Clock Phase Change compensation using Graham Scan

Augusto Ciuffoletti*, Galina Antonova†

Abstract

The estimate of the Phase Change Rate of the local clock makes more efficient a clock synchronization protocol, like IEEE1588 Precision Time Protocol. We show how to obtain such parameter with a lightweight protocol that uses one-way messages from a Master to a Slave

1 Introduction

Emerging network applications for various markets require distributed clock synchronization. The new 1588 IEEE standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems [4], describes a Precision Time Protocol (PTP), designed for small scale networks. The protocol is split into two phases:

- one-way delay calculation using message exchange between one Master and one Slave (`Sync` and `Delay_req` messages), assuming symmetric delays;
- periodic Slave clock adjustments using obtained one-way delay and Sync messages from Master to this Slave.

For a wider applicability of the PTP, two facts should be taken into account:

- in larger networks delays are significantly variable, and not symmetric;
- Clock Phase Change Rate is characterized by a linear term (determined by crystal cut tolerance), which dominates non-linear terms (due to temperature and ageing).

We propose an algorithm based on the *Graham Scan* [2] (an algorithm mainly applied in computational geometry) for long term Clock Phase Change Rate compensation. We show that an effective implementation of such algorithm, integrated into a local clock synchronization servo, can bring desirable savings in cost (by using cheaper oscillators), utilized bandwidth (by less frequent messages) and power consumption (by using one-way messages), while tolerating Internet-like dispersions in network delays.

Theoretical foundations of Clock Phase Change Rate compensation and a description of Graham Scan algorithm are presented. Algorithm evaluation, based on modelling with Internet delays, and simulation results follow. Applicability to the IEEE 1588 is noted where applicable.

*University of Pisa - Italy. This work was supported by the National Institute of Nuclear Physics (INFN) - Italy, in the frame of the European CoreGRID project

†General Electric, Inc., Burnaby BC, CANADA

2 Theoretical foundations of Clock Phase Change Rate Compensation using Graham Scan

We divide the operation of a clock synchronization schema into two almost orthogonal activities: one is the retrieval of an accurate clock value (clock reading), the other is the compensation of Clock Phase Change Rate. While the former activity results in clocks showing the same time value at the same time, the latter focuses on the less restrictive, but equally important, property that clocks return the same measure for identical time intervals.

A sequence of clock readings alone may solve the problem of clock synchronization, although the frequency of this operation may be quite high: in order to obtain a 1 msec accurate clock from a quartz clock with a typical frequency tolerance of 1%, we would need a synchronization update every 0.1 second (i.e., 10 messages/sec).

However, since the Clock Phase Change is characterized by a term that is constant in time (henceforth referred as Clock Phase Change Rate), we might obtain a highly accurate clock by compensating such linear term: residual Clock Phase Change would be due to thermal variations (variable in a range below 0.01% in a reasonable temperature range, according to data-sheets from `http://www.rfglobalnet.com`) or ageing (even lower). As a result, synchronization update every 10 seconds (i.e., 0.1 messages/sec) would be enough to obtain a 1 msec accurate clock without using a temperature-compensated oscillator.

To estimate the linear term of the Phase Change of a clock with respect to another remote clock, we study the use of a *convex hull* algorithm [2]. A similar methodology is used by [6] to measure message latencies.

This methodology is of special interest for mobile devices, since it is based on one-way messages: therefore the remote mobile may save energy by just listening to Phase Change compensation messages, and exchange infrequent synchronization messages to correct compensation inaccuracies. Another favorable property is that such methodology does not require timely message generation: therefore the source can simply piggyback the phase correction activity onto another application that requires message exchange. This approach also could be easily integrated into the IEEE 1588 architecture, since it is based on messages similar to those already provided for PTP.

We carry out our investigation using a simple simulator. The first step consists of generating data that reproduce a system with given Clock Phase Change. These data are fed to a tester program that calls the algorithm, computing Clock Phase Change estimates. These estimates are checked against the parameters passed to the simulator.

2.1 Clock Phase Change Rate estimate

We want to attach *timestamps* to *events* occurring in *hosts*, each equipped with a *physical clock* device, producing *timestamps* for local events. These timestamps are related to real time through a *clock function*: for an event e that occurs at real time $t(e)$, an associated timestamp is denoted as $ts(e)$. The *clock phase difference* $g(e)$ at time $t(e)$ corresponds to the difference between the real time and the timestamp:

$$ts(e) = t(e) - g(e)$$

.

Assuming that Clock Phase Change is linear in time, which is consistent with our previous discussion, we can approximate the $g(e)$ term with an equation linear in time, and:

Definition 1

$$ts(e) = t(e) - (a * t(e) + g(0)).$$

.

We are interested in two series of events occurring on two hosts, a *Master* and a *Slave*. One series consists of *send events*, and the other is made of *receive events*, all related to a uni-directional *message* exchange activity from the Master to the Slave. Each message contains a timestamp of the send event, obtained using Master's clock function.

The *timestamp difference* Δ_i, which corresponds to the arithmetic difference between the timestamp recorded by the Master in the i-th message and the timestamp registered by the Slave when it receives the message, is defined as follows:

Definition 2

$$\begin{aligned}
\Delta_i &= ts(rcv_i) - ts(snd_i) \\
&= t(rcv_i) - t(snd_i) - (a_s * t(rcv_i) - a_m * t(snd_i)) - (g_s(0) - g_m(0)) \\
&\simeq t(rcv_i) - t(snd_i)) - a_{s,m} * t(snd_i)) - g_{s,m}(0)
\end{aligned}$$

,

where $a_{s,m}$, the *relative Clock Phase Change Rate*, and $g_{s,m}(0)$, the *relative clock offset* at origin, are two constants. The expression is simplified considering that $a_s * (t(rcv_i) - t(snd_i))$, the Clock Phase Change during message delivery, is negligible.

Note that

- $ts(rcv_i) - ts(snd_i)$ term is known, and corresponds to the timestamp difference,
- $a_{s,r}$ and $g_{s,r}(0)$ are unknown constants that describe the relationships between two clocks,
- $t(snd_i)$ is the time of the send operation, which is in principle not observable,
- $t(rcv_i) - t(snd_i)$ is the message latency, in principle not observable.

2.2 Approximation of the relative Clock Phase Change Rate

At the core of the *convex hull* methodology is the fact that $a_{s,r}$ can be found if two messages p and q with identical message latencies are selected. At first, let us call the corresponding timestamp differences Δ_p and Δ_q. Solving the system with two equations we obtain:

$$\begin{aligned}
\Delta_p - \Delta_q &= a_{s,m} * (t(snd_q) - t(snd_p)) \\
&= a_{s,m} * (ts(snd_q) - ts(snd_p) + a_m(t(snd_q) - t(snd_p))) \\
&\simeq a_{s,m} * (ts(snd_q) - ts(snd_p))
\end{aligned}$$

.

We reintroduce Definition 1 and discard the second order term $a_{s,m} * a_m * (t(snd_q) - t(snd_p))$.

Finally, under the hypotheses of a linear Clock Phase Change and identical latencies for messages p and q, a relative Clock Phase Change would be as follows:

$$a_{s,m} = \frac{(ts(snd_q) - ts(snd_p))}{\Delta_p - \Delta_q}$$

This equation contains only terms known to a Slave: Master's timestamps for messages p and q, and corresponding timestamp differences: we will call an *observation* a pair $(ts(snd_i), \Delta_i)$, for messages progressively indexed with i.

To use the above equation, we devise a way to select two messages with (almost) identical latencies, recording only a small number of observations.

2.3 Selecting messages with close latencies

In practice, we cannot find two messages with exactly identical latencies: however, we prove that two messages that experienced *close* latencies can be found. This imprecision affects the estimate of the Clock Phase Change Rate $a_{s,r}$: let ϵ be the difference between the two latencies, $a_{s,r}$ will fall in an interval whose width is

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

We will call this value the *residual Clock Phase Change*.

In order to minimize this value, we should minimize ϵ and the distance between two observations. An *ad hoc* heuristic helps solving this difficult task: this means that a result appropriate for the Internet-like environment we envision might be unsuccessful or sub-optimal in another environment.

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

- we select a small subset of the observations that contains two observations with minimal latencies;
- we select two successive observations with minimal latencies, separated by the largest timing gap.

A relevant feature of this heuristic is that it can be implemented in logarithmic time and space: therefore, the cost of a new observation will not increase linearly with time, but will be a logarithm of the number of observations made. The number of recorded observations also exhibits the same tendency.

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

We start from considering that a distribution of message latencies in the Internet, as observed by F. Cristian [1], is characterized by an asymmetric bell distribution, with a median located near the minimum; we'll discuss this distribution in section 3.1.

We note that by selecting messages with minimum latencies, their latency is bound to converge rapidly to the minimum, which is determined by minimal intervening buffering along the shortest route. This satisfies the basic requirement of finding two messages with close latency.

In order to find a subset that contains the two messages with minimal latencies in the sequence of observations, we introduce a (low) convex hull concept: given a set A of (x, y) points, the low convex hull corresponds to a subset H of A that contains vertices of the low end of a convex polygon that topologically contains A (see Figure 1). A well-known theorem states the following:

Theorem 1 *Given a generic line $y = ax + b$ below all points in A, the two points of A with minimal distance from that line are necessarily adjacent points in H.*

In time versus timestamp difference plot (see Figure 4), each point corresponds to an observation. The unknown *clock distance equation* (derived from definition 1 by replacing a and g with relative values) is a line below all points: the distance of an observation from that line is necessarily positive, since it corresponds to unknown message latency.

Using the above theorem we conclude that, for any *real* clock distance equation, two messages with minimal latencies necessarily correspond to adjacent points in the convex hull.

This concludes the first step in our heuristic, limiting our search for two observations with minimal latencies to those in the convex hull.

The *Graham Scan* algorithm is used to compute the lower convex hull. This algorithm iteratively processes points with growing abscissa: each iteration uses a new point and a previously computed lower convex hull, by performing a single scan of these points. Therefore both time and space complexities for processing a new observation correspond to the number of points in the convex hull. In theory this number tends to grow logarithmically with the number of points in A: our experiments show that the number of points in the convex hull tends to stabilize around twenty.

As a result, the cost of re-computing the convex hull when a new message arrives corresponds to scanning a bi-dimensional array of less than twenty rows and performing simple arithmetic operations for each row, as shown in the Table 1.

The next step in our heuristic is selecting observations with minimal latencies in the convex hull: we select two adjacent points, separated by the longest time gap. The rationale behind this simple heuristic is that such selection can minimize estimated latencies of other observations, which is consistent with the observed distribution of message latencies. We stress that this heuristic is extremely simplistic, and further investigation would be beneficial.

In the Table 1 we summarize the algorithm to update a list, representing the convex hull, each time a new message is received: the receiver decides which observation to remove, and adds the last observation to the list. When appropriate, an estimate of the Clock Phase Change Rate is recomputed (not shown in table 1).

3 Algorithm evaluation

In order to check the validity of the Clock Phase Change estimate algorithm, we need a sequence of observations, each composed of a timestamp and a corresponding times-

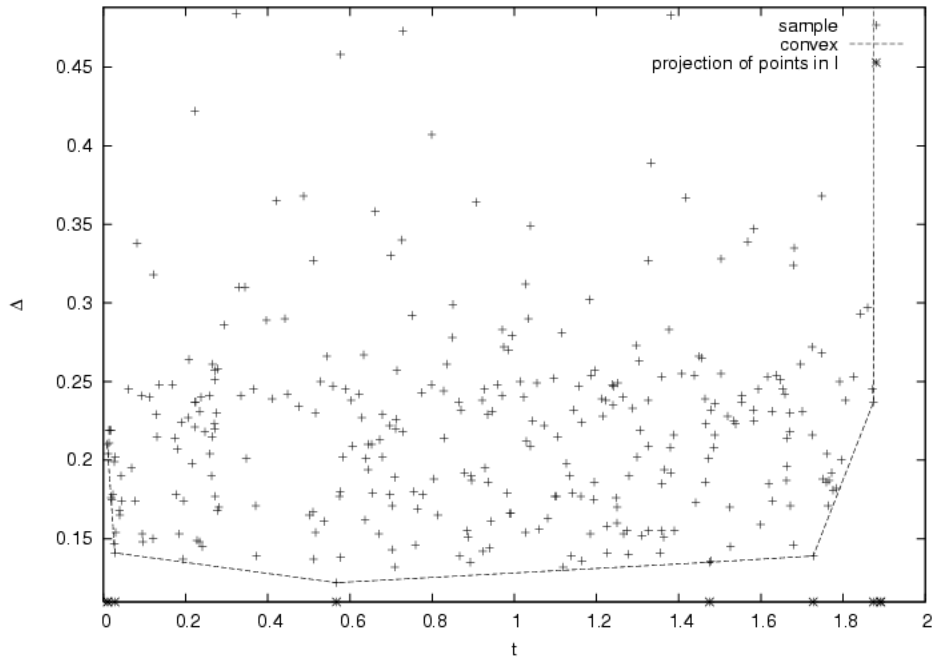

Figure 1: A X-Y representation of observations: X is the send timestamp of a message in seconds, Y is the timestamp difference in milliseconds. The dashed line corresponds to the lower convex hull. t values marked with an `x` indicate observations collected in set I by Graham algorithm (see algorithm 1)

```
comment: A observation is a (timestamp difference, send timestamp) pair
type observation = record(d, t : real);
comment: X contains a new observation after a message is received
X :;
comment: I contains the observations in the convex hull
I : arrayof observation;
p : integer;
comment: K computes the slope of the line passing from a and b
proc K(a, b : observation) ≡ (b.d − a.d)/(b.t − a.t).
while read X
        do
        if (#I > 2)
           p = #I − 1;
           while (p > 1 ∧ K(I[p − 1], I[p]) ≥ K(I[p], X))
                  do
                  pop I;
                  p = p − 1;
           end
        end
        push I, X
end
```

Table 1: *Processing of a new observation with Graham scan*

tamp difference. Each pair represents the observable characteristics of the message delivery.

While the generation of a timestamp is trivial, the generation of timestamp differences requires further study and an introduction to a reference model. The generation of a timestamp difference is divided into two steps, each discussed separately: generation of random message latencies and altering message latencies using the clock difference equation. Parameters of this equation are a Clock Phase Change Rate and a periodic thermal variation.

3.1 Generating random message latencies

We do not aim at a model that, given a network layout, returns a sequence of random latencies with statistical properties similar to those found in the real system. Here we simply want to have a sequence of latencies, which are reasonably similar to those found in real networks: in essence, we will focus on distribution shape and self-similarity.

We assume that a message traverses through a number of routers and/or switches. Although these devices are capable of queuing messages, the probability that the message is not queued is high (see [3]): networks are designed so that queues are used only during short and infrequent traffic peaks. The number of *queuing events* has a binomial distribution, like the number of successes in a sequence of Bernoulli trials. The p parameter depends on traffic, while N reflects the number of intermediate routers that may delay the message. Since each queuing introduces a variable delay, we simulate a total queuing delay using a Weibull random numbers generator.

The characteristics of this distribution vary in time: for networks with a lot of traffic, the probability p of a single packet being queued raises. In order to keep our generator simple, we did not simulate individual processes or packets, but simply the overall effect queuing has on our messages. We assume that at any time, with a given probability, the probability of finding non-empty queues may change. This probability (the p parameter in the binomial distribution) has a bell distribution, which we implemented using a Weibull generator with $\alpha > 1$. This mechanism contributes to the *self-similarity* of the produced sample.

Finally, apart from the time spent in queues, the message spends some time in transfers from router to router, and is processed by edge hosts and routers. A contribution for this activity in message latency is represented as exponentially distributed, shifted to a value, corresponding to the minimum transfer time. Since processing is exposed to random events due to process synchronization, an exponential distribution seems appropriate.

The final result of our message latency generator is described in Figure 2. We set generator's parameters to be similar to those of the Auckland traces (see [5]), whose latency distribution is plotted in Figure 3. The aspect relevant to our task is the distribution around minimum value: both samples show a 20% percentile at about 0.2 msecs. The *Hurst parameter*, which measures self-similarity, is around 0.7 for the generated sample, as for our reference Auckland trace.

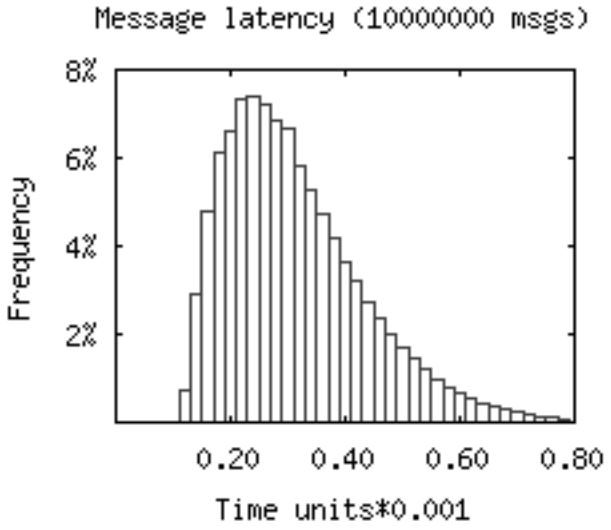

Figure 2: *Distribution of message latencies obtained from generator.*

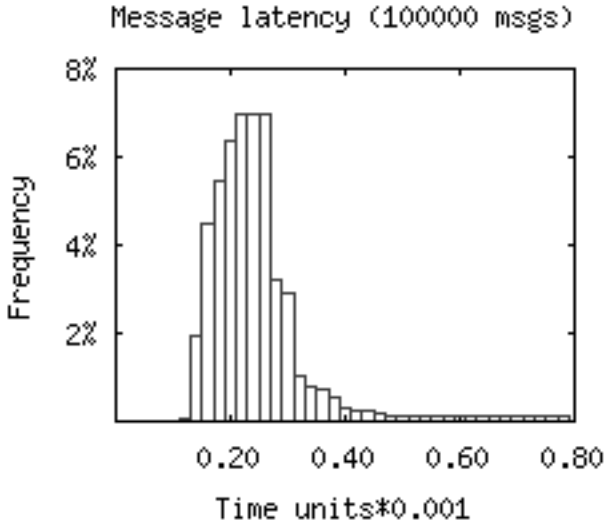

Figure 3: *Distribution of message latencies from Auckland trace. Values over* 0.8*msec have been omitted.*

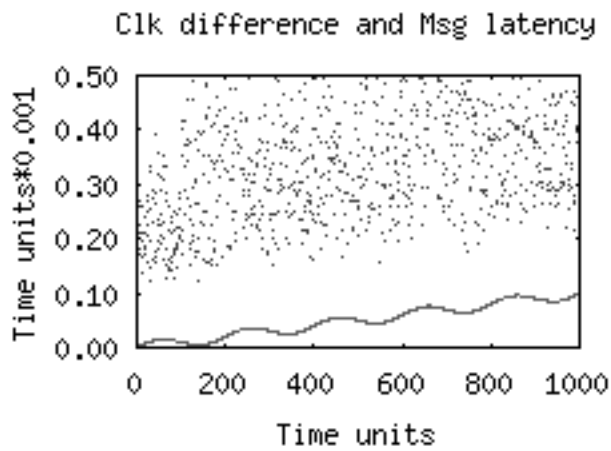

Figure 4: *Clock and timestamp differences obtained from the simulator.*

3.2 Mapping message latencies to clock differences

The message latency undergoes two simple transformations that model physical clock inaccuracy.

The first source of inaccuracy is the Clock Phase Change Rate that the algorithm under test should measure. Its effect is the progressive increment of the clock differences, which is added to the difference due to message latency.

$$(ts(rcv_i) - ts(snd_i)) = Latency_i + a_(s,r)t(snd_i)$$

We made the hypotheses that at time 0 the two clocks are perfectly synchronized, which does not affect the significance of our test.

In order to model temperature driven phenomena, we added a phase oscillation: a periodical shape seems appropriate, since temperature is always exposed to some sort of discontinuous regulation. This brings us to the following:

$$(ts(rcv_i) - ts(snd_i)) = Latency_i + \left(a_{s,r} + 3.6\ P_s A_s sin\left(\frac{2\pi\ t(e)}{P_s}\right)\right) t(e)$$

where P_s is the period of the Clock Phase Change Rate, in time units, and A_s is its amplitude, in thousandths of a time unit.

The trace in Figure 4 shows a generated sequence of clock differences. Sparse dots are the clock differences, while the continuous line below is the clock difference equation.

3.3 Evaluating the Clock Phase Change Rate estimate algorithm

As a last step, we feed the generated sequence of timestamp differences to the algorithm: since we know the correct value of the Clock Phase Change Rate, we can check its output.

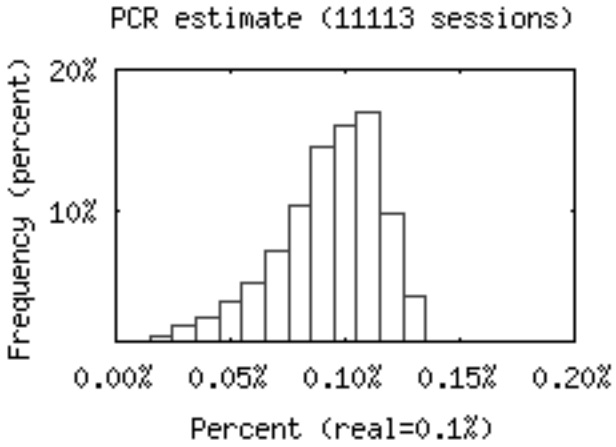

Figure 5: *Distribution of clock Phase Change Rate estimate at stabilization.*

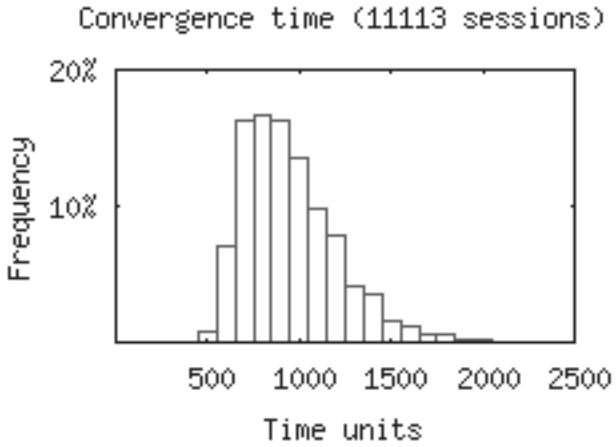

Figure 6: *Distribution of time to reach stabilization.*

Figure 5 contains a histogram, summarizing the result: the x-coordinate reflects the estimated Clock Phase Change Rate when it appears to be sufficiently stable. In most cases it falls in the interval $[0.5, 1.5]$ parts per thousand. The clock differences generator was set to introduce a Clock Phase Change Rate of 1 part per thousand.

Figure 6 represents another histogram for the time needed to reach stabilization, which may be reasonably bound within the range [500,1500] time units. Considering the case of one message per second, the estimated time to converge is in the tens of minutes.

4 Conclusions

For its wider applicability the IEEE 1588 standard may benefit from considering message delay asymmetry and Clock Phase Change Rate compensation. Such compensa-

tion can be performed using Graham Scan algorithm, which allows to tolerate Internet-like dispersions in message latency by using an *almost linear* model of the Clock Phase Change Rate.

Simulation-based evaluation of this approach shows estimates of the Clock Phase Change Rate within $10^{-}4$, which is two orders of magnitude better than the rate of a typical quartz clock. Time to converge varies from 10 to 30 minutes for one synchronization update message per second.

Besides expanding PTP applicability to larger networks, a Clock Phase Change Rate compensation algorithm may bring desirable savings in cost (by using cheaper oscillators), utilized bandwidth (by less frequent messages) and power consumption (by using one-way messages).

References

[1] F. Cristian. Probabilistic clock synchronization. *Distributed computing*, 1989.

[2] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 2nd edition, 2000.

[3] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, 1996.

[4] IEEE Std 1588-2002. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, November 2002.

[5] K. Mochalski, J. Micheel, and S. Donnelly. Packet delay and loss at the Auckland Internet access path. In *Passive and Active Measurement Workshop*, pages 46–55, Fort Collins, CO, 2002.

[6] S. B. Moon, P. Skelly, and D. 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.