterization of the delay of each packet.

The measurement of a *real one-way delays* series requires that the two sites performing the measurement have access to synchronized clocks, whose accuracy must be negligible with respect to the measured delay: the experiment reported in [7] follows this headline. Devices performing such measurements are designed *ad hoc*, and therefore the availability of one-way delay observations is limited, and expensive.

In many applications *delay variations* are more useful than *real* one way delays [1]: for instance, we may want to estimate the difference between the delays of adjacent packets, that we call *delay jitter*, or statistical parameters, like *standard deviation*. Among other uses, the former is an indicator of an upcoming congestion, while the latter reveals self-dependency patterns [10].

To analyze *one-way delay variations* it is sufficient to measure *one-way delay deviations*, that correspond to *one way delays* decremented by an unknown, but constant, *delay offset*. The system requirements for the measurement of *one-way delay deviations* are milder, since we only need that the clocks of the two hosts tick with the same frequency. The software implementation of this requirement is known as *clock skew compensation*.

Following the headline of [9] we design a *clock skew compensation* algorithm that is designed *ad hoc* for the specific purpose of computing *one-way delay deviations* series. The basic idea is similar to that introduced in [2], and consists in computing the convex hull that contains uncompensated roundtrip observations. Like [8] we use "Graham's scan" [5] to compute the lower hull.

Our design specifically addresses a *Network Monitoring* environment, that is illustrated by the following use case:

> A network monitoring tool **NM** installed on host **X** sends regularly (not necessarily with a constant period [4]) *request* packets to host **Y**, which returns each time a *reply* packet.
>
> At any time **NM** may receive a *service* request, the service consisting in computing and returning to the querier a function

f of the *delay deviation* of the packets exchanged between **X** and **Y** (e.g. the jitter).

The basic observation is that the service time of **NM** should depend on the number of observations that need to be accessed to compute f: for instance, the response time for the jitter at time t should be O(1). The response time cannot depend on the number of measurements in the database!

Since [8] proves that the computation of the *clock skew compensation parameters* cannot have a complexity less than O(N), we cannot compute them during the service time: therefore, since such parameters are needed to compute the *delay deviations* (the arguments of f), they must be computed each time a new roundtrip observation becomes available.

A consequential observation is that the computation of the new *clock skew compensation parameters* must be inexpensive, since it is repeated for each roundtrip: again, a O(N) cost is unacceptable.

The algorithms introduced in [9] and [8] share a feature: they scan the whole roundtrip observation list, in order to compute a corresponding list of compensated roundtrips. Using such algorithms, the cost for computing the new list of compensated roundtrips upon observation of a new roundtrip is O(N): this is in contrast with our requirements. These algorithms are meant to process a packet trace and extract from it a graphical representation together with statistical characteristics. This is a *Network Administration* environment, which does not fit our *Network Monitoring* perspective.

The algorithm we introduce in this paper is based, like Moon's one, on the use of the "Graham scan" algorithm [5, 11, 3] to solve a geometric problem (the details are in section 2.1.2). We propose an implementation that is oriented to a *Network Monitoring* application: the update of the *clock skew compensation parameters*, when a new roundtrip observation becomes available, does not entail the scan of all observations, but a subset that we prove to be O(logN).

In section 2.1 we formally define the *clock skew compensation* technique, that we use to prove that the algorithm is correct and has an expected time and space complexity O(logN):

1. we discuss the properties of the data that can be collected using the local clocks as time references, and we show how to compute apparent delays that are potentially associated to packets that experienced minimal delays: the algorithm to do this is recursive, and has a cost that does not exceeds the number of potential minimums collected so far;

2. we prove that the number of minimums tends to be logarithmic in the dimension N of the series;

3. we introduce a criterion to identify, with high probability, the two minimals: the line that interpolates these two points is a good approximation for a line parallel to the clock difference function, that can be used to compute the *clock skew compensation parameters*.

To complement the expected features of the algorithm with the assessment of its reliability, we check its behavior against packet traces whose *clock skew compensation parameters* are known. We also investigate the response of the algorithm to skew variations induced by thermal variations.

Part of the above results can be extended to Moon's algorithm, whose validation was limited to the few case studies illustrated in the referenced report [8].

The appendixes are dedicated to the exhaustive formal proofs.

2 One-way delay deviation

An *event* is associated with an *occurrence time* and a *component* where it occurs. We indicate with $t(e)$ the occurrence time of event e. An estimate is obtained reading the *physical clock* of the component where the event occurs: the value at time $t(e)$ of the clock of the component where e occurs is indicated with $ts(e)$. The *offset* $g(e)$ at time $t(e)$ of the clock of the component where e occurs, is indicated by

$$g(e) = t(e) - ts(e) \tag{1}$$

In a system of two components, the time interval $\Delta(e_1, e_2)$ between two events e_1 and e_2, each occurring on a distinct component, is:

$$\Delta(e_1, e_2) = t(e_2) - t(e_1) = (ts(e_2) - ts(e_1)) + (g(e_2) - g(e_1)) \tag{2}$$

One-way (packet) delay corresponds to the length of the time interval between send and receive events of a packet. Let snd_i and rcv_i be the send and receive events of the i-th packet in a sequence of n, the corresponding sequence of packet delays is indicated as:

$\mathbf{\Delta} = (\Delta_i, i \in [0, n-1])$
where $\Delta_i = \Delta(snd_i, rcv_i) = ts(rcv_i) - ts(snd_i) + (g(rcv_i) - g(snd_i))$

One-way delay deviation is defined as the difference between the real delay, and a constant *delay offset*:

Definition 1 *Let* $\mathbf{\Delta} = (\Delta_i, i \in [0, n-1])$ *be a packet delays sequence. A sequence* $\mathbf{D} = (D_i, i \in [1, n-1])$ *is a packet delay deviation sequence for* $\mathbf{\Delta}$ *if a delay offset* K *exists such that*

$$\forall i \in [0, n-1], D_i = \Delta_i - K$$

We conclude the formal introduction of the *one-way delay deviation* noting that $t(e)$ and $g(e)$ are not available. On the contrary, $ts(e)$ can be read with a good approximation.

2.1 Computing a packet delay deviation series

In the case of a crystal clock, the offset $g(e)$ defined by equation 1 can be modeled using a polynomial whose linear term dominates higher order terms. In fact, the linear term, called *skew*, is due to crystal cut tolerance, on which clock characteristic frequency depends. Higher order terms depend on the nature of the crystal: they are influenced by environmental conditions (like temperature or aging), or induced by clock synchronization.

We can use a linear approximation of the difference between two clocks, and approximate the *relative offset* as:

$$g_{s,r}(e) = (a_{s,r}t(e) + g_{s,r}(0)) \tag{3}$$

Where $a_{s,r}$ is the *relative skew*, and $g_{s,r}(0)$ is the *relative clock offset* at origin. The *one-way delay* can be written as follows:

Lemma 1

$$\Delta(snd_i, rcv_i) = ts(rcv_i) - ts(snd_i) + a_{s,r}t(snd_i) + g_{s,r} \tag{4}$$

The formal proof is in appendix A. Note that

- ts terms are known, since they correspond to local clock values,
- $a_{s,r}$ and $g_{s,r}(0)$ are unknown constants that describe the relationships between the two clocks,
- $t(snd_i)$ is the time of the send operation, which is in principle not observable.

Either $a_{s,r}$ or $g_{s,r}(0)$ can be bound if we select two messages p and q such that the corresponding delays are identical. We will discuss the reasonableness of this assumption in the next section: here we assume that such two indexes are known. Using that assumption, we write the following expression for the *one way delay*:

Lemma 2

$$\begin{aligned}\Delta(snd_i, rcv_i) \;=\; & (ts(rcv_i) - ts(snd_i)) - \\ & \tfrac{ts(snd_i)-ts(snd_p)}{ts(snd_q)-ts(snd_p)}((ts(rcv_p) - ts(snd_p)) - (ts(rcv_q) - ts(snd_q))) - \\ & (ts(rcv_p) - ts(snd_p)) + \\ & \Delta_p\end{aligned}$$

The exhaustive proof is in appendix B. Here we only observe that, comparing the above equation with the definition of delay deviation, the right term is a valid delay deviation for packet i, when the constant term K in definition 1 is Δ_p.

Theorem 1 *Modeling local clocks with a constant* skew *and* origin offset*, let* $\mathbf{\Delta}$ *be a sequence of packet delays, and* p*,* q *the indexes of two packets such that* $\Delta_p = \Delta_q$*.*

The sequence $\mathbf{D_p} = (D_i, i \in [0, n-1])$ *where*

$$\begin{aligned}D_i \;=\; & (ts(rcv_i) - ts(snd_i)) - \\ & \tfrac{ts(snd_i)-ts(snd_p)}{ts(snd_q)-ts(snd_p)}((ts(rcv_p) - ts(snd_p)) - (ts(rcv_q) - ts(snd_q))) - \\ & (ts(rcv_p) - ts(snd_p))\end{aligned}$$

is a one-way packet delay deviation *sequence for* $\mathbf{\Delta}$ *with offset* Δ_p*.*

Summarizing, in order to compute the *one-way delay deviation* of a packet i, one needs to know:

- the local timestamps of the send and receive events for the packet, $ts(rcv_i)$ and $ts(snd_i)$;
- the local timestamps for the send and receive events of the two packets that experienced identical delays, $ts(rcv_p)$, $ts(snd_p)$, $ts(rcv_q)$, $ts(snd_q)$.

Such values are contained in the observations database. However we introduce a new subproblem, that consists in finding the indexes of two packets with identical delays: the next section introduces an algorithm that solves this problem.

2.1.1 Selecting packets with identical packet delay

In practice, we cannot find two packets with exactly identical delays: however we can approximate this result, finding two packets that experienced close

delays. This imprecision will affect the estimate of the drift $a_{s,r}$: let ϵ be the difference between the two delays, $a_{s,r}$ will fall in an interval whose width is

$$\frac{\epsilon}{|ts(snd_q) - ts(snd_p)|} \tag{5}$$

We will call this value the *residual skew*, meaning that the resulting *one-way delay deviation* will be as accurate as one obtained using clocks whose skews are compensated within the *residual skew*.

In order to minimize this figure, we should minimize ϵ, and maximize the distance between the two observations: a hard task, for which it is appropriate to look for an *ad hoc* heuristic. In this context, *ad hoc* means that the heuristic is designed for *clock skew compensation*, and might be unsuccessful when applied to a different problem.

The heuristic we introduce is split into two parts, each aiming at optimizing part of our target:

- we select a subset of the observations that contains several packets that experienced the smallest delays;
- we select the two successive that are separated by the largest gap.

A relevant feature of the above heuristic is that an algorithm exists that implements it in logarithmic time: in other words each time a new observation is available, the update of p and q has a cost that is a logarithm of the number of observations in the database, and does not keep increasing linearly with its size.

The evidence for the validity of such heuristic is partly experimental: we are not able to find a probabilistic model that exactly describes the dynamics of the *residual skew*. An approximate model is used to suggest the rationale behind the algorithm, and its expected behavior.

2.1.2 Selecting fast packets

Observed delays can be arranged in a X-Y plot, as in figure 1: each observation is represented by a point with coordinates $(ts(snd_i), ts(rcv_i) - ts(snd_i))$. We want to select observations $I = (t_i, d_i), i \in [1..n]$ such that:

$$\exists a, \forall \text{ observations } (x, y), y > a(x - t_i) - d_i \tag{6}$$

Starting from an elementary result in operational research, we can prove that such set contains, independently from the offset and the drift of the

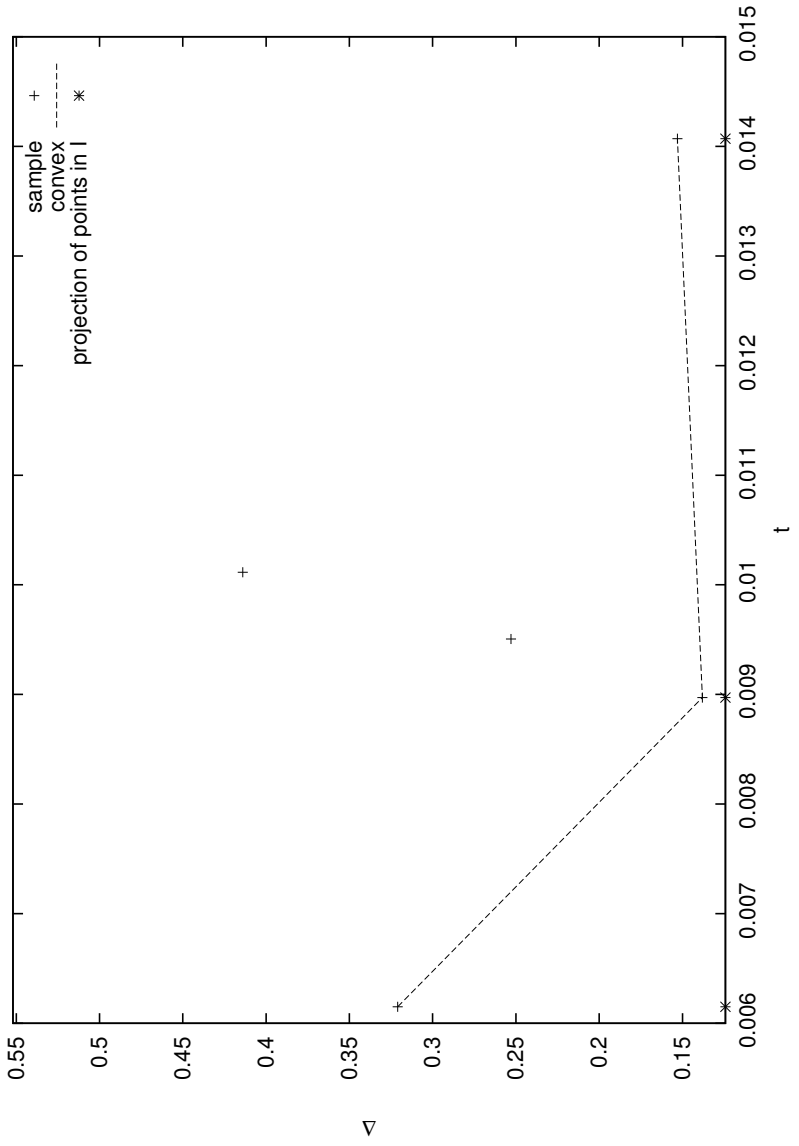


Figure 1: *A X-Y representation of one-way delays: X is the send time of a packet (in seconds starting from the send of the first packet), Y is the measured delay in milliseconds. The dashed line corresponds to the lower convex hull described in section 2.1.2. t values marked with a* `x` *indicate observations collected in set I by Graham algorithm (see algorithm 1)*

local clocks, the observation of the fastest packet, and of the second fastest ones among those that precede and follow.

The algorithm in figure 1 implements a stepwise computation of such set: each time a new step is performed, the algorithm recomputes a new set I. A detailed description of the algorithm, which is a variation of "Graham's scan" [5], and its proof are in appendix C.

Algorithm 1 *Insertion of a new observation in I according with Graham algorithm*

$$\textbf{Require:} \begin{cases} \mathbf{m} = ((d_0,t_0),...,(d_{N-1},t_{N-1})) \wedge \forall i,j \in [0,N-1], (i<j) \leftrightarrow (t_i < t_j) \\ \mathbf{I} = (i_0,...,i_{r-1}), \begin{cases} \forall x \in [0,r-1], (d_{i_j},t_{i_j}) \in I(\mathbf{m}) \wedge \\ \forall x,y \in [0,r-1], (x<y) \leftrightarrow (t_{i_x} < t_{i_y}) \end{cases} \\ (d_N,t_N) \wedge t_{N-1} < t_N \end{cases}$$

define $a_{i,j} = \frac{d_j - d_i}{t_j - t_i}$

if $r > 2$ **then**

$p = r-1$

while $\left(p > 1 \ \wedge \ a_{i_{p-1},i_p} \geq a_{i_p,N}\right)$ **do**

pop **I**

$p = p-1$;

$$\textbf{Ensure:} \quad \forall y \in [i_p+1, N], a_{i_p,y} \geq a_{i_p,N}$$

end while

$$\textbf{Ensure:} \begin{cases} \forall x \in [0, i_p-1], a_{x,i_p} \leq a_{i_p,N} \\ \forall y \in [i_p+1, N], a_{i_p,y} \geq a_{i_p,N} \end{cases}$$

end if

push $\mathbf{m}$,(d_N,t_N)

push $\mathbf{I}$,(d_N,t_N)

$$\textbf{Ensure:} \begin{cases} \mathbf{m} = ((d_0,t_0),...,(d_N,t_N)) \wedge \forall i,j \in [0,N], (i<j) \leftrightarrow (t_i < t_j) \\ \mathbf{I} = (i_0,...,i_{s-1}), \begin{cases} \forall x \in [0,s-1], (d_{i_j},t_{i_j}) \in I(\mathbf{m}) \wedge \\ \forall x,y \in [0,s-1], (x<y) \leftrightarrow (t_{i_x} < t_{i_y}) \end{cases} \end{cases}$$

The computational complexity of such algorithm is a relevant issue, since it is executed each time a new observation becomes available. The algorithm in figure 1 updates I by pushing first the new observation, and next rescanning I in order to ensure an invariant on its elements: its computational complexity is $O(I)$, and we prove that this corresponds to a $O(logN)$, where N is the number of observations in the series (the classical proof of the complexity of "Graham's scan" proves only that each step has a O(N) cost [3][1]).

[1]more explicitely, a scan of N points has a cost O(N), but the distribution of this cost over the N steps is not determined

N	experiment		theory
10^2	8	$(6, 10)$	11
10^3	10	$(7, 13)$	18
10^4	12	$(10, 15)$	24
10^5	15	$(12, 18)$	31

Table 1: *Numbers of observations in I: N=sequence length, experiment=mean and* 98% *interval,model=*$2log_2(N/2)$

The proof starts from the fact that the observation of packets with low delays can be modeled as a Poisson process: therefore we expect that successive elements in I are separated by intervals that grow exponentially, and then decrease with the same law. In figure 1 the occurrences of packets in I are marked on the X axis: although irregular, the pattern is easily recognized. A pessimistic assumption (not the worst case assumption) is that the two parts are exactly balanced, and that the minimum falls exactly at index $N/2$. In such case the expected number of elements in I is $2log_2(N/2)$. In table 1 this expectation ("theory") is compared against experimental results ("experiment"), and appears quite pessimistic.

Since density is monotonic increasing, we expect that the smallest delay is indicated by the longest interarrival lapse: however, we do not know which of the two extremes of such lapse corresponds to the smallest delay. The second smallest delay would correspond to the other end of one of the two intervals ending at that point. There is no way to decide which of the three alternatives corresponds to the correct selection, so we opt for an heuristic that consists in selecting the two points in I separated by the longest gap. This heuristic has two advantages:

- it maximizes the probability of finding the two minimums and
- it maximizes the distance between the two points, thus optimizing the denominator of the *residual skew*, our target function

The two minimums have a very low probability to occur: being Poisson events, their probability decreases with the inverse of the expected interarrival time. Therefore we expect that they are close to the minimum, and close to each other: so their selection minimizes the numerator of the *residual skew*.

3 Comparison with other algorithms

As for efficiency, the *linear regression* algorithm referenced by [8] is optimal: using known formulas, it is possible to update the *clock skew compensation parameters* in a time $O(1)$. However, as explained in the referenced paper, this approach is unreliable.

The algorithms introduced by V. Paxson [9] as well as the *linear programming* algorithm introduced by S. Moon [8] are not designed to reuse the result of the previous run: they scan the whole series of roundtrips, to compute new *clock skew compensation parameters*. Since their cost is $O(N)$, they are not applicable to our *Network Monitoring* environment.

The linear programming algorithm introduced in Moon's paper is similar to ours, since we both use "Graham's scan" to compute the convex hull (for us the set of potential minimums). The heuristic that selects the points used to interpolate the clock difference is different: in our case we select points that represent close delays, while Moon's algorithm selects those that minimize the sum of the distances between the estimated clock distance function, and the measured delays. While the "close delays" heuristic is well motivated by a probabilistic model, the "least difference" one is intuitively valid, but is not equally founded. However, although very different in concept, the two heuristics return with high probability the same result.

Using our conceptual framework, we are able to prove that each step has an expected time and space complexity of $O(log\ N)$, which is a significant result in our *Network Monitoring* perspective. In addition, using *residual skew* (see equation (5)) we can quantify the accuracy of our estimate.

4 Experimental results

The formal arguments introduced above prove that the algorithm statistically gives the desired results, but fail to assess the reliability of our approach.

In order to give a formal validation we should introduce a probabilistic model for communication delays, which is a piece of information that is hard to obtain, and should take into account peculiar aspects of delay distributions, such as self-dependency [10]. We have therefore opted for an experimental validation of our heuristic, feeding the algorithm in figure 1 with real data.

Ideally, we need a sequence $\mathbf{m}$ of timestamp pairs $(ts(snd_i), ts(rcv_i))$, one for each packet, $ts(snd_i)$ being the timestamp taken at the sender's side at sending time, and $ts(rcv_i)$ the one taken at the receiver's side at receiving time. This sequence should be used as an input for the algorithm, which

returns a sequence of estimated delay variations $\mathbf{D}$. Another series $\mathbf{\Delta}$ should contain accurate measurements of *delay deviations* for the same stream of packets. Checking $\mathbf{D}$ against $\mathbf{\Delta}$ we obtain a dependable indication of the accuracy of our algorithm.

We have found that data known as the *Auckland-IV data set* [7] fits our need: they are publicly available at `http://pma.nlanr.net/Traces/long/auck6.html`, and report packet traces with microsecond accuracy. In particular, we use traces of packets exchanged between two routers. A trace consists of pairs (t_i, Δ_i), where t_i is the timestamp corresponding to the sending event of the i-th packet, and Δ_i is the real delay experienced by that packet, measured with microsecond accuracy. The sample we are going to consider consists of delays in both directions with a frequency of approximately 230 samples every second during an interval of 6 hours. This amounts to approximately 5 millions measured delays, almost equally distributed between forwards and backwards.

We split this large sample in smaller chunks, in order to collect a statistically significant number of results for input samples of variable length: we split the sample into subsamples of 100, 1000, 10000 and 100000 elements, thus obtaining an insight of how the accuracy improves using larger samples.

To map *Auckland* traces to input series we used three transformations:

- the simplest one consists in using exactly the *Auckland* data as an input sequence $\mathbf{m}$: since the measurement is not affected by clock skews, the output sequence $\mathbf{D}$ should be identical to $\mathbf{m}$, except for a constant difference. Our algorithm ignores that the input is already "correct", and therefore this experiment alone validates the approach;

- a test with a better coverage simulates drifting clocks, according with the constant skew model. Delays from the *Auckland* sequence are transformed according with equation 3, using an arbitrary value for $g_{s,r}(0)$ and $a_{s,r} = 1 * 10^{-3}$ which is consistent with experience. The new sequence is the input sequence $\mathbf{m}$. The result sequence $\mathbf{D}$ should exactly match the results of the previous test, since the algorithm result does not depend on the skew. This transformation is used to validate algebraic aspects of the algorithm, which are left untested by the previous test;

- a more realistic test simulates a slowly changing skew, as in the case of a thermal drift. To this purpose we add to equation 3 a component that varies periodically:

$$g_{s,r}(e) = (a_{s,r}t(e) + 3.6\ P_s A_s sin\left(\frac{2\pi\ t(e)}{3600\ P_s}\right) + g_{s,r}(0))$$

where P_s is the period of the skew variation, in hours, and A_s is the amplitude of the skew variation, in milliseconds. In our experiments we used a maximum skew deviation of 10^{-5} with a period of 2 hours, simulating an ordinary PC equipped with a 50 ppm quartz clock experiencing $20C$ temperature excursions. This transformation stresses the algorithm beyond the assumed *constant skew* model.

For our tests, we used forward and backward delays of the *Auckland* trace dated June 11, 2001: the delays in the two directions have remarkably different distributions, as shown in figure 2.

In order to evaluate the degree of accuracy of the algorithm, we computed two characteristics of the **D** sequences:

- the standard deviation,
- the mean of the absolute value of the difference between successive values, that we call expected jitter:

$$\frac{\sum_{i=1}^{n-1} |D_i - D_{i-1}|}{n-1}$$

The two values are compared with those computed for the $\mathbf{\Delta}$ sequence, and their percent difference is used as an indicator of the accuracy of the estimate: a difference smaller than 10% is tagged **acceptable**, **correct** when below 1%.

The tests have quite different profiles. The test based on the *standard deviation* reflects the accuracy of the estimate of each *delay deviation*, while that based on the *expected jitter* reflects the accuracy in the estimate of the difference between adjacent *delay variations*. While the former returns a preciser indication of the accuracy of the approach, the latter is closer to a real application.

In table 2 and 3 we summarize the results of the first experiment: in both cases the *expected jitter* is correct 99% of times after a "warmup" period of 1000 roundtrips, approximately $17'$ at one roundtrip per second. The *standard deviation* is more problematic, since it takes more time to stabilize: 100000 roundtrips to return a value which is correct 99% of times.

The second experiment returns, as expected, exactly the same results: tables for that experiment are omitted.

The results of the simulation that introduces a thermal drift are reported in figure 4 and 5. As for the *expected jitter*, the experiment confirms the validity of our algorithm even in case of a sensible skew variation induced by an uncompensated quartz clock. On the contrary, the results for the standard

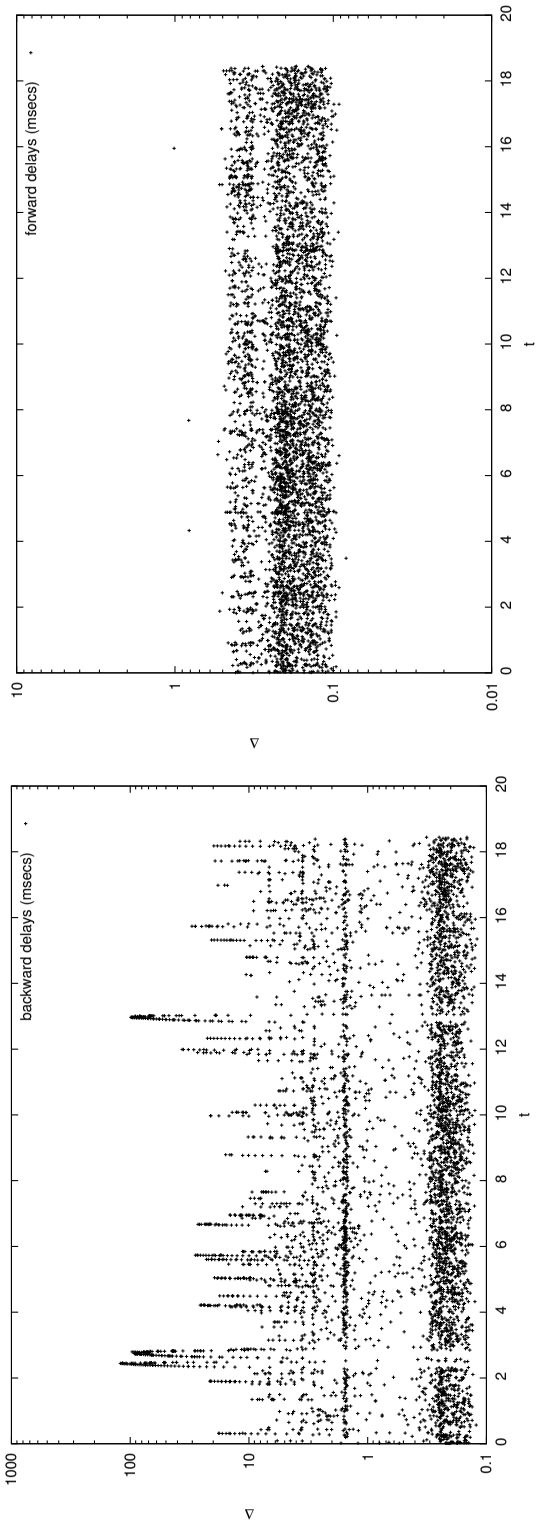


Figure 2: *First 10000 backward and forward delays from* Auckland *trace: t in seconds, Δ in msecs (logarithmic scale).*

		standard deviation		jitter mean	
N	samples	$< 1\%$	$< 10\%$	$< 1\%$	$< 10\%$
100	23498	68.5%	99.0%	99.9%	100.0%
1000	2349	87.4%	100.0%	100.0%	100.0%
10000	234	92.3%	100.0%	100.0%	100.0%
100000	23	100.0%	100.0%	100.0%	100.0%

Table 2: *Comparison of* forward *delays with* $\mathbf{m} = \mathbf{\Delta}$

		standard deviation		jitter mean	
N	samples	$< 1\%$	$< 10\%$	$< 1\%$	$< 10\%$
100	26305	96.4%	97.6%	97.1%	98.9%
1000	2630	99.0%	99.2%	99.6%	100.0%
10000	263	99.6%	100.0%	100.0%	100.0%
100000	26	100.0%	100.0%	100.0%	100.0%

Table 3: *Comparison of* backward *delays with* $\mathbf{m} = \mathbf{\Delta}$

deviation are critical: while in the case of the forward delay the comparison is favorable, in the case of the backward delay the standard deviation of $\mathbf{D}$ *diverges* from the real one! This is due to the fact that in the latter case the input sample is characterized by a very low standard deviation: in such case, the standard deviation component due to uncompensated skew oscillation becomes significant, when compared with the standard deviation of the sample.

		standard deviation		jitter mean	
N	samples	$< 1\%$	$< 10\%$	$< 1\%$	$< 10\%$
100	23498	68.4%	99.0%	99.9%	100.0%
1000	2349	80.5%	99.8%	100.0%	100.0%
10000	234	5.1%	28.6%	100.0%	100.0%
100000	23	0.0%	26.1%	100.0%	100.0%

Table 4: *Comparison of* forward *delays with* $\mathbf{m} = \mathbf{\Delta} + at + g(t) + g_0$

We do not analyze the case of skew variations induced by clock synchronization. This issue is covered in depth in [9], but the techniques illustrated in that paper seem hardly adaptable to a *Network Monitoring* environment: they infer the presence of induced clock anomalies using a "post-facto" attitude, which is inappropriate in our perspective. In our case, when the induced variations are *not* sufficiently smooth to be assimilated to a continuous thermal variation, the presence of inaccurate estimates seems unavoidable:

		standard deviation		jitter mean	
N	samples	$< 1\%$	$< 10\%$	$< 1\%$	$< 10\%$
100	26305	96.4%	97.6%	97.1%	98.9%
1000	2630	99.0%	99.2%	99.6%	100.0%
10000	263	99.6%	100.0%	100.0%	100.0%
100000	26	100.0%	100.0%	100.0%	100.0%

Table 5: *Comparison of* backward *delays with* $\mathbf{m} = \mathbf{\Delta} + at + g(t) + g_0$

once the variation is detected, for instance using reasonableness tests, the algorithm should be restarted to compute the new *clock skew compensation parameters*.

5 Conclusions

We showed that the continuous monitoring of some characteristics of the one-way delay (jitter included) without specific configuration of the measurement endpoints is viable.

We introduced an algorithm that performs such measurement with a time and space complexity that is logarithmic with the number of collected roundtrips. Its design is based on a probabilistic model, and the reliability is tested experimentally

The algorithm may give incorrect results during an initial "warmup" period, or as a consequence of an abrupt clock synchronization.

References

[1] Maximo Alves, Luigi Corsello, Daniel Karrenberg, Cagdas Ögüt, Mark Santcroos, Reinhard Sojka, Henk Uijterwaal, and Renè Wilhelm. New measurements with the RIPE NCC test traffic measurement setup. In *Passive and Active Measurement Workshop*, pages 66–75, Fort Collins, CO, 2002. 1

[2] Jean-Marc Berthaud. Time synchronization over networks using convex closures. *IEEE Transactions on Networking*, 8(2):265–277, April 2000. 1

[3] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry, Algorithms and Applications*.

Springer, 2nd edition, 2000. Riferimento al 'Graham scan' sul libro di Mimmo Luminati. 1, 2.1.2

[4] C. Demichelis and P. Chimento. IP packet delay variation for IPPM. Technical Report RFC3393, Network Working Group, February 2001. 1

[5] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, june 1972. 1, 2.1.2

[6] Warren Matthews and Les Cottrell. The PingER project: Active internet performance monitoring for the HENP community. *IEEE Communications Magazine*, 38(5):130–137, May 2000. 1

[7] Klaus Mochalski, Jorg Micheel, and Stephen Donnelly. Packet delay and loss at the Auckland Internet access path. In *Passive and Active Measurement Workshop*, pages 46–55, Fort Collins, CO, 2002. 1, 4

[8] Sue B. Moon, Paul Skelly, and Don Towsley. Estimation and removal of clock skew from network delay measurements. Technical Report 98-43, Department of Computer Science - University of Massachusetts at Amherst - USA, 1998. 1, 1, 3

[9] Vern Paxson. On calibrating measurements of packet transit times. In *Proceedings of ACM SIGMETRICS*, June 1998. (document), 1, 3, 4

[10] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995. analisi di traffico telnet/ftp/generico e dimostrazione che il modello Poisson non e' valido. Processi self similar, dipendenza long-range, bibliografia e commenti a lavori precedenti. 1, 4

[11] Franco Preparata and Michael Shamos. *Computational Geometry: An Introduction*, chapter 3. Da completare: trattazione del 'Graham scan'. 1

A Proof of lemma 1

$$\Delta(snd_i, rcv_i) = ts(rcv_i) - ts(snd_i) + g(rcv_i) - g(snd_i)$$

We rewrite:

$$\begin{aligned} g(rcv_i) - g(snd_i) &= (a_r t(rcv_i) - a_s t(snd_i)) + (g_r(0) - g_s(0)) \\ &= (a_r t(rcv_i) - a_s t(snd_i) + a_r t(snd_i) - a_r t(snd_i)) + (g_r(0) - g_s(0)) \\ &= (a_r - a_s) t(snd_i) + a_r (t(rcv_i) - t(snd_i)) + (g_r(0) - g_s(0)) \end{aligned}$$

that is simplified introducing the relative clock skew $a_{s,r} = a_r - a_s$, the relative clock offset $g_{s,r}(0) = g_r(0) - g_s(0)$, and $\Delta(snd_i, rcv_i) = t(rcv_i) - t(snd_i)$ from equation 2:

$$g(rcv_i) - g(snd_i) = a_{s,r} t(snd_i) + a_r \Delta(snd_i, rcv_i) + g_{s,r}(0)$$

The second order term $a_r \Delta(snd_i, rcv_i)$ can be deleted, and the right term can be substituted in the definition of Δ.

$$\Delta(snd_i, rcv_i) = ts(rcv_i) - ts(snd_i) + a_{s,r} t(snd_i) + g_{s,r}(0)$$

□

B Proof of lemma 2

We interpolate $a_{s,r}$:

$$a_{s,r} = -\frac{(ts(rcv_p) - ts(snd_p)) - (ts(rcv_q) - ts(snd_q))}{t(snd_q) - t(snd_p)}$$

$t(snd_p)$ and $t(snd_q)$ are real times, therefore not observable. But the two events occur on the same component, the sender, and we can use local clock readings introducing a minor imprecision:

$$\begin{aligned} t(snd_q) - t(snd_p) &= ts(snd_q) + g(snd_q) - (ts(snd_p) + g(snd_p)) \\ &= ts(snd_q) - ts(snd_p) + a_s t(snd_q) + g_s(0) - (a_s t(snd_p) + g_s(0)) \\ &= ts(snd_q) - ts(snd_p) + a_s (t(snd_q) - t(snd_p)) \end{aligned}$$

The second order term $a_s(t(snd_q) - t(snd_p))$ can be discarded, and we can rewrite the $a_{s,r}$ term as:

$$a_{s,r} = -\frac{(ts(rcv_p) - ts(snd_p)) - (ts(rcv_q) - ts(snd_q))}{ts(snd_q) - ts(snd_p)}$$

The $g_{s,r}(0)$ constant is derived from the Δ_p equation

$$g_{s,r}(0) = \Delta_p - ((ts(rcv_p) - ts(snd_p)) + a_{s,r} t(snd_p))$$

We substitute the two parameters that describe the relationships between the two clocks in formula 4:

$$\begin{array}{rcl}\Delta(snd_i, rcv_i) & = & (ts(rcv_i) - ts(snd_i)) - \\ & & \frac{t(snd_i)-t(snd_p)}{ts(snd_q)-ts(snd_p)}((ts(rcv_p) - ts(snd_p)) - (ts(rcv_q) - ts(snd_q))) - \\ & & (ts(rcv_p) - ts(snd_p)) + \\ & & \Delta_p\end{array}$$

Another $t(snd_i) - t(snd_p)$ term can be replaced with $ts(snd_i) - ts(snd_p)$, and the proof is concluded.
□

C Description and proof of the algorithm

The set I is defined as:

Definition 2 *Let* $\mathbf{d} = ((t_i, d_i), i \in [0, N-1])$ *be a sequence of observed delays where*
$t_i = ts(snd_i)$
$d_i = ts(rcv_i) - ts(snd_i)$
The set $I \subseteq [0, N-1]$ *is*

$$I \stackrel{\text{def}}{=} \{i \mid \exists a, c, \forall x \in [0, N-1], d_i - (a\ t_i + c) \leq d_x - (a\ t_x + c)\}$$

The following lemma explains the criteria used to select the elements of the set:

Lemma 3 *Let* $a_{r,s} = \frac{d_s - d_r}{t_s - t_r}$

$$I = \{i \mid \forall x \in [0, i-1], \forall y \in [i+1, N-1], a_{x,i} \leq a_{i,y}\}$$

Proof:

$$\begin{array}{rcl} i \in I & \leftrightarrow & \exists a, c, \forall x \in [0, N-1], d_i - (a\ t_i + c) \leq d_x - (a\ t_x + c) \\ & \leftrightarrow & \exists a, c, \forall x \in [0, N-1], d_i - d_x \leq (a\ t_i + c) - (a\ t_x + c) \\ & \leftrightarrow & \exists a, c, \forall x \in [0, N-1], d_i - d_x \leq a(t_i - t_x) \\ & \leftrightarrow & \exists a, \forall x \in [0, N-1], \left\{ \begin{array}{l} x < i, \frac{d_i - d_x}{t_i - t_x} \leq a \wedge \\ i < x, \frac{d_x - d_i}{t_x - t_i} \geq a \end{array} \right. \\ & \leftrightarrow & \exists a, \forall x \in [0, N-1], \left\{ \begin{array}{l} x < i, a_{x,i} \leq a \wedge \\ i < x, a_{i,x} \geq a \end{array} \right. \end{array}$$

$$
\begin{aligned}
& \leftrightarrow \quad \exists a, \begin{cases} \forall x \in [0, i-1], a_{x,i} \leq a \wedge \\ \forall x \in [i+1, N-1], a_{i,x} \geq a \end{cases} \\
& \leftrightarrow \quad \forall x \in [0, i-1], \forall y \in [i+1, N-1], a_{x,i} \leq a_{i,y} \\
I \quad & = \quad \{i \mid \forall x \in [0, i-1], \forall y \in [i+1, N-1], a_{x,i} \leq a_{i,y}\}
\end{aligned}
$$

Algorithm 1 selects the *candidate fastest packets*: the proof of its correctness can be extracted from the REQUIRE and ENSURE clauses.

Shortly, the **m** sequence contains observed delays, represented as a list of 2-ples (d_i, t_i) where d_i is the difference of local timestamps associated to the delay of the i-th packet, and t_i is the timestamp associated to the sending event of the same packet. The **I** list contains *candidate fastest packets*.

Each time a new observation (d_N, t_N) is available, the slope of the line that interpolates the new data and the point represented by the last element in the **I** list is compared with the slope of the line that interpolates the last two elements in the **I** list. If the latter dominates the former, the last element of **I** is deleted and the comparison is repeated with the previous element in **I**. Finally, (d_N, t_N) is inserted in the **I** list as the last element, and the algorithm terminates

The algorithm exhibits some interesting features:

- only delays corresponding to *candidate fastest packets* are of interest, and the processing of a new measurement consists at worst of a single scan of these data;

- although there is no apparent upper bound to the number of observations stored in **I**, the probabilistic model described in the paper shows that it grows with the logarithm of the number of packets exchanged in a monitoring session.