2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

TICSync: Knowing When Things Happened

Alastair Harrison and Paul Newman

Abstract— **Modern robotic systems are composed of many distributed processes sharing a common communications infrastructure. High bandwidth sensor data is often collected on one computer and served to many consumers. It is vital that every device on the network agrees on how time is measured. If not, sensor data may be at best inconsistent and at worst useless. Typical clocks in consumer grade PCs are highly inaccurate and temperature sensitive. We argue that traditional approaches to clock synchronization, such as the use of NTP are inappropriate in the robotics context. We present an extremely efficient algorithm for learning the mapping between distributed clocks, which typically achieves better than millisecond accuracy within just a few seconds. We also give a probabilistic analysis providing an upper-bound error estimate.**

I. Introduction

This paper is about precise time stamping of events on a robot or distributed system containing multiple clocks. We introduce the TICSync algorithm, for efficiently learning the mapping between distributed hardware clocks to allow precise, synchronized time stamping. TICSync is an incremental algorithm, with O(1) update cost. Unusually, it provides probabilistic bounds on its accuracy. TICSync is not just about drastically reducing timing error, it is about knowing how good timing is.

Our motivation for developing TICSync is simple - we must know the time at which a measurement is valid if we are to interpret it fully and fairly. Despite this being a truism, little attention is given to this issue in robotics literature. While there is a vast corpus of work on how to handle uncertainty in measurements (range sensor noise, wheel slip, shot CCD noise etc) less time has been spent on considering (and reducing) the noise in measurement time stamps.

Multi-processor systems are now ubiquitous; pretty much every camera, laser or IMU we connect to our robots contains a free running microprocessor with an independent clock. We should worry about the way in which time varying transport delays between sensors (sources) and host processors (sinks) impinge on the measurements. If our stated goal is operational longevity then we must care about (and mitigate) how clocks drift relative to one another. The TICSync algorithm addresses these needs in a lightweight and principled way.

A. How bad can clocks be?

The clocks in consumer grade computer equipment are often of very low quality, being particularly susceptible to changes in temperature. To illustrate this, Figure 1 shows large clock offset fluctuations in a SICK LMS151 laser range sensor, due to only modest temperature variations.

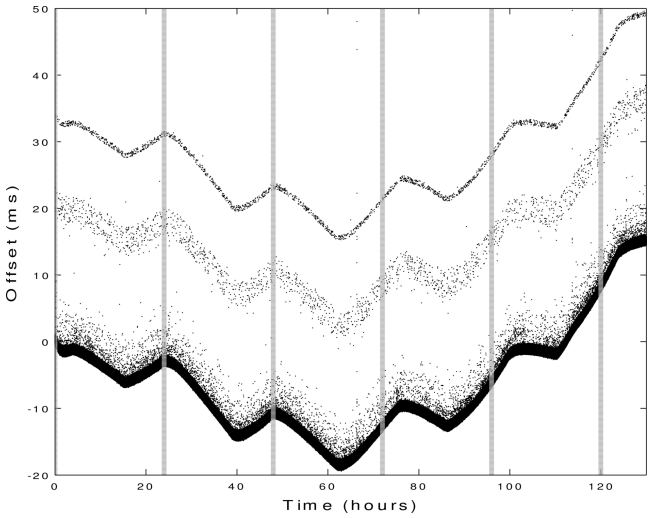


Fig. 1: As an example of the poor quality clocks in the equipment commonly used in robotic systems, we performed a 6 day long experiment, measuring clock offsets between a SICK LMS151 laser range scanner and a standard desktop computer in an office environment. The vertical lines represent 24 hour periods. At the end of the experiment, the laser had lost 92 seconds compared to GMT. To produce this figure, we removed the first order skew from the data, to leave only second order effects. The remaining fluctuations correlate well with temperature measurements from the same period, and show drift effects causing swings of over 30ms in offset. Note that the two upper bands in the offset data are caused by task scheduling effects delaying time stamping on the desktop computer.

A common approach to clock synchronization is to run an NTP (Network Time Protocol) [1] server on one computer and have the others adjust their clocks to it. The clocks are brought slowly into alignment by varying their frequencies by small amounts. NTP is an excellent way of achieving long term synchronization, but we argue that it is undesirable in a robotics context because it can take many hours to synchronize clocks via NTP, and frequency adjustments can cause inconsistencies in time stamps.

We call our algorithm *Time stamp based Incremental Clock Synchronization*, or *TICSync* for short. It is capable of rapidly learning the mapping between clocks (e.g after a shutdown) and able to adjust to clock upsets such as frequency changes which may be exhibited by clocks under normal varying temperature conditions or under the control of NTP. In other words, we have a system which maintains synchronization *despite* the use of NTP. Our motivation here is a light touch. We don't want to demand that NTP is not run, as that seems rude.

Alastair Harrison and Paul Newman are with the Mobile Robotics Research Group, Department of Engineering Science, University of Oxford, OX1 3PJ, England {arh,pnewman}@robots.ox.ac.uk

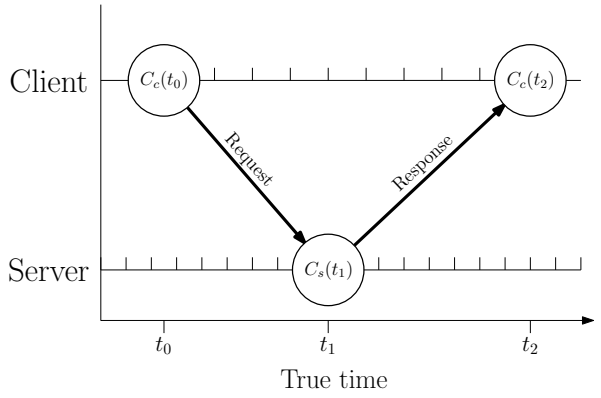


Fig. 2: Time stamping mechanism between two computers. The client and time server clocks are running at different frequencies, represented by the tick marks. When either computer sends or receives a packet, it adds a local time stamp. Possession of all three local time stamps allows a bounded offset estimate to be obtained. Note that propagation delays may be asymmetric.

II. Synchronization Mechanism

Consider two computers communicating over a network: a client and a time server. Time on the server is denoted as $T_s = C_s(t)$ and the client clock as $T_c = C_c(t)$, with t being the universal time. The *offset* between the two clocks may be positive or negative and is defined as

$$\tau(t) = C_c(t) - C_s(t) \tag{1}$$

Similarly, clock *skew* is defined as $C_c'(t) - C_s'(t)$ and *drift* as $C_c''(t) - C_s''(t)$.

In a simple synchronization experiment, the client prepares a data packet, time stamps it with the current local (client) time and sends it to the time server. If the clocks of the the two computers are perfectly synchronized then the time server will observe a difference between the time of transmission and the time of receipt equal to the network propagation delay. If the clocks are *consistent*[1] then the time server will observe a difference equal to the sum of the offset and the network propagation delay. Crucially, the time server is *unable* to recover the offset between the two clocks without having knowledge of the network delay.

The offset may be estimated using a two-way timing mechanism. Consider Figure 2 which shows a single packet communication in each direction with local time stamps being stored at each *send* or *receive* event. We will refer to a packet from the client to the time server as a *request* and a packet from the time server to the client as a *response*.

The packet transport delays may be considered as random variables, with distribution depending on network load, trip distance, and CPU load at the client and time server. It can not even be assumed that the delays are symmetric. Indeed, if the time server is busy, it may not process and time stamp the request packets immediately, meaning that the outbound request journey often takes longer than the inbound response journey.

Because the three events in Figure 2 must occur in strict order, and clocks are strictly monotonic, it is possible to obtain upper and lower bounds on the true offset. The lower bound is given by $\tau_{LB}(t_1) = C_c(t_0) - C_s(t_1)$, since

$$\tau(t_1) = C_c(t_1) - C_s(t_1) \tag{2}$$
$$> C_c(t_0) - C_s(t_1) \tag{3}$$

A similar argument can be used to show that the upper bound is given by $\tau_{UB}(t_1) = C_c(t_2) - C_s(t_1)$.

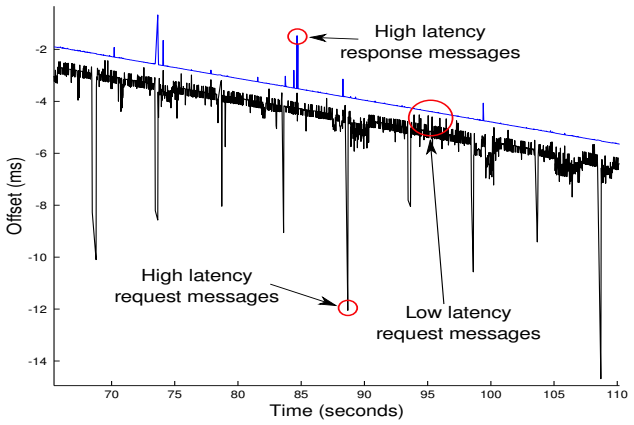


Fig. 3: Typical offset measurements for a moderately loaded time server. The bottom line shows offsets measured from the request packet, and the bottom line shows offsets measured from the response packet. The two lines bound the true offset. Notice that request packets typically have higher latency than responses, yet there are still a few low latency request packets.

Figure 3 shows typical upper and lower bound offset values measured by a client, with a moderately loaded network. It can be seen that the upper and lower bound lines form a corridor in which the true offset value must lie. At a particular instant, the total Round Trip Time (RTT) is given by the difference between the upper bound and the lower bound. As network load increases, packet journey times will increase, and the bounds will diverge, corresponding to larger RTTs. If δ is the minimum one-way transport delay, then on a completely unloaded system, the RTT will reach a minimum value, 2δ, corresponding to the shortest possible time for an exchange between the client and server. The upper and lower lines forming the corridor will *never* intersect each other, thus

$$\tau_{UB}(t_i) - \tau_{LB}(t_i) \geq 2\delta \tag{4}$$

When there is a lot of traffic passing through the server, or the CPU is under high load, it may take longer than usual for the server to get round to processing a client request message. This explains the fact that in Figure 3 the lower bound line is noisier than the upper bound line. It can also be seen that occasionally a low latency message will slip through. These low latency messages can provide useful information, even if the total round trip time is large.

Over periods of the order of tens of minutes, the effect of drift between clocks is typically small. It is reasonable to assume a first order (constant-skew) model for both clocks, so that given a recent time datum T_0, the offset $\tau(t)$ (Equation 1) is approximated by

$$\tau(t) \approx C_c'(T_0)t + C_c(T_0) - C_s'(T_0)t - C_s(T_0) \tag{5}$$
$$= \alpha t + \beta \tag{6}$$

[1]Consistent clocks have equal frequency (no relative skew) but a constant offset

where $\alpha = C'_c(T_0) - C'_s(T_0)$ is the skew between clocks and $\beta = C_c(T_0) - C_s(T_0)$ is the relative offset at T_0.

Here α and β are parameters that are typically learned by the client. Once they have been determined, it is a trivial matter to apply equation 6 to map between client and server times.

III. Previous Work

The problem of synchronizing two clocks comes down to using the bounds corridor to estimate the skew and offset between the clocks, though many approaches consider only messages sent in one direction, For those algorithms the offset estimate will always be biased by the minimum network delay, δ. Some authors have attempted to learn the unknown offset by comparing outputs from different sensors [2].

Early techniques [3], [4] made clock adjustments based on information from only the most recent exchange and would not estimate skew. The recent IEEE1588 Precision Time Protocol loop to provide accurate time stamps at a hardware level.

Filtering approaches can obtain more accurate results than are available from point estimates. Paxson [5] takes an off-line multi-step filtering approach using bi-directional time stamped messages and a series of heuristics. Veitch et al. [6] use highly accurate driver-level time-stamping to minimize packet timing error, then average offset measurements with low Round Trip Times (RTTs) within a sliding window. Aweya et al. [7] use a linear regression approach to finding the skew.

Noh et al. [8] derive Maximum Likelihood Estimators (MLEs) for offset and skew under the assumption of both Gaussian and Exponential delay distributions. The algorithms are not suitable for online use, because they are computationally intensive and require all historical data to be stored.

Moon et al. [9] consider only one-way timing information and ensure that the bounds are satisfied by using a linear programming technique to fit a single line, pushed up against the data points. Zhang et al. [10] showed that the Moon objective function could be minimized using a convex hull approach in linear time. It should be no surprise that the convex hull plays a part, since any straight line which is pushed up against the data cannot touch any point not on the convex hull. The algorithm is very effective and lends itself to an incremental implementation. To reiterate, the methods of Moon et al. and Zhang et al. operate on one-way timing data only. The next section goes on to propose an efficient algorithm which operates on two-way timing data.

IV. Algorithm

Sirdey and Maurice [11] proposed a linear programming approach to estimating the skew and offset between clocks, using two-way timing data. Their aim was to find a pair of maximally separated parallel lines which pass between the upper and lower bound offset measurements. If the upper bound measurements are given by $D = \{d_0 \,..\, d_{N-1}\}$ and the lower bound measurements by $Q = \{q_0 \,..\, q_{M-1}\}$ then

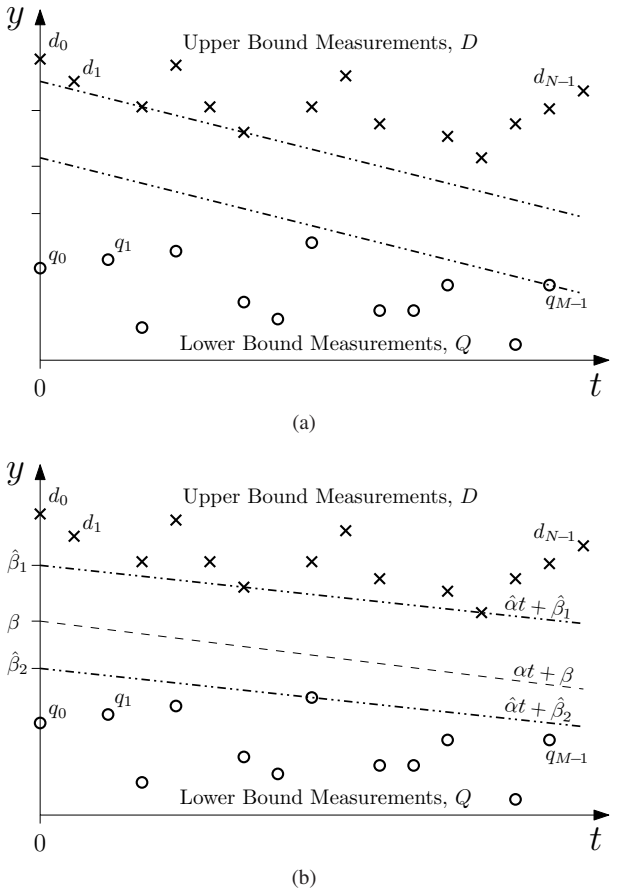


Fig. 4: There are many pairs of parallel lines which pass through the bounds corridor whilst touching at least one upper and lower offset measurement respectively. Figure 4(a) shows two such lines, pushed as far apart as possible for the given slope. The objective function in Equation 7 finds the slope angle $\hat{\alpha}$ such that the two lines ($\hat{\alpha}t + \hat{\beta}_1$ and $\hat{\alpha}t + \hat{\beta}_2$) are maximally separated, as shown in Figure 4(b). The line $\alpha t + \beta$ shows the correct mapping between the two clocks from which the offset measurements are derived. As more measurements are gathered, $\hat{\alpha}$ converges to α.

the lines of maximum separation are found by maximizing the objective function,

$$\begin{aligned} \max_{\hat{\alpha},\hat{\beta}_1,\hat{\beta}_2} \quad & \hat{\beta}_1 - \hat{\beta}_2 \\ \text{s.t.} \quad & \hat{\alpha}t_i + \hat{\beta}_1 \le d_i, \qquad i = 0\,..\,N-1 \\ & \hat{\alpha}t'_j + \hat{\beta}_2 \ge q_j, \qquad j = 0\,..\,M-1 \end{aligned} \tag{7}$$

Figure 4(b) shows an example, with the lines of maximum separation labelled as $\hat{\alpha}t + \hat{\beta}_1$ and $\hat{\alpha}t + \hat{\beta}_2$. Given that the data is expected to form a linear corridor, and assuming that message delays are symmetric, one would use a line $\hat{\alpha}t + (\hat{\beta}_1 + \hat{\beta}_2)/2$ as the final mapping estimate.

Sirdey and Maurice proposed a batch approach to solving the objective function. We contribute an efficient incremental approach to solving the same objective function, based on the use of convex hulls.

A. A Convex-Hull based Algorithm

We begin with a simple observation about the batch approach of Sirdey and Maurice, who use a rapid 50Hz sample rate in their synchronization experiments. They suggest running the algorithm over 10 minute windows of data,

over which the error due to clock drift is negligible. For a 10 minute run, the batch linear-programming algorithm must process 60,000 constraints; many of them far from the feasible region and therefore never active.

The algorithm's efficiency would be greatly improved by using as constraints only those points which appear as vertices on their respective convex hulls. In our own experiments, convex hulls applied to linear timing data rarely exceed 20 segments, independent of the number of samples; the segments just tend to get longer. The hulls could be maintained incrementally and then batch linear programming updates would be cheap.

We can demonstrate how the algorithm can be further improved such that the whole optimization may be performed incrementally as new measurements come in. Space constraints preclude us from providing the full proof, but the key intuition is that the lines of maximum separation pass through the points of closest vertical distance between upper and lower hulls. If the upper and lower hulls are denoted as Ω_u and Ω_l respectively then we use the notation $\mathcal{D}(\Omega_u, \Omega_l)$ to represent a function returning the closest vertical distance between them.

When a new packet delay measurement arrives, it is added to the appropriate convex hull, Ω_u or Ω_l. The hull for a time ordered set of points can be computed with the simple $O(N)$ package wrapping algorithm given by Zhang et al. [10]. Thus the cost of adding each new point is $O(1)$ amortized. Then $\mathcal{D}(\Omega_u, \Omega_l)$ is used to find the instant, t_* of minimum vertical distance between the two hulls. The slope of the hulls at t_* gives the slope of the two Maximum Separation lines. Finally the intersects $\hat{\beta}_1$ and $\hat{\beta}_2$ are computed such that they contact their respective hulls at t_*.

Note that $\mathcal{D}(\Omega_u, \Omega_l)$ is efficient. The point of minimum vertical distance between two hulls will *only* change when a new measurement falls between the existing lines of Maximum Separation and will never move backwards in time. The procedure is thus:

1) Add new measurements to Ω_u and Ω_l ($O(1)$)
2) If either measurement lies between $\hat{\alpha}t+\hat{\beta}_1$ and $\hat{\alpha}t+\hat{\beta}_2$ then
   a) Call $\mathcal{D}(\Omega_u, \Omega_l)$ to search forward in time from the previous point of minimum hull distance. (Amortized $O(1)$)
   b) Recompute the lines of maximum separation to pass through the new closest hull vertices or segments. ($O(1)$)

Usually there is no work to do beyond adding a new hull segment, but occasionally the maximum separation lines will need to be recomputed. As the algorithm converges, the regularity with which a datum falls between the Maximum Separation Lines will decrease.

V. Estimator Convergence Properties

This section shows how to obtain a probabilistic bound for the TICSync estimator. As before, let $D = \{d_0 \,..\, d_{N-1}\}$ be the set of upper bound offset measurements and $Q = \{q_0 \,..\, q_{M-1}\}$ be the set of lower bound offset measurements. If the data are generated from a pair of clocks with a linear relationship defined by $\tau(t) = \alpha t + \beta$ then each point can be thought of as being produced by a probabilistic process whereby

$$d_i = \alpha t_i + \beta + \delta + w_i \tag{8}$$

$$q_j = \alpha t_j + \beta - \delta - v_j \tag{9}$$

Here δ is the minimum one-way network delay and $w_i, v_j \sim \mathcal{W}$ are *positive* random variables representing unknown packet delays. The form of the distribution $\mathcal{W}$ is discussed later.

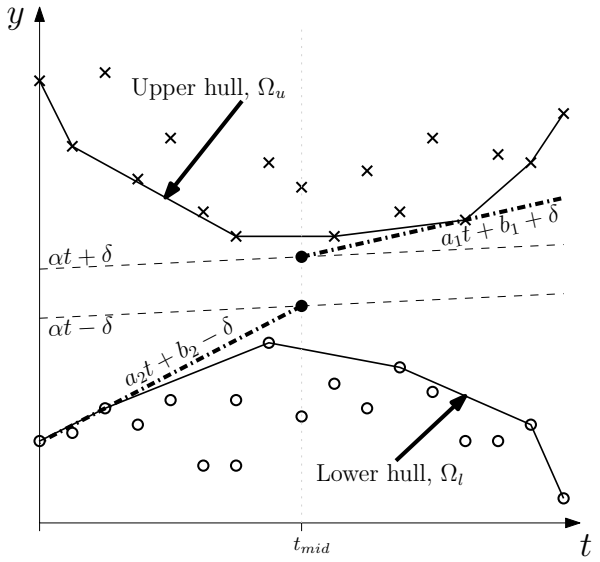


Fig. 5: The estimator, $\alpha^{\oplus}$ is defined as the steepest of two lines anchored at the halfway time, t_{mid}. One is a line crossing $\alpha t + \delta$ at t_{mid} and lying at a tangent to the right half of the top hull, and the other a line crossing $\alpha t - \delta$ at t_{mid} and lying at a tangent to the left half of the bottom hull. Represented here as a thick dash-dotted lines, one or the other is guaranteed to be at least as steep as the TICSync slope estimate, $\hat{\alpha}$ at all times.

We now investigate a hypothetical estimator $\alpha^{\oplus}$, which is guaranteed to give estimates at least as steep as $\hat{\alpha}$, the solution to the objective function in Equation 7. We can use its convergence properties to provide a bound on the convergence properties of $\hat{\alpha}$. For the purposes of the analysis, let us assume a constant sample rate, with period T. We also define the halfway time,

$$t_{mid} = \frac{N-1}{2}T \tag{10}$$

and the midpoint sample,

$$m = \left\lceil \frac{N-1}{2} \right\rceil \tag{11}$$

Consider Figure 5, which shows upper and lower bound packet delay measurements D and Q along with their respective hulls. $a_1t + b_1 + \delta$ is the steepest line passing through the point $(t_{mid}, \alpha t_{mid} + \delta)$ such that $d_i \geq a_1t + b_1 + \delta$ for all i. Likewise, $a_2t + b_2 - \delta$ is the steepest line passing through the point $(t_{mid}, \alpha t_{mid} - \delta)$ such that all $q_j \leq a_2t + b_2 - \delta$. If we take the steepest of these two lines and denote it as

$\alpha^{\oplus}t + \beta^{\oplus} \pm \delta$ then it can be shown that

$$\alpha^{\oplus} = \max(a_1, a_2) \tag{12}$$
$$\geq \hat{\alpha} \tag{13}$$

This is a key point. *No matter* what slope the skew estimate $\hat{\alpha}$ takes, $\alpha^{\oplus}$ will *always* be at least as steep. If we can derive convergence properties for $\alpha^{\oplus}$ then $\hat{\alpha}$ must converge at least as rapidly as that.

Suppose that the slope estimate, $\hat{\alpha}$ has error ε, so that $\hat{\alpha} = \alpha + \varepsilon$ and the upper bound estimate, $\alpha^{\oplus}$ has slope error $\varepsilon^{\oplus}$, so that $\alpha^{\oplus} = \alpha + \varepsilon^{\oplus}$, with $\varepsilon^{\oplus} \geq \varepsilon$. For $\varepsilon^{\oplus}$ to be strictly greater than some value, ϕ then all measurements $d_i \in D$ must lie above the line $(\alpha + \phi)t + \beta^{\oplus} + \delta$, and all measurements $q_j \in Q$ must lie below the line $(\alpha + \phi)t + \beta^{\oplus} - \delta$. This is trivially satisfied for one half of each line; for the other halves we express the slope error probability as:

$$P(\varepsilon^{\oplus} > \phi) = \prod_{j=0}^{m-1} P(q_j \leq (\alpha + \phi)Tj + \beta^{\oplus} - \delta) \tag{14}$$
$$\prod_{i=m}^{N-1} P(d_i > (\alpha + \phi)Ti + \beta^{\oplus} + \delta), \tag{15}$$

but because $(\alpha + \phi)t + \beta^{\oplus}$ intersects αt at t_{mid}, then $\beta^{\oplus} = -\phi t_{mid}$, so with further manipulation and using Equations 8 and 9 we obtain

$$P(\varepsilon^{\oplus} > \phi) = \prod_{j=0}^{m-1} P(v_j > (t_{mid} - Tj)\phi) \tag{16}$$
$$\prod_{i=m}^{N-1} P(w_i > (Ti - t_{mid})\phi) \tag{17}$$

Recalling that $t_{mid} = \frac{N-1}{2}T$ and $m = \left\lceil \frac{N-1}{2} \right\rceil$ this becomes

$$P(\varepsilon^{\oplus} > \phi) = \prod_{i=0}^{m-1} P(w_i > \phi Ti)^2 \tag{18}$$

Converting to the more standard Cumulative Density Function (CDF) form gives,

$$P(\varepsilon^{\oplus} \leq \phi) = 1 - \prod_{i=0}^{m-1} \left[1 - P(w_i \leq \phi Ti)\right]^2 \tag{19}$$

1) Skew Error Bound: Since $\alpha^{\oplus}$ is guaranteed to be at least as steep as $\hat{\alpha}$ then it must be the case that

$$P(\varepsilon \leq \phi) \geq P(\varepsilon^{\oplus} \leq \phi), \tag{20}$$

leading us to the main result of this Section, which is a bound on the probability distribution of slope error for the objective function of Equation 7,

$$\boxed{P(\varepsilon \leq \phi) \geq 1 - \prod_{i=0}^{m-1} \left[1 - P(w_i \leq \phi Ti)\right]^2} \tag{21}$$

This means that if we have knowledge of the network delay distribution, $p(w_i)$ then we are able to make good predictions about the skew error achieved by the TICSync estimator. Note that the convergence rate of the estimator is *independent* of the minimum network propagation delay, δ.

2) Offset Error Bound: Since the estimator $\alpha^{\oplus}$ is guaranteed to be at least as steep as the TICSync estimate, $\hat{\alpha}$, it also provides an upper bound on the offset error. If $\alpha^{\oplus}$ has slope error, $\varepsilon^{\oplus} = \alpha^{\oplus} - \alpha$ then its offset error after N samples will be $\varepsilon_{\tau}^{\oplus} = \varepsilon^{\oplus} TN/2$, providing a bound on the TICSync offset error, $\varepsilon_{\tau} \leq \varepsilon^{\oplus} TN/2$.

VI. Quantifying Estimator Error

In this Section we look at a way to quantify the convergence properties of the TICSync estimator. If we have a suitable model for the distribution of packet delays then we can substitute it into Equation 19 and ask questions about the number of samples required for the algorithm to converge to a required level of accuracy.

A. Modelling Network Noise

We use the Weibull distribution because of its ability to represent the sharp rise and long tail typical of network delay distributions. Another key advantage is the availability of algorithms for learning the parameters of the distribution online and incrementally. We will show that using a Weibull distribution will usually cause us to generate close, but conservative estimates of the TICSync estimator's error.

1) The Weibull Distribution: The Weibull distribution [12] is a more general form of the Exponential Distribution. It takes two parameters: shape, k and scale, λ and is defined as

$$f(t; \lambda, k) = \begin{cases} \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-\left(\frac{t}{\lambda}\right)^k} & t \geq 0 \\ 0 & t < 0 \end{cases}, \tag{22}$$

with the Cumulative Distribution Function being given by

$$F(t; \lambda, k) = 1 - e^{-\left(\frac{t}{\lambda}\right)^k} \tag{23}$$

The mean and variance of $f(t; \lambda, k)$ are

$$\mathrm{E}(t) = \lambda \Gamma\left(1 + \frac{1}{k}\right) \tag{24}$$
$$\mathrm{var}(t) = \lambda^2 \Gamma\left(1 + \frac{2}{k}\right) - \mathrm{E}(t)^2 . \tag{25}$$

with Γ being the Gamma function.

2) Finding a bound on the true distribution: If the network delays are Weibull distributed with $w_i \sim f(\lambda, k)$, then

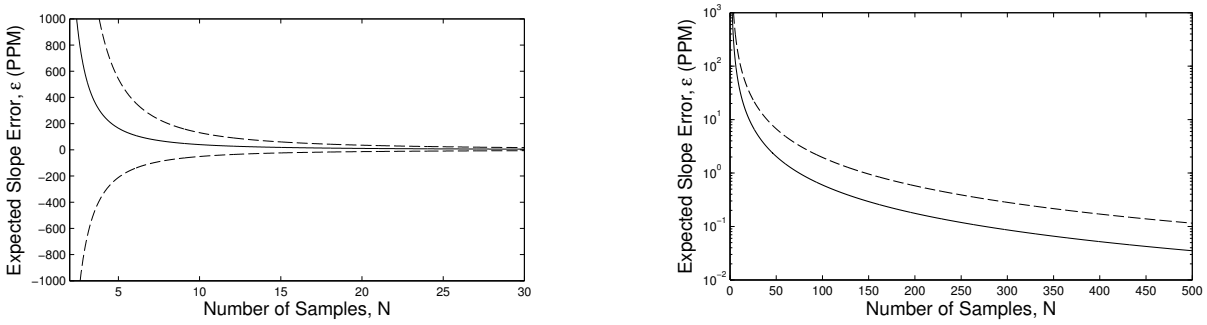
$$P(w_i \leq \phi) = 1 - e^{-\left(\frac{\phi}{\lambda}\right)^k}. \tag{26}$$

Inserting this into Equation 21 gives

$$P(\varepsilon \leq \phi) \geq 1 - \prod_{i=0}^{m-1} e^{-2\left(\frac{\phi Ti}{\lambda}\right)^k} \tag{27}$$

which in log form becomes

$$\ln\left[P(\varepsilon > \phi)\right] \geq -2 \sum_{i=0}^{m-1} \left(\frac{\phi Ti}{\lambda}\right)^k \tag{28}$$
$$= -2\left(\frac{\phi T}{\lambda}\right)^k \sum_{i=0}^{m-1} i^k \tag{29}$$



(a) (b)

Fig. 6: Expected slope error of the TICSync estimator for 10Hz sampling and Weibull parameters $k = 1.33$ and $\lambda = 5.4 \times 10^{-5}$. The dotted lines are 3σ bounds. Note the rapid convergence. The figure on the right shows the same plot on a log scale. Errors are given in Parts Per Million (PPM)

The summation term in Equation 28 has no closed form solution. Instead we must resort to finding bounds on the true result. Consider the function

$$g(N) = \sum_{i=0}^{N-1} i^k \tag{30}$$

The discrete function i^k is bounded above and below by the continuous functions $y = (x+1)^k$ and $y = x^k$ respectively. Thus we may obtain the following bounds on the function $g(N)$,

$$\int_0^{N-1} x^k dx \quad < \quad g(N) \quad < \quad \int_0^{N-1} (x+1)^k dx \tag{31}$$

$$\frac{(N-1)^{k+1}}{k+1} \quad < \quad g(N) \quad < \quad \frac{N^{k+1}}{k+1} \tag{32}$$

As $N \to \infty$, the bounds converge, because

$$\lim_{N\to\infty} \frac{(N-1)^{k+1}}{k+1} = \frac{N^{k+1}}{k+1} \tag{33}$$

and in practice the bounds are tight for $N >> 1$. Applying this result to Equation 28 yields

$$\ln\left[P(\varepsilon > \phi)\right] \geq -2\left(\frac{\phi T}{\lambda}\right)^k \frac{m^{k+1}}{k+1} \tag{34}$$

and finally, approximating $m \approx (N-1)/2$ and rearranging gives

$$\boxed{\ln\left[P(\varepsilon > \phi)\right] \geq -\left(\frac{\phi}{\psi\lambda}\right)^k} \tag{35}$$

where $\psi = \frac{2}{T(N-1)}\left(\frac{k+1}{N-1}\right)^{\frac{1}{k}}$.

We are now equipped with a probabilistic bound for the convergence properties of the TICSync estimator, defined in terms of the network delay distribution parameters, λ and k, the sample period, T and the number of samples, N.

B. Quantitative estimates

Differentiating Equation 35 allows us to recover the PDF over slope error for the TICSync estimator,

$$\begin{aligned} p(\varepsilon) &= -\frac{d}{d\phi}P(\varepsilon > \phi) \\ &= \frac{k}{\psi\lambda}\left(\frac{\varepsilon}{\psi\lambda}\right)^{k-1} e^{-\left(\frac{\varepsilon}{\psi\lambda}\right)^k} \end{aligned} \tag{36}$$

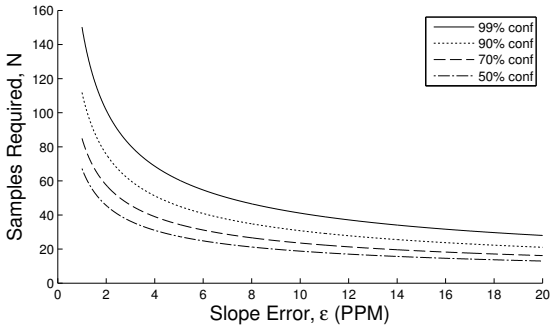


Fig. 7: Required number of samples to achieve a given level of slope accuracy, with $k = 1.33$ and $\lambda = 5.4 \times 10^{-5}$ and a sampling frequency of 10Hz. Various confidence levels are shown. Errors are given in Parts Per Million (PPM).

which is a pleasing result, being a Weibull distribution, with $p(\varepsilon) = f(\varepsilon; \psi\lambda, k)$ – ie. a scaled version of the original distribution in Equation 22.

With reference to the mean and variance of the Weibull distribution (Equations 24 and 25) we can state that,

$$\mathrm{E}(\varepsilon) \leq \psi\lambda\Gamma\left(1 + \frac{1}{k}\right) \tag{37}$$

$$\mathrm{var}(\varepsilon) \leq (\psi\lambda)^2\left[\Gamma\left(1+\frac{2}{k}\right) - \left[\Gamma\left(1+\frac{1}{k}\right)\right]^2\right]. \tag{38}$$

The availability of fast and accurate approximation algorithms for the Gamma distribution [13] make these feasible for online use. Using them we can calculate the expected error of the TICSync estimator, given the number of samples, N.

We fitted a Weibull distribution to a set of sample offset data gathered on a Gigabit local network and found suitable parameters to be $k = 1.33$ and $\lambda = 5.4 \times 10^{-5}$. Figure 6 shows the expected error with a sample rate of $10Hz$, along with 3σ bounds calculated from Equation 38.

Perhaps more usefully, we may also derive an expression for estimating the minimum number of samples required to achieve a particular slope error ε with a given confidence c. The confidence value lies in the range $[0, 1]$, so that $c = 0.99$ corresponds to a 99% confidence bound. Rearranging

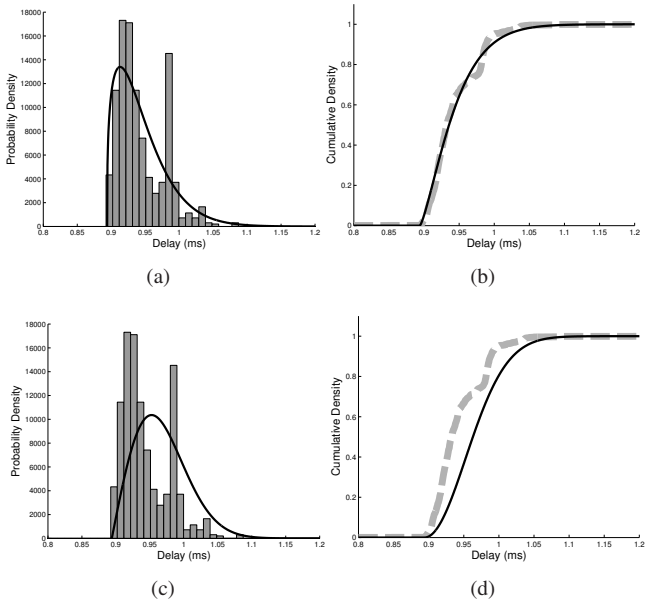


Fig. 8: A packet delay measurement experiment was performed over a fast Gigabit network between two standard PCs, with a message rate of 2Hz. (a) shows a histogram of 1000 delay measurements, along with the Weibull Distribution fitted by the Modified Moment Estimator (parameters: $\lambda = 5.4 \times 10^{-5}, k = 1.33$). It is clear that the data is not really drawn from a Weibull distribution. However, our error estimates make use of the Cumulative Distribution rather than the PDF. (b) shows the CDF of the data as a thick dashed line, with the CDF of the Weibull distribution overlaid. The similarity here is much greater. (c) shows the same histogram, but this time the MME was forced to choose a shape parameter value of $k > 2$. The front end of the distribution is less steep, leading to a conservative estimate of the CDF (d).

Equation 35,

$$N \geq \left[-(k+1) \left(\frac{2\lambda}{\varepsilon T} \right)^{k} \ln(1-c) \right]^{\frac{1}{k+1}} + 1 \qquad (39)$$

Sample plots of this function are shown in Figure 7. Note that reducing the required confidence by only a small amount can drastically reduce the required number of samples.

VII. Learning the Weibull Distribution Online

In order to make good predictions of error or required sample counts, it is important that the network delay distribution is well characterized. In practice we need to learn the distribution online as data is accumulated.

We use the 3-parameter Modified Moment Estimator [14] to learn the distribution incrementally from the measured round trip times. The MME algorithm is easily made to run online, with amortized $O(1)$ updates.

Figure 8 shows some real network delay histograms from a week-long experiment we carried out on our local Gigabit network. The packet delay distribution is clearly multimodal, so the Weibull model is not strictly correct. But recall that our expression for the TICSync estimator error in Equation 21 uses the CDF of the distribution, rather than the PDF. In fact, the Weibull model follows the CDF of the raw data rather well.

1) Conservative Estimates: If the initial part of the Weibull CDF is too steep, or leads the true CDF then it will result in overconfident TICSync error estimates. Conversely, if the Weibull CDF lags the true CDF or begins at too shallow a slope then it will result in conservative error estimates. The latter situation is clearly preferable, so on that basis we place a restriction on the minimum value of the shape parameter for our Weibull distributions, enforcing that $k > 2$. The decision is not arbitrary, as $k = 2$ is the smallest value for which the initial slope of the Weibull distribution is 0. It helps to ensure that the sharp front edge of the true histogram is only softly reproduced by the Weibull distribution. Figure 8 shows the result of applying the restriction. We find that the approximation works very well throughout our week-long dataset, giving a Weibull CDF which is close to the true data, but usually conservative.

VIII. Results

We performed both synthetic and physical experiments to measure how well TICSync can learn the mapping between separate clocks. Ground truth was hard to provide to the accuracy required, so we ran the client and time server processes on the same machine, but caused the communications between them to be routed from Oxford, England, to an Amazon EC2 server in North America and then back again. This resulted in a mean round trip time of 162ms and authentic network delays.

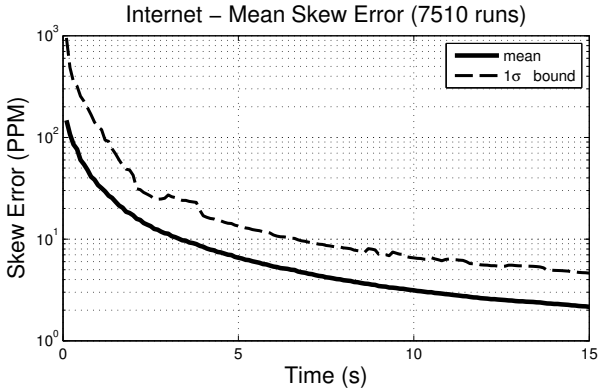
Figure 9 shows that the skew error evolved much as predicted by the probabilistic bounds. The offset error converges very rapidly to microsecond levels, but stabilizes at around 35μs, as the the granularity of the computer clock measurements is reached, and quantization errors begin to dominate.

We also ran a series of 1500 synthetic experiments where delay data was drawn from a Weibull distribution and random skews and offsets were applied (see Figure 10). The ensemble average shows good agreement with the probabilistic predictions. Recall that the predictions are an upper bound, which explains the separation between expected and actual results.

The rapid convergence and low computational cost of the TICSync estimator make it ideal for tracking low frequency drift such as that in Figure 1. A pair of overlapping estimators which are occasionally and alternately reset are a simple and effective solution. More advanced upset detection is made possible by using the probabilistic error bounds on the TICSync estimator, but we leave this for future work.

IX. Conclusions

In this paper we have described a new algorithm, TICSync, for unified and precise timing across distributed networked systems. Its standout properties are rapid convergence and probabilistic bounds on performance. Crucially it has O(1) incremental complexity. We have demonstrated its performance on real data. The value of TICSync to the robotics community lies in distributed sensing, its application to synchronizing sensors to hosts, and allowing unified time



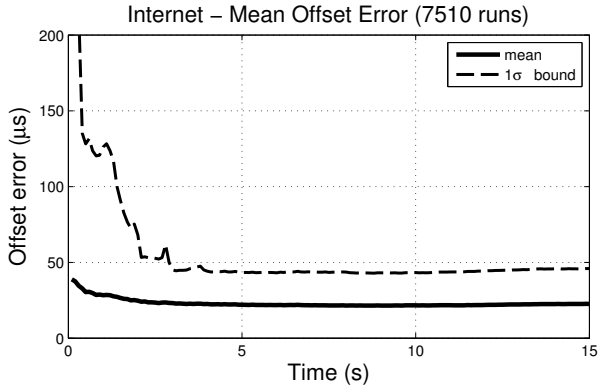
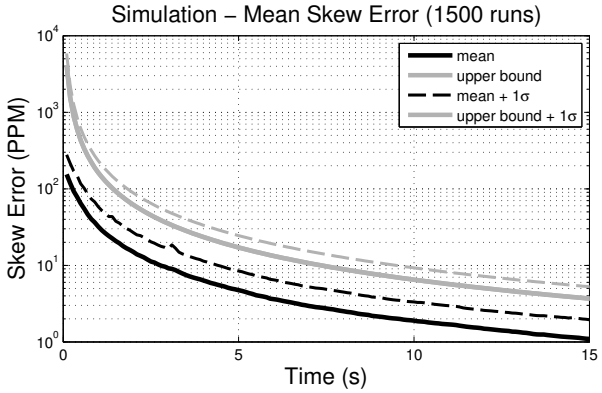


Fig. 9: Results from a physical experiment run on a standard desktop PC, where timing pings at 10Hz were sent between Europe and North America. The mean round trip time was 162ms. TICSync was run over 7510 ping sequences, each lasting 15s and an ensemble average was taken. Offset error almost immediately comes down to tens of microseconds, which is close to the resolution of the PC clock used in this experiment.



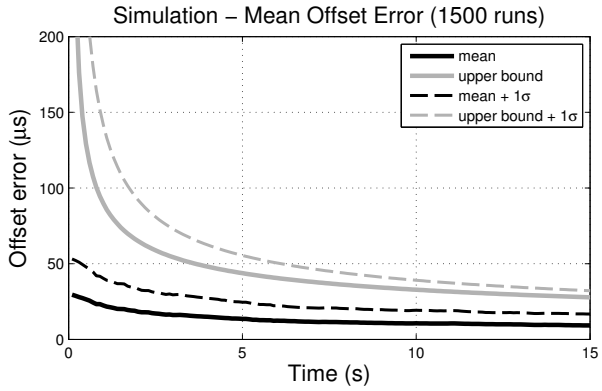


Fig. 10: Synthetic data experiments to compare TICSync performance with expected performance (from Equation 37). Packet delay data was generated at 10Hz from a Weibull distribution with parameters $\lambda = 1.4 \times 10^{-4}, k = 2.5$ and minimum network delay $\delta = 75$ms, which were found to best fit our actual round-internet data. We performed 1500 independent experiments and then took the ensemble average to produce the figures here.

stamping in robots with multiple on-board computers. We will be releasing TICSync as a library for use by the community at large. Our intent is to provide excellent timing for practically zero overhead, allowing us to estimate when things happened and with what temporal confidence.

X. Acknowledgements

This work has been generously supported by the European Commission under grant agreement number FP7-231888-EUROPA and the Office of Naval Research Grants N000140810337 and N000140710550.

References

[1] D. Mills, "Internet time synchronization: The network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.

[2] F. Tungadi and L. Kleeman, "Time synchronisation and calibration of odometry and range sensors for high-speed mobile robot mapping," in *Proc. Australasian Conference on Robotics and Automation*, Canberra, Australia, December 2008.

[3] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.

[4] R. Gusella and S. Zatti, "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Trans. Software Eng.*, vol. 15, no. 7, pp. 847–853, 1989.

[5] V. Paxson, "On calibrating measurements of packet transit times," in *SIGMETRICS*, 1998, pp. 11–21.

[6] D. Veitch, J. Ridoux, and S. B. Korada, "Robust synchronization of absolute and difference clocks over networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 417–430, 2009.

[7] J. Aweya, D. Y. Montuno, M. Ouellette, and K. Felske, "Clock recovery based on packet inter-arrival time averaging," *Computer Communications*, vol. 29, no. 10, pp. 1696–1709, 2006.

[8] K.-L. Noh, Q. Chaudhari, E. Serpedin, and B. Suter, "Novel clock phase offset and skew estimation using two-way timing message exchanges for wireless sensor networks," *Communications, IEEE Transactions on*, vol. 55, no. 4, pp. 766–777, April 2007.

[9] S. B. Moon, P. Skelly, and D. F. Towsley, "Estimation and removal of clock skew from network delay measurements," in *INFOCOM*, 1999, pp. 227–234.

[10] L. Zhang, Z. Liu, and C. Honghui Xia, "Clock synchronization algorithms for network measurements," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1. IEEE, 2002, pp. 160–169 vol.1.

[11] R. Sirdey and F. Maurice, "A linear programming approach to highly precise clock synchronization over a packet network," *4OR*, vol. 6, no. 4, pp. 393–401, 2008.

[12] A. C. Cohen and B. Whitten, *Parameter estimation in reliability and life span models*, 1st ed., ser. STATISTICS: textbooks and monographs. New York: Marcel Dekker, Inc, September 1988.

[13] W. J. Cody, "An overview of software development for special functions," in *Proceedings of the Dundee Conference on Numerical Analysis*, vol. 506/1976. Springer Berlin / Heidelberg, 1975, pp. 38–48.

[14] C. A. Cohen and B. Whitten, "Modified maximum likelihood and modified moment estimators for the three-paramter weibull distribution," *Communications in Statistics - Theory and Methods*, vol. 11, no. 23, pp. 2631–2656, 1982.