Time Keeping in Myriad Networks: Theories, Solutions and Applications

by

Aggelos Anastasiou Bletsas

Diploma in Electrical and Computer Engineering
Aristotle University of Thessaloniki (1998)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

Author _____
Program in Media Arts and Sciences
May 11, 2001

Certified by _____
Shrikumar Hariharasubrahmanian
Visiting Research Scientist
MIT Media Lab
Research Supervisor

Certified by _____
Stephen A. Benton
Allen Professor of Media Technology
MIT Media Lab
Thesis Supervisor

Accepted by _____
Stephen A. Benton
Chair, Department Committee on Graduate Students
Program in Media Arts and Sciences

Time Keeping in Myriad Networks: Theories, Solutions and Applications

by

Aggelos Anastasiou Bletsas

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning on May 11, 2001,
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ABSTRACT

Distributed sensor networks make extensive use of a common time reference. In this work we address the problem of time dissemination in a packet switched network when the nodes are NOT generally all connected to an accurate, external time reference source.

We thoroughly analyze Network Time Protocol – version 3 and identify its oversimplified clock modeling and its neglect of network delay variance (network jitter) as the primal causes for its inaccuracy. We explicitly address frequency skew in our clock model and propose a novel Kalman filtering technique for de-noising (remove of network jitter) during the NTP time synchronization process. The parameters of the Kalman linear estimator are optimal and they are computed online from the network environment, with a well-defined procedure. Our End-to-End technique decreases NTP rms error by two orders of magnitude and is compared with a software phased lock loop and a linear programming technique, with cross traffic exhibiting long-range dependence (fractional Brownian motion cross-traffic) or no dependence at all (white Gaussian case).

We conclude with applications over packet switched networks that require time synchronization, like spatial filtering (beam-forming). The suite of algorithms and applications define a new class of packet switched networks, called Myriad Networks.

Research Supervisor: Shrikumar Hariharasubrahmanian
Title: Visiting Research Scientist, MIT Media Laboratory

Thesis Supervisor: Stephen A. Benton
Title: Allen Professor of Media Technology, MIT Media Laboratory

Time Keeping in Myriad Networks: Theories, Solutions and Applications

by

Aggelos Anastasiou Bletsas

Thesis Readers

Approved by

Neil Gershenfeld

Associate Professor of Media Arts and Sciences
MIT Media Lab

Approved by

Joseph M. Jacobson

Associate Professor of Media Arts and Sciences
MIT Media Lab

Acknowledgments

Coming to MIT was a very old dream for me. It should be those years in the secondary school when I read in a greek daily newspaper about the intellectual power and the hard-working ethics of MIT students. I soon realized that I wanted to become part of the MIT community. During my undergraduate studies in Greece I worked hard and I tried to enrich my CV so as to prove that I was eligible for this place. Easy solutions and average grades were never within my choices: only the best that a few can achieve, since that is what an MIT student always tries to do. With hard work and also a lot of luck, since my brother Michalis was already in MIT, I was admitted for the fall semester of 1999.

Prof. J. Jacobson picked me and brought me here, even though I hadn't applied directly to him. I am grateful to him for this, once in a lifetime, opportunity. With him and one of his smart students and thereinafter good friend of mine, Brian Hubert, we created a nano-assembly machine which was eventually awarded the MIT-Lemelson prize. The whole project was like a lovers' relationship: tremendous excitement at the beginning, exploration and experimentation at the second stages, frustration and anger at some other time points. In any case, it was a rich and fruitful experience from which I learned the value of patience and the importance of persistence.

I would also like to help Prof. N. Gershenfeld who allowed me to work at his lab and interact with his fine students. Matt Reynolds, Matt Hancher, Rich Fletcher, Femi Omojola, Yael Maguire, Jason Taylor, Ravi Pappu, Ben Vigoda, Zoe Teegarden and Ben Brecht were always ready to discuss a good idea or help. Particularly Matt Hancher, Rich and Femi helped me with all the demos I was involved, so I would like to thank them. My research supervisor there, Shrikumar H. with his own special way, taught me the significance of persisting and fighting for what I believe is right. He was excited about my work and gave me useful research feedback.

Prof. Gershenfeld and Prof. Jacobson gave me detailed remarks and comments on my draft document improving the presentation of the material. My thesis supervisor, Prof. Benton was also reliably available for any help. I'm grateful to all of them.

Karrie Karahalios, Rosane Kariadakis and Murray Whitehead corrected my silly english mistakes in the draft document. My officemates Jeremy Levitan and Aaron Weber kept me good company in the research breaks.

Linda Peterson deserves credits for her administrative assistance and her willingness to always find time to talk with me and answer my questions.

My brother Michalis always tried to show me that there are many important issues in life and I should not worry that much. His common phrase "oh, come on now... " guided me, alleviated

difficult situations and his office sofa was always the best retreat after the biggest frustrations. This work would never finish without him being always there for me.

My family was a great resource of energy simple because they deeply believe in me. The weekly telephone call from my parents or my girlfriend in Greece, from my sister and her husband in Norway was full of encouragement and motivation for better work. I dedicate this work to all of them.

Στο Χριστινάκι,

Contents

1 Introduction

1.1 Motivations

The last decade marked a revolution in the way people and machines exchanged information: it was the era of Networks.

In the beginning of the new millennium we are experiencing a strong interest not only in the manipulation, exchange and presentation of numeric and symbolic data but also in the interaction with the dynamic physical world, through sophisticated sensors and actuators. This bridge between the virtual world *("bits")* and the physical world *("atoms")* was identified quite early in the late 60's by the Nobel laureate H. Simon [Simon1969], however it was the recent advance in low cost VLSI design that led to active research in large scale, highly distributed systems of miniaturized nodes with remarkable information sensing, processing and actuating capabilities.

In brief, Networks of the future are characterized by an unprecedented number of nodes minimal in size, resource constrained but able to exploit their sheer numbers so as to perform a coherent task. Ubiquitous Computing [Weiser1991], Amorphous Computing [Abelson2000], Smart Dust [Kahn1999], Proactive Computing [Tennenhouse2000], Paintable Computing [Butera2001] and Computing Swarm [Evans2000] are different proposals dealing with various aspects of large scale distributed systems, as we described them above, showing a bold mindset among the research community about the ways computing will/should be realized in the future. A common denominator in all the above efforts, explicitly or implicitly apparent, is the critical need for *Time Synchronization*. Distributed sensor networks make extensive use of synchronized (common) time:

i) to measure the time-of-flight of known signals in order to calculate relative distances and therefore relative position. Relative position and common time reference is a critical piece of information for beam-forming (*spatial filtering*) applications.

ii) to time-stamp and buffer diverse forms of sampled information upon acquisition, "allowing their subsequent processing to proceed asynchronously with respect to the external processes being monitored" [Tennenhouse2000]. In that mode, programming of each node is simplified

since processing of the information is streamlined after data aggregation from multiple nodes. Local processing, *data fusion* and data summarization are often employed because of the reduced energy cost of computation compared to the cost of communication. This *proactive* mode of future networks would be impossible without a common time reference.

iii) to discard duplicate correlated information captured by different sensors, to coordinate events scheduled in the future or to deploy cryptographic algorithms/techniques.

Therefore, a common time reference is considered a given, explicitly or implicitly due to the reasons explained above, in the realizations of future large-scale distributed processing systems. This assumption, its importance, as well as the fact that common, accurate time keeping in large networks is not an easy, inexpensive task synthesize the motivation behind this work. **Global Positioning System** (G.P.S.) is not considered a solution to the problem mainly because of the following reasons:

a) it requires an expensive, external satellite network
b) it performs poorly at indoor structures, due to its high carrier frequency and large attenuation
c) it is unsuitable for underwater or extraterrestrial environments
d) GPS transceivers are still large in size compared to sensor nodes and expensive, especially when they are considered to be attached to a sheer number of nodes.

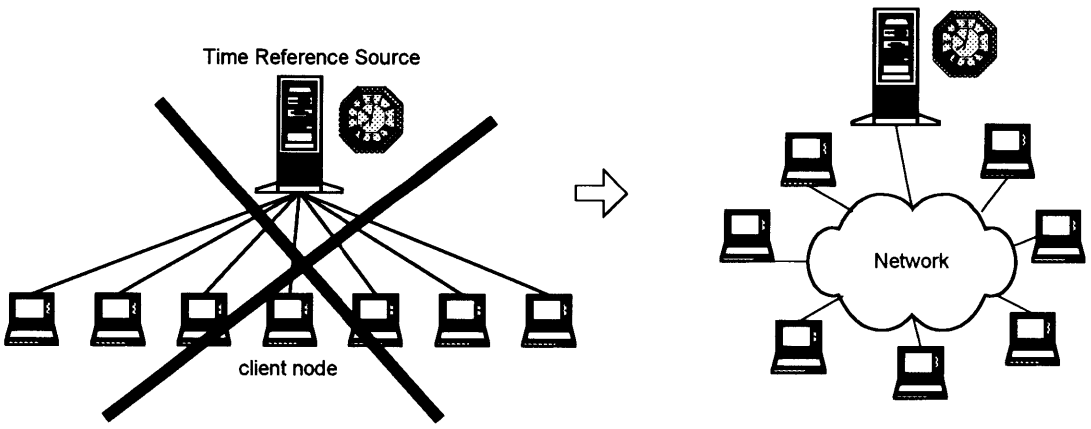


Figure 1.1 Centralized Timing vs Distributed Time Keeping over a network. The second approach is the focus of this thesis.

The questions addressed in this thesis are simple: can the **network itself** provide a time keeping service with certain accuracy? What are the hardware requirements of each node? What are the requirements of the network architecture? Is it going to be an End-to-End solution for integrated time-keeping or is it going to require vast architectural changes in the way we currently think about networks? Which of the proposed solutions can be incorporated in the current Internet and what are the implications (and applications)?

We will attempt to answer the entire above questions focusing on the End-to-End approaches, having in mind the implementation of the proposed algorithms in the current Internet. Of course, *our long-term goal is to incorporate the proposed algorithms into a Network Architecture suitable for large-scale distributed processing applications, called "**Myriad Networks**" where the coherent collaboration of a sheer number of unorganized, minimal in size and resource constrained nodes is simplified by the common time reference.*

1.2 Roadmap

In the following chapter we will review the basics of time-keeping hardware structures and spotlight the physical causes behind clock instabilities. Basic terminology will be introduced. In chapter 3 we are going to analyze thoroughly the Network Time Protocol and identify the reasons behind its shortcomings, basically sloppy clock modeling and queuing delays variance (jitter). Chapter 4 reviews briefly other approaches in the field of common time in distributed system. In Chapter 5, based on the findings of the previous chapters we propose a novel, structured and optimal Kalman filter for network de-jittering, frequency skew and clock offset estimation and compare it with a software phased lock loop approach as well as with a linear programming technique, in terms of time synchronization error, computational efficiency as well as other factors. The traffic models used exhibit short-range dependence (jitter as white gaussian noise) or long range dependence (*self-similar* traffic), testing the algorithms at the two extremes. We summarize our findings about our novel End-to-End synchronization technique in chapter 6 and finally, we provide examples of applications that require or could benefit from time-synchronization.

2 Clock Basics

Which are the structures we have invented in order to measure and keep time? Since we are interested in time synchronization, what are the fundamental reasons behind instability of a clock? For the sake of completeness we will briefly address the above questions. In subsequent sections we are going to show that time synchronization in a packet switched network is affected by two major factors:

a) *Clocks offset*, *skew* and *drift* due to physical limitations.
b) Network one way delay variance (jitter) due to the way packet-switched networks are currently structured.

As a matter of fact, we will show (in the following chapter) that even if the physical limitations could be overcome, still the time synchronization could be affected by network jitter. In this chapter the physical causes behind clock offset, skew and drift will be outlined.

2.1 Terminology

A *clock* T(t) reporting the value of true time t, can be considered as a piecewise continuous function, twice differentiable except on a finite set of points:

$$\text{T(t): } R \longrightarrow R$$

Where $T'(t) = dT(t)/dt$, $T''(t) = d^2T(t)/dt^2$ exist everywhere except for $t \in P \subseteq R$, $\|P\|$ is finite. Using the above definition, the set P includes all the points were the clock parameters suddenly change, therefore no derivative of the function of time T(t) can be defined. In this work, we are going to model each clock as a piecewise liner function of time T(t). At the figure below, the set P includes time points t_1, t_2 and t_n.

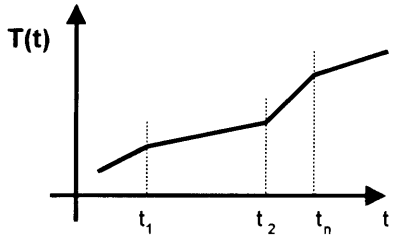


Figure 2.1 piecewise linear clock modeling in this thesis

A clock able to report the true time, a "true" clock can be described by the identity below:

$$T(t) = t \text{ and } \|P\| = 0$$

which basically means that the theoretical source of true time is a strictly linear function of time (not piecewise linear function). In practice, clocks due to thermodynamic fluctuations deviate from the true time and we are going to be more detailed about these reasons.

Using the representation T(t) to describe a clock, as explained above, we can give the following definitions:

- **Offset:** the difference between the time reported by a clock and the "true" time; the offset of $T_a(t)$ is $(T_a(t)-t)$. The offset of the clock $T_a(t)$ relative to $T_b(t)$ at time t, is $T_a(t)- T_b(t)$.

- **Skew:** the difference in the frequencies of a clock and the "true" clock. The skew of $T_a(t)$ relative to $T_b(t)$ is $(T_a'(t) - T_b'(t))$.

- **Drift:** the drift $T''(t)$ of a clock shows the rate with which the clock's update frequency changes, therefore ideally it should be zero. The drift of $T_a(t)$ relative to $T_b(t)$ at time t is $(T_a''(t) - T_b''(t))$.

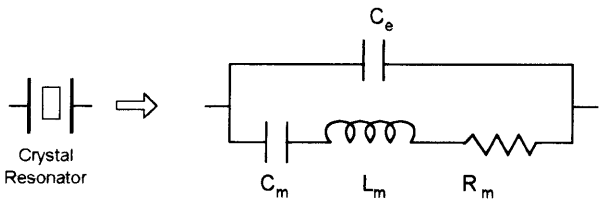
Two clocks are said to be *synchronized* at a particular moment if both the relative offset and skew are zero.

Having given the basic definitions, we can now proceed to the natural causes (physical limitations) of clock offsets, skew and drift.

2.2 Clock Generation Mechanisms and Their Uncertainties

Piezoelectric crystals are the basic oscillators used in clock generation mechanisms. The electrical behavior of those crystals depend on their mechanical properties and the effective circuit of a crystal consist of the electrical capacitance C_e of the crystal in parallel with an RLC component associated with each mechanical response. In particular, C_m corresponds to the spring mechanical element and stores energy due to displacement, L_m corresponds to the mass of the material and R_m corresponds to the damping element (dashpot) representing the mechanical dissipation. The ratio of C_e/C_m is a measure of the interconversion between electrical and mechanical energy stored in the crystal, i.e., of the piezoelectric coupling factor, k.

Figure 2.2 Effective circuit of a piezoelectric crystal



A crystal resonator is connected to the feedback loop of an amplifier with output/input relation y = - Ax. The feedback loop scales the output with a complex coefficient F(ω) according to the relation x = F(ω) y. Therefore, x = - A F(ω) x, which is feasible only if A F(ω) = -1. F(ω) is complex, therefore its phase should be a multiple of π and actually this happens for $\omega = \frac{1}{\sqrt{L_m C_m}}$. Thus, any noise initially in the circuit at that frequency will grow exponentially until it either clips or it is intentionally limited, providing the clock waveform.

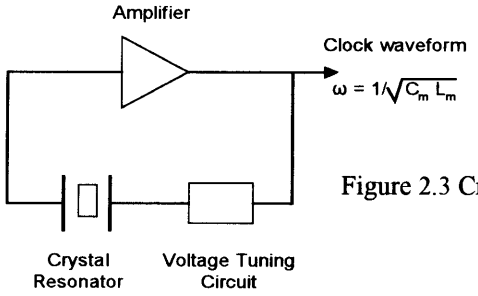


Figure 2.3 Crystal Oscillator

Quartz (SiO_2) is the most common material used. For a quartz resonator, C_m is typically on the order of femtofarads and L_m is millihenrys, giving resonant frequencies on the order of megahertz and showing mechanical properties with weak thermal dependence. The quality factor $Q = f/\delta f$ of a quartz resonator is on the order of 10^4-10^5 giving a relative temporal uncertainty $\delta t/t$ on the order of 10^{-5}-10^{-4}.

Regular LC circuits with resonant frequencies on the order of megahertz are more prone to thermal drift and parasitic coupling. Moreover, the quality factor is far less on the order of 10-100.

Therefore, piezoelectric resonators are more stable than regular LC circuits, however *there is still a frequency variation (frequency drift as defined above) due to thermal drift, which changes the crystal stiffness*. Frequency drift can be reduced by measuring the temperature and using the measure to tune the resonator circuit as in a TCXO (Temperature Compensated Crystal Oscillator)

or by controlling the temperature at a set-point where the crystal is less sensitive to drift as in OCXO (Oven Compensated Crystal Oscillator). Table 2.4 shows the uncertainty in time for the various versions of piezoelectric resonators. *For example OCXOs have an uncertainty of 10^{-8}, which means that an error of 1 ns in time measurement will happen in $10^{-9}/10^{-8}$ = 0.1 sec. Therefore in 1 sec there would be an uncertainty of 10 nsecs, giving an uncertainty of $3\ 10^{8} * 10 * 10^{-9}$ = 3 m in range estimation, assuming RF signals!*

Therefore, even the impressive stability/certainty of 10^{-8} may not be adequate for certain classes of applications, unless there is a repeated correction mechanism that reduces that frequency variation (and therefore increasing the precision). Another major problem with piezoelectric oscillators is the bias (accuracy) error due to the fact that the nominal values of the crystal depend on its physical properties (dimensions, shape etc.). Even if thermal drifts could be controlled, if the dimension of the crystal does not correspond to its nominal frequency, offset from "true" time should be anticipated. Clock precision and accuracy are affected from macroscopic properties of the piezoelectric resonators used (dimensions, environmental temperature etc.). Macroscopic objects may differ, but the microscopic properties of all atoms are identical. "Quantum mechanically there is no way to distinguish between two atoms, so clocks based on atomic resonators will keep the same time" [Gershenfeld2000]. Switching from quartz to a cesium beam reduces the relative uncertainty in time to 10^{-12}-10^{-11}. By using lasers this can be extended to 10^{-15}. For a detailed survey of Atomic Clocks, the interested reader is referred to [Major1998].

In this chapter we tried briefly to outline the physical causes of lack of precision (variation) and accuracy (bias) in clock structures. In the following chapter, we are going to show that even if we could equip all our computers with atomic clocks, network one way-delay jitter could still deteriorate time synchronization among distant computers. Before ending this chapter, we will give the definition of 1 second, which nowadays is defined quantum mechanically.

Hierarchy of Oscillators

Oscillator Type*	Accuracy**	Typical Applications
Crystal oscillator (XO)	10^{-5} to 10^{-4}	Computer timing
Temperature compensated crystal oscillator (TCXO)	10^{-6}	Frequency control in tactical radios
Microcomputer compensated crystal oscillator (MCXO)	10^{-6} to 10^{-7}	Spread spectrum system clock
Oven controlled crystal oscillator (OCXO)	10^{-8} (with 10^{-10} per g option)	Navigation system clock & frequency standard, MTI radar
Small atomic frequency standard (Rb, RbXO)	10^{-9}	C^3 satellite terminals, bistatic, & multistatic radar
High performance atomic standard (Cs)	10^{-12} to 10^{-11}	Strategic C^3, EW

* Sizes range from <5cm³ for clock oscillators to > 30 liters for Cs standards
Costs range from <$5 for clock oscillators to > $50,000 for Cs standards.

** Including environmental effects (e.g., -40°C to +75°C) and one year of aging.

Table 2.4 Hierarchy of Oscillators. Strategic C^3 includes applications based on the the well known GPS satellite constellation

2.3 Definition of a "second"

The time unit "second" is one of the SI base units. Until 1956 the second was derived from the earth's rotation around its axis, later from the earth's motion around the sun. In 1967, the change from the astronomical to the atomic definition of the second was done, because the frequency of the electromagnetic radiation emitted by atoms is much more constant in time than the angular frequency of the earth or the oscillation frequency of a pendulum or of a quartz oscillator (as we showed above). The new definition of the SI second is based on the non-radioactive cesium, Cs^{133}, whose atomic frequency had been fixed at 9192631770 Hz in 1967.

3 Network Time Protocol Analysis

In the previous chapter we attributed the physical causes of clock offset, skew and drift, mainly to thermodynamic effects. In this chapter we will examine one of the oldest Internet Protocols, namely the Network Time Protocol (NTP) which was designed to provide *Time Synchronization* across IP networks and find out that network one-way delay variance (jitter) is underestimated, deteriorating the efficiency of the protocol time sync algorithms. This finding is validated with detailed modeling and analysis, experimental results over the Internet as well as simulations.

3.1 NTP Overview

NTP is a hierarchical, semi-self organizing protocol. A layer of *stratum-1* servers are connected to a source of *"true time"* which can be a GPS or a WWVB receiver (WWVB is terrestrial, radio, time dissemination service operated by the National Institute of Standards and Technology). Stratum-1 servers are the time source for the second layer of servers, stratum-2 servers that are the time source for the third layer of NTP servers, stratum-3 servers and so on, up to stratum-16. The need for a hierarchical spanning tree of NTP servers is required so as to balance the NTP packet traffic across the network avoiding overloading specific links and NTP servers. Moreover, the NTP algorithm favors samples coming from "closer" NTP servers (i.e. samples with smaller delay).

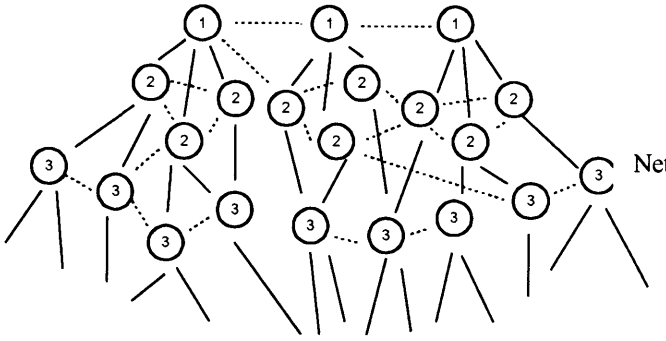


Figure 3.1
NTP
Network Hierarchy

The NTP daemon is manually set and the administrator should select the synchronization peers in a client/server mode. There are other modes as well, symmetric active or passive and

broadcast modes used between clusters of NTP servers for redundancy and error control, however we will not examine them. The interested reader should refer to RFC-1305, which is the specification of NTP – version 3 [Mills1992], the version widely deployed and the version on which our analysis is based upon. NTP – version 4, has a focus on more efficient clock discipline algorithms however the central idea remains the same.

After the peer stratum servers have been defined, the client NTP daemon can run unsupervised, electing the samples from the stratum servers having the smallest *dispersion*, which is the smallest error in synchronization based on the roundtrip delay of the NTP packet as well as the frequency skew error, accumulated during the message exchange. *Root dispersion*, from the synchronization peer to its stratum-1 server is also considered in the election of the most appropriate peer. History of 8 packets is held for every peer and weighted sums of the sample dispersions are used for the election of the best sample/peer as well as the best peer (*Filter selection, Clock selection algorithms*). Message exchange with every peer happens every 64 seconds or more depending on how slowly the peer is exchanging NTP packets with its own lower stratum time source.

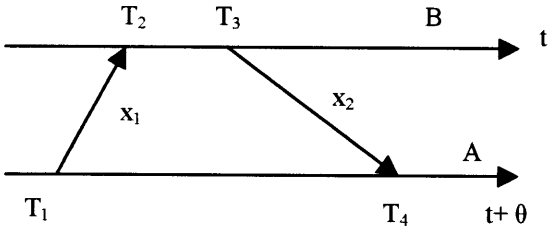


Figure 3.2
NTP
Message exchange

At each message exchange with each peer server, the NTP client daemon makes an estimate of the clock offset between its clock and its peer clock. After timestamping the NTP packet (T_1), it sends it (over UDP) to its peer. The NTP server timestamps the packet according to its own clock, at the times of arrival and departure (T_2, T_3) and the client timestamps the packet again upon arrival (T_4). At each sample (T_1,T_2,T_3,T_4) in one NTP packet, the NTP daemon in A calculates the following:

$$T_2 - T_1 = \alpha, \quad T_3 - T_4 = b$$

Roundtrip time $\delta = T_4 - T_1 - (T_3 - T_2) = T_2 - T_1 - (T_3 - T_4) = a - b$ (1)

Estimated offset $\theta = \frac{T_2 - T_1 + T_3 - T_4}{2} = \frac{a+b}{2}$ (2)

Observe that the one way delay from A to B is denoted as x_1 and the one way delay from B to A as x_2. The one way delays are due to propagation delays (the time needed for the first bit to be

propagated, to reach the destination and is limited of course by the speed of light), transmission delays (the time needed after the first bit has arrived so as the reception of the packet be completed and is limited by the link capacity) as well as queueing delays. The network is a shared medium, in a chaotic fashion [Taqqu1997], therefore the distribution of one way delay is nearly impossible to be characterized due to the chaotic behaviour of the cross traffic. Therefore, even if the forward path from A to B is physically the same as the path from B to A, the cross traffic will generally be different across the two paths, leading to variable queueing delays, therefore we can conclude that:

$$x_1 \neq x_2$$

However, the assumption $x_1=x_2$ is hidden in the offset estimation formula above. This erroneous assumption is the cause of the *time synchronization* inaccuracy NTP experimentally demonstrates on the order of milliseconds, as we will show below.

3.2 Error Budget Analysis of NTP

Under the assumption that $T_a = t$ and $T_b = t + \theta_0$ during the packet exchange, the NTP algorithm should calculate θ_0. From the diagram above, we can easily derive the following:

$$T_2 = T_1 + x_1 + \theta_0 \qquad (3)$$

$$T_4 = T_3 + x_2 - \theta_0 \qquad (4)$$

$$\delta = T_4 - T_1 - (T_3 - T_2) = T_3 + x_2 - \theta_0 - T_1 - (T_3 - T_1 - x_1 - \theta_0) = x_1 + x_2 \qquad (5)$$

$$\theta = \frac{T_2 - T_1 + T_3 - T_4}{2} = \frac{T_1 + x_1 + \theta_0 - T_1 + T_3 - T_3 - x_2 + \theta_0}{2} = \theta_0 + \frac{x_1 - x_2}{2} \qquad (6)$$

From the above simple calculation in (6), it is evident that NTP provides a *biased* estimate θ of the true offset θ_0, due to asymmetry of the one way delays x_1, x_2 or more accurately stated, *due to the variation of the network delay (jitter).* In the next passage we will estimate the maximum error (and consequently the *confidence intervals*) of every measurement during the NTP algorithmic calculations and compare those confidence intervals with those taken into account in the NTP protocol. Our purpose is to justify the inaccuracy on the order of milliseconds NTP has experimentally showed. Before proceeding, we have to prove the following theorem:

3.2.1 Clock Reading Error Theorem: Whenever we make a system call requesting a timestamp, a reading error ε from the 'true' time occurs because of the limited resolution of the data structure that keeps the time and the inherent instability (due to thermal drift as explained in the previous chapter) of the clock. This error ε is bounded:

$$-\rho + f\tau \le \varepsilon \le f\tau \rightarrow \varepsilon \in [-\rho + \tau f, \tau f]$$

where ρ is the maximum resolution error, f is the maximum frequency skew error and τ the interval between subsequent clock readings.

Proof: The query of the clock at time *t* results in the answer:

$$T(t) = t + r + fdt,$$

where r represents the resolution error and f the frequency skew of the clock.

Since the limited resolution of every computer system leads to truncation, we can model r as a random variable taking negative values in the interval [-ρ, 0], *uniformly* distributed. Therefore, $p_r(r)$=uniform(-ρ,0) (7).

The random variable *f* representing frequency errors of the clock can be modeled as a *Gaussian* distribution with zero mean and std σ. We can also assume that the gaussian distribution is a truncated distribution in the interval [-φ,φ] (i.e. the maximum frequency error is ± φ).

Therefore, $$T(t) = t + \varepsilon, \quad \varepsilon = r + fdt$$

(dt denotes the time interval between subsequent clock readings. We can treat dt as a constant $\tau \Rightarrow fdt = f\tau = \tau f$). τf is still a gaussian distribution with mean zero and std = τ σ and we can say that it is bounded in the interval [-τφ,τφ],

$$\tau f \in [-\tau\phi, \tau\phi] \quad (8).$$

$$\varepsilon = r + \tau f \Rightarrow p_e = p_r * p_{\tau f} \quad (9)$$

(where * denotes convolution). At this point we made also the assumption that reading the r and f are independent variables - we don't have any reason not to -.

$$\text{From } (7),(8),(9) \Rightarrow \varepsilon \in [-\rho + \min(\tau f), 0 + \max(\tau f)] \quad (10).$$

At this point we can make another assumption: the frequency error occurs very slowly at every clock with small changes, so

$$\min(\tau f) \approx \max(\tau f) \approx \tau f \quad (11).$$

$$(10),(11) \Rightarrow \varepsilon \in [-\rho + \tau f, \tau f]$$

Q.E.D.

Therefore the reading error $\varepsilon \in [-\rho+\tau f, \tau f]$, which occurs *each time a timestamp is requested from the pc clock!* We are now ready to proceed to the next theorem:

3.2.2 Maximum NTP Error Theorem: The maximum error in the offset calculation using NTP is on the order of

$$\max(\rho_\alpha, \rho_\beta) + \frac{\phi_\beta(T_3 - T_2) + \phi_\alpha(T_4 - T_1)}{2} + \frac{|x_1 - x_2|}{2}$$

where ρ_α, ρ_β are the resolution errors of clock A, B respectively and $\varphi_\alpha, \varphi_\beta$ are their maximum frequency errors.

Proof: At the previous theorem we calculated the inherent error every time we retrieve a timestamp. From (2) we have $\theta = \frac{T_2 - T_1 + T_3 - T_4}{2} = \frac{a+b}{2}$ so, every time we compute (a+b)/2 we make an error which belongs to the following interval:

$$\varepsilon_\vartheta \in \left[\frac{\min(\varepsilon_{T2}) - \max(\varepsilon_{T1}) + \min(\varepsilon_{T3}) - \max(\varepsilon_{T4})}{2}, \frac{\max(\varepsilon_{T2}) - \min(\varepsilon_{T1}) + \max(\varepsilon_{T3}) - \min(\varepsilon_{T4})}{2}\right]$$

$$= \left[\frac{-\rho_\beta - 0 - \rho_\beta + (T_3 - T_2)f_\beta - (T_4 - T_1)f_\alpha}{2}, \frac{0 + \rho_\alpha + (T_3 - T_2)f_b + \rho_\alpha - (T_4 - T_1)f_a}{2}\right] \Rightarrow$$

$$\varepsilon_\vartheta \in \left[-\rho_\beta, \rho_\alpha\right] + \frac{(T_3 - T_2)f_\beta - (T_4 - T_1)f_\alpha}{2} \quad (11)$$

Therefore, from (11) we can compute the maximum inherent error in calculating the quantity (a+b)/2 which is

$$\varepsilon_\vartheta'' = \max(\rho_\alpha, \rho_\beta) + \frac{(T_3 - T_2)\phi_\beta + (T_4 - T_1)\phi_\alpha}{2} \quad (12)$$

We proved above that $\theta_0 = \frac{a+b}{2} + \frac{x_2 - x_1}{2} = \theta + \frac{x_2 - x_1}{2} \Rightarrow$

$$\theta - \varepsilon_\vartheta'' - \frac{|x_2 - x_1|}{2} \le \theta_0 \le \theta + \varepsilon_\vartheta'' + \frac{|x_2 - x_1|}{2} \quad (13)$$

From (13) we conclude that the maximum error in calculating the real offset is

$$\varepsilon_{\vartheta}''+\frac{|x_2-x_1|}{2}=\max(\rho_\alpha,\rho_\beta)+\frac{(T_3-T_2)\phi_\beta+(T_4-T_1)\phi_\alpha}{2}+\frac{|x_2-x_1|}{2}$$

Q.E.D.

The above simple theorem nicely summarizes all the issues involved in *time synchronization* in packet switched network architectures:

a) **limited resolution due to the bounded size of software buffers needed to keep time** $\{\max(\rho_\alpha,\rho_\beta)$ term$\}$

b) clock update frequency differences (**skew**) due to physical causes (thermal drift) $\{\frac{(T_3-T_2)\phi_\beta+(T_4-T_1)\phi_\alpha}{2}$ term$\}$ and

c) network delay variance (**jitter**) $\{\frac{|x_2-x_1|}{2}$ term$\}$.

A good time keeping scheme should take into account all the above factors. As we have already underlined, NTP ignores the last network term. This is natural to understand as NTP was first introduced very early when IP networks where not as chaotic as they are today and of course the protocol specification required sub-second accuracy.

Specifically, NTP for every estimate of θ ((a+b)/2) assigns a confidence interval of $\lambda=\varepsilon_{\vartheta}''+\frac{\delta+\varepsilon_\delta}{2}$ (14), where ε_δ is the measurement error in the round trip time δ, based on (1).

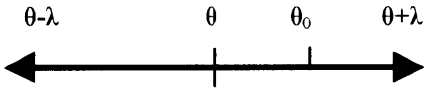


Figure 3.6 Offset Confidence Interval in NTP

Therefore, according to NTP, the true value θ_0 should be around θ by an interval

$$\lambda=\varepsilon_{\vartheta}''+\frac{\delta+\varepsilon_\delta}{2}$$ (as shown above).

Since NTP daemon client makes multiple measurements from different peers, there are multiple estimates of θ_0 with confidence intervals according to (14). A modified Marzullo and Owicki [Marzull1985] algorithm is used to find the appropriate interval containing the correct time given the confidence intervals of a set of measurements. Nevertheless, in NTP those confidence intervals are bigger than they should be by a quantity

$$\frac{\delta - |x_1 - x_2|}{2} = \frac{(x_1 + x_2) - |x_1 - x_2|}{2} \qquad (15)$$

as can be seen from what have been said so far. In other words, NTP takes into account the round trip time $\delta = (x_1 + x_2)$ instead of jitter $|x_1 - x_2|$. With this last observation, we can justify the observed inaccuracy (even on the order of msecs) NTP demonstrates. In a following section we will provide experimental verification of that inaccuracy.

Another important question is lurking here: if network jitter is the prominent reason behind NTP errors, are there any bounds for delay variation (jitter) and subsequently for NTP error? What about resolution and frequency skew errors?

- *Resolution error*: NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0^h on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits. The precision of this representation is about 200 picoseconds.
- *Frequency skew error*: As we showed in chapter 2, time uncertainty of 10^{-8} down to 10^{-15} is physically feasible. Common quartz crystal oscillators can have 1 ppm frequency skew, which is a reasonable bound, imposing at the same time a need for a frequent correction mechanism.
- *Delay variance (jitter):* one way delays are limited, therefore jitter should be limited.

Variance due to propagation and transmission delays can be zeroed if the communication path form the NTP client towards the NTP server is exactly the same with that from the server to the client. Nevertheless, queuing delays exist due to cross traffic (*the network* is always a shared medium) and are asymmetric (TCP traffic for example is asymmetric). However they are bounded since router queues are bounded. An interesting theoretical result from renewal processes and queuing theory summarizing the above intuitive remarks: jitter in a fully utilized network (utilization ρ_t close to 1) is bounded by an amount which is a function of load ρ of cross traffic and cross traffic variance σ_B^2 (and independent of the number of intermediate nodes) [Matragi1997]:

$$delay\ \mathrm{var}iance(jitter) = \frac{\frac{1}{1-\rho}\sigma_B^2}{1-\rho^2}, \qquad \rho_t = \rho_{NTP} + \rho \to 1$$

Therefore, resolution error, frequency skew and jitter as well are bounded so the overall NTP error can be bounded according to the above observations. Quantifying this error is difficult, since

quantifying jitter in current Internet is difficult: that would require to characterize the statistical properties of cross traffic against the NTP flow and calculate the jitter according to the above formulation. Unfortunately the chaotic behavior of today's Internet makes extremely difficult such adventurous theoretical characterization attempts. However, we can evaluate NTP error experimentally, in operating conditions in the current Internet. This is done in the following passage.

3.3 Experimental Evaluation of NTP

NTP client daemon `xntpd` is shipped with a set of utility tools that give the capability of querying remote NTP servers for their lower stratum synchronization sources (`xntpdc` client program) or the time offset between the client and a specific NTP server (`ntpdate` client program).

The NTP distribution can be downloaded from http://www.eecis.udel.edu/~ntp/index.html. The unix distribution 3-5.93 (for NTP version 3) was easily installed and configured in a Linux 6.2 machine and synchronized to stratum 2 NTP servers of Media Lab and stratum 1 of MIT. Using the utility programs mentioned above, it is feasible to discover the spanning tree of the Internet NTP servers as well as the time offset between the client and whichever NTP server. This offset is never zero due to all the erroneous assumptions of NTP algorithm explained thoroughly in the previous section, as well as because of physical limitations.

NTP runs over UDP so there might be lost packets, therefore multiple sampling packets should be addressed to a server in order to query its status, list of peers, source of time (if it is stratum 1) etc. Moreover, whenever the NTP client discovers offset more than 128msec to a server, it refuses to synchronize. The following results take that into account and report offset statistics (mean, media, and standard deviation) between "working associations" of the NTP protocol. The two most recent large scale NTP surveys were conducted by D. Mills, the author of NTP protocol specification, in 1997 [Mills1997] and Media Lab researcher N. Minar in 1999 [Minar1999]. The results are summarized in the following table 3.5:

Table 3.5 Experimental NTP error statistics

	1999 (Minar)	1997 (Mills)
Offset Mean	8.2 msec	28.7 msec
Offset Std	18ms	Not reported
Offset Median	1.8 msec	20.1 msec
Number of Stratum 1 servers	957	220
Stratum 2	26,830	4,438
Stratum 3	85,332	6,591
Stratum 4	38,339	2,254
Stratum 5	7,134	317
Total number of servers queried	175,527	38,722

We can see that NTP reported offset is on order of milliseconds. If the time synchronization algorithm was perfect, this offset variable should converge to zero. The fact that this error persists in time, clearly demonstrates that the zero jitter assumption of the algorithm does not hold. Mills in his 1997 paper observes "clock offset errors cannot be distinguished from asymmetric network delays. By design the errors due to asymmetric delays are bounded by half the roundtrip time" which is exactly what we showed in the previous section. The msec inaccuracy shown above is critical in sensor network beam-forming applications, since an error in time reference on the order of 5 msecs is translated in an error of 1500 km in location estimation using RF waveforms!

The offset error is affected by the stability of the clocks, as we showed above therefore the reduced offset error of 1999 experiment compared to the 1997, could be justified by the hardware advances in computer clocks. A more convincing explanation stems from the increased number of NTP servers (by one order of magnitude) and from the reduced roundtrip delay of NTP packets (reported in [Mills1997], [Minar1999]), because of higher bandwidth links and more dense NTP topology. Therefore, the confidence intervals for every offset measurement are smaller, as explained in the previous section, leading to less erroneous offset estimates.

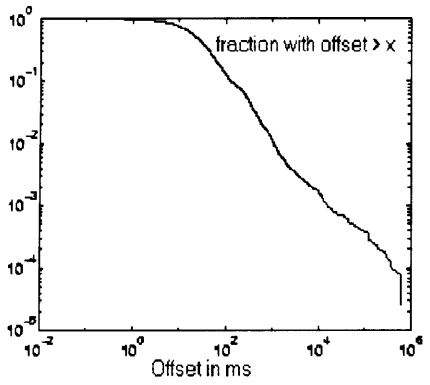


Figure 3.7 Cumulative distribution function of the observed offset at the 97 survey [Mills1997]

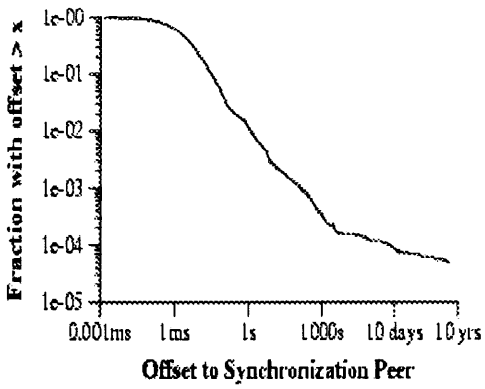


Figure 3.8 Cumulative distribution function of the observed offset at the 99 survey [Minar1999]

The graphs directly above are the log/log plot of the cumulative distribution function (CDF) of the observed time offset. The Y-axis shows the fraction of hosts whose value is greater than its position on the X-axis. For example, only 10% of the hosts in the 1999 experiment have an offset more than 20ms, and only 1% have offset greater than 1s. The shape of both curves is similar. However, the whole distribution of 1999 has shifted to the left towards shorter offsets. There is also a longer tail, suggesting a small but noticeable fraction of hosts that have pathologically incorrect clocks (1 in 1000 are over 100 seconds off).

It is evident that a sub-millisecond accuracy *time-synchronization* algorithm should take into account the network jitter and treat it as noise to the time synchronization scheme. Of course physical limitations would still affect the performance (thermodynamic drift instability), but this could be corrected by running the synchronization algorithm more frequently.

4 Related Work

In the previous chapter we analyzed NTP and identified network jitter as one of the most prominent causes for NTP time synchronization inaccuracy. Before proceeding to the proposed solutions we will briefly review related work in time synchronization algorithms for packet switched networks.

4.1 Paxson Algorithms

V. Paxson at his Ph.D thesis [Paxson1997] instrumented a large-scale End-to-End Internet-dynamics measurement project, by setting up suitable ping Network Probe Daemons (NPD) in a large number of remote hosts. He didn't use external time-synchronization schemes like GPS or WWVF, because he wanted to facilitate as many remote participating sites as possible, therefore installing additional time-synchronization modules would be impractical. Therefore, Paxson utilized his own time synchronization algorithms.

For offset calculation between the clock of the sender s and the receiver r, Paxson provided the following formula:

$$\theta \equiv \Delta C_{r,s} \approx \frac{\delta\tilde{\mathrm{T}}_{p_s} - \delta\tilde{\mathrm{T}}_{p_r}}{2} \qquad (1) \text{ where,}$$

$\delta\tilde{\mathrm{T}}_{p_s}$, $\delta\tilde{\mathrm{T}}_{p_r}$ are the minimal one way delays observed across the path from s to r and r to s, respectively. The above formula makes the assumption, exactly like NTP, that the one way delays x_1,x_2 from s to r and vice versa are the same, therefore (1) gives a poor estimate of θ. This was recognized by Paxson who states that the inequality of x_1,x_2 "requires us to make only casual use of the estimate for $\Delta C_{r,s}$".

Since he used bi-directional traffic and he was interested in measurements of inter-arrival time of packets at the receiver (and at the sender because of the TCP ACK packets at the reverse path) he mainly focused on the estimation of frequency skew φ between the sender and receiver's clock $(dT = d(\phi t_2 + \theta - \phi t_1 - \theta) = \phi dt)$. He made the observation that for a set of

measurements $O_1 = T_r^1 - T_s^1, O_2 = T_r^2 - T_s^2$, where T_r^i, T_s^i are the timestamps at the arrival of packet i at the receiver (according to receiver's clock) and at the departure of the i packet at the sender (according to sender's clock) the following relationship holds:

$$O_2 = O_1 + (\eta - 1)(T_s^2 - T_s^1) \qquad (2),$$ where η is the frequency skew of clock in r relative to clock in s.

Therefore, there is a linear function $O_j = O(T_s^j)$ (2*), with slope (1-η), so samples (T_s^j, O_j) are enough so as to estimate the offset η. Unfortunately, (2) is an approximation since it is based on the zero jitter assumption: for frequency skew $\eta \equiv \phi$ we make the assumption that $T_s(t) = t, T_r(t) = \phi t + \theta$, therefore:

$$O_1 = \phi T_s^1 + \theta + x_1 - T_s^1 \qquad (3)$$
$$O_2 = \phi T_s^2 + \theta + x_1' - T_s^2 \qquad (4) \Rightarrow$$
$$O_2 - O_1 = (\phi - 1)(T_s^2 - T_s^1) + x_1' - x_1 \qquad (5)$$

where x_1', x_1 are the one way delays across the path from s to r, therefore the variation of the difference $x_1' - x_1$ (jitter) is pure noise for the above estimation procedure. Paxson, after using statistical heuristics to remove the noise, he applies linear regression so as to estimate φ out of the de-noised one way measurements, based on the linear relationship in (2), (2*).

4.2 Moon algorithm

In [Moon1999] a linear programming technique was presented and compared to the Paxson algorithm with slightly better frequency skew-estimation results. However, there was no offset estimation scheme. We are going to incorporate a linear optimization technique, as well, however our derivation is different and more intuitive than that in [Moon1999], indicating why *linear programming is more suitable for frequency skew estimation than linear regression*, in packet switched network. The performance of the algorithm will be tested having jitter power (variance of queuing delays) as input parameter into the simulation environment rather than just queuing delay as in [Moon1999]. We will provide an offset estimation algorithm as well.

4.3 Poor Algorithm

In [Poor1999] an attempt for time synchronization is made in a rather simplistic way.

According to that algorithm, the clock in the receiver is being disciplined according to the average of sender's timestamp and receiver's timestamp. No analysis of this algorithm is provided, even though simulations show msec time synchronization error. This model is an over-simplification, since it does not account for frequency skew between the two clocks and of course, completely neglects propagation, transmission and queuing delays which in normal cases are never zero. We will analyze this algorithm: If $T_s(t) = t, T_r(t) = \phi t + \theta$,

$$T_r(t) = \frac{T_r + T_s}{2} \qquad (6)$$

$$(6) \Rightarrow T_r(t) = T_r(T_s + x_1) = \frac{\phi T_s + \theta + \phi x_1 + T_s}{2} \qquad (7)$$

now, if we assume $\phi \approx 1, (7) \Rightarrow T_r(T_s + x_1) = T_s + \frac{\theta + x_1}{2}$ (8),

the offset of the two clocks becomes $T_s + \frac{\theta + x_1}{2} - (T_s + x_1) = \frac{\theta - x_1}{2}$ (9).

If the experiment is repeated, then assuming that frequency skew remains $\phi = 1$, the offset becomes

$$\frac{\frac{\theta - x_1}{2} - x_1'}{2} \qquad (10)$$

where x_1' is the new one way delay due to propagation, transmission and queuing. For mobile wireless nodes, apart from the queuing delay variation, we have propagation delay variation due to the position change of the nodes, therefore jitter power increases. Generalizing the above, if n experiments are made, the offset error becomes

$$\frac{\theta - x_1 - 2x_1' - 4x_1'' - \ldots - 2^{n-1} x_1^*}{2^n} \qquad (11)$$

From (11) we can see that the algorithm never reaches zero. Generally speaking, this algorithm as can be seen from (11) can not be guaranteed to work.

4.4 MPEG-2 algorithm

In the MPEG community, there is strong need for a synchronization mechanism to lock the receiver's time base to the clock of the remote encoder so as consistent decoding and play-out of

audio-video units can be performed at the receiver. Clock skews and drifts at the encoder or decoder seriously affect the video play-out, so in the MPEG standard there are stringent requirements about maximum allowable clock skew, or drift. For example, MPEG defines the maximum frequency skew of the encoder's clock to 20 ppm =540 Hz/ 27 MHz [MPEG1994].

A software phased lock loop is used to estimate the frequency skew between encoder's and decoder's clocks. A software pll is a non-linear filter, imposing practical difficulties on the estimation of its optimal parameters. Its efficiency will be compared with our original proposal. Even though the pll, utilizes phases differences to estimate frequency differences, accurate offset estimation is not feasible since the samples include both offset and one way delays. Therefore, a more accurate offset technique is needed. This is also provided in the following chapter.

4.5 Other algorithms

In the highly referenced work of [Singh1994], a linear estimator is used for frequency skew estimation. We are going to provide a linear estimator, which is provably the optimal linear estimator of frequency skew in the presence of network jitter.

Finally, in [Troxel1994] a running average of clock adjustments over a series of NTP experiments is used so as to predict the NTP error. We will pursuit a more mathematically structured work.

4.5.1 The old (or new?) approaches

Our goal is to provide an End-to-End solution to the time synchronization problem over packet switched networks. This is the way to scale the proposal in today's networks. If we had the opportunity to define a new network architecture, then we should force the network to zero the delay variance for NTP flows. This could be done in many ways:

- Give first priority to NTP packets so they are placed at the beginning or at the end of the queue in each intermediate router, so as to endure constant delay. Constant delay means zero delay variance (jitter). Another way would be to increase or decrease a time header of the NTP packet proportionally to the delay the packet experienced at that router.
- Delay each NTP packet a specific amount of time so as the packet across the path to the destination sustains a constant delay. The above ideas were proposed at the early 90's [Ferrari1990], [Verma1991] however they didn't become very popular due to their impractical requirements for high-speed network routers.

Nevertheless, if time synchronization and keeping are the most vital services of a new network architecture, like in *Myriad Networks*, then routers modifications should be adopted at the expense of network bandwidth/capacity.

In the next section we are going to focus on our solutions for End-to-End time synchronization mechanisms.

5 End to End Solutions for Time Synchronization

In the previous chapters we identified the network jitter as the primal source for NTP inaccuracy on the order of milliseconds. This led to the idea to explicitly deal with jitter in a new time synchronization algorithm where the time source packets are the useful information and network delay variance (jitter) of cross traffic is considered noise for the time-sync scheme. In that case, it would be possible to incorporate standard de-noising signal processing techniques so as to extract the useful timing information out of noise (network jitter).

We will concentrate on end to end approaches, meaning that we will focus on the algorithms that should be implemented at the end nodes so as to disseminate the correct time, without needing to change the (intermediate) network. This is the hard and interesting problem to attack. Any proposed solutions could be directly applied to the today's Internet. Of course, tuning/modifying the way Network Nodes are layered so as to reduce jitter is also a valid suggestion, especially when a new Network architecture is being proposed: the Myriad Networks. However such an approach is easier to do and not very appealing since it can't quickly scale to today's Internet.

5.1 Working Assumptions

NTP algorithm does not treat explicitly frequency skew between the client and the NTP server clock, but estimates the frequency skew error as an overall offset error *("curse of modeling"*). Nevertheless, having a more accurate model for each clock can improve time synchronization performance, as we will show in the following sections. So, if the NTP server is the source of the "true time" the following relationship holds:

$$\text{NTP server: } T(t) = t \qquad (1)$$

$$\text{NTP client: } T(t) = \phi t + \theta \qquad (2)$$

where φ is the frequency skew between the two clocks and θ their offset. Observe that from (2), $T'(t) = \phi$ consistently with the definitions we gave in chapter 2. The frequency drift for clock in client B is considered $T''(t) = 0$. This is a reasonable assumption only for short periods of time. The algorithm should be re-run assuming a different frequency skew and offset after a time

interval in order to compensate for the modeling error of $T''(t) = 0$. This is an important note to remember, especially when the uncertainty t/δt of common quartz crystals used in personal computers is on the order of $10^{-6} \sim 10^{-8}$ (as we described in chapter 2) which means that an error of 1 nanosecond will happens in $10^{-9} * 10^{8} = 0.1$ sec = 100 msec. Therefore, the NTP should be re-run more frequently on the order of hundreds of msecs so as to compensate for nsec timing errors because of the instability of the local quartz resonators. Vice versa, more stable resonators should be incorporated in order to relax the frequency of NTP messages exchange.

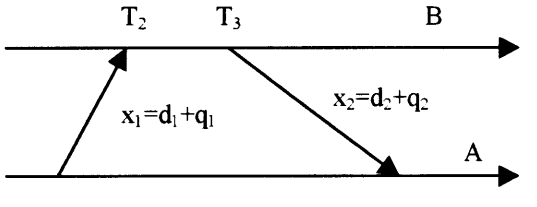


Figure 5.1
NTP Message exchange
(revised clock model)

x1, x2 are the forward and reverse path one way delays: $x_1 = d_1 + q_1, x_2 = d_2 + q_2$, where d_i is the propagation delay (the time needed for the firt bit to arrive at the destination) and the transmission delay (the time needed for the whole packet to reach the destination, after the first bit arrived) across the forward or reverse path. If the physical paths are the same from A to B and vice versa then we can assume that $d_1 \approx d_2$ (3). The quantities are not exactly the same, since processing time is involved in every intermediate node which is not a determenistic variable, nevertheless their difference should be small (again, if and only if the forward and reverse physical paths are the same).

q_1, q_2 are the queueing delays of the NTP packet across the forward (A ⇒ B) and reverse path (B ⇒ A) respectively, which are not the same due to the chaotic nature of cross traffic as explained in previous chapter. The variation of $q_1 - q_2$ (jitter) complicates the time synchronization task. Therefore, $q_1 \neq q_2 \Leftrightarrow x_1 \neq x_2$ (4).

5.1.1 Definition of the problem

The problem is stated accordingly: given samples of (T_1, t_2, t_3, T_4), where T_1, T_2 are measured according to the client's clock ($T=\varphi t+\theta$) and t_2, t_3 according to the server's clock (t), how can φ, θ be estimated given $jitter \Rightarrow x_1 \neq x_2$.

In the following section we are going to give three different end-to-end algorithms for the estimation of frequency skew φ and offset θ, providing better accuracy than NTP. With the clock model adopted above, we have the following relations, derived trivially from figure 5.2

$$T_1 = \phi t_1 + \theta \qquad (5)$$

$$t_2 = t_1 + d_1 + q_1 \qquad (6)$$

$$t_3 = t_3 \qquad (7)$$

$$T_4 = \phi t_3 + \theta + \phi(d_2 + q_2) \qquad (8)$$

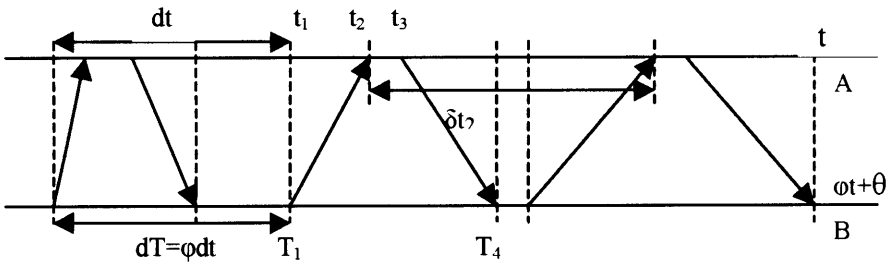


Figure 5.2 Packet departure from client B, every φdt

5.1.2 Jitter Modeling or ("the curse of modeling")

Since network jitter deteriorates the quality of every end2end time-synchronization algorithm, the obvious question emerges: what is the jitter probability distribution and how jitter correlation between subsequent measurements could be utilized in order to improve the time synchronization algorithm? Unfortunately, there is no obvious answer [Kim1999]. In chapter 3 we provided a formula about the jitter variance in high utilization links. If we define jitter as the inter-arrival time of a stream of packets sent periodically, it turns out that the jitter distribution for a large number of nodes is White Gaussian [Kim1999]. This observation basically states that one jitter sample can NOT be predicted by any combination of previous samples and therefore, we should not anticipate or exploit any correlation in jitter samples. Although this is not always correct, it basically forces us to adopt a worst case analysis of any proposed time synchronization technique, without any assumptions about jitter sample correlation. This is a common practice followed in telecommunication systems analysis, since it is general enough to cover all the possible cases. In any case, jitter energy (or noise energy) is the primal feature that affects the time-synch mechanism, not the probability distribution of jitter [Andreotti1995], [Landry1997].

Therefore, in the following analysis we are going to assume zero mean, additive white gaussian noise for the inter-arrival intervals of the periodic NTP stream. Specifically, we will assume that the queuing delays q_1, q_2 in $x_1 = d_1 + q_1, x_2 = d_2 + q_2$, are distributed according to |Q|, where the pdf of Q is a normal distribution N(0, σ^2). Therefore, Y=|Q| is *a half-normal distribution.* Assuming q_1, q_2 independent and identically distributed according to Y=|Q|, where Q is N(0,σ^2) we have:

$$E(q_i) = \sigma\sqrt{\frac{2}{\pi}}, \quad \sigma_{q_i}^2 = \sigma^2(1-\frac{2}{\pi}) \qquad (9)$$

$$\mathrm{E}(q_1 - q_2) = 0, \quad \sigma_{q_1-q_2}^2 = 2\sigma^2(1-\frac{2}{\pi}) \qquad (10)$$

The bounded variance of the jitter noise in (10) will be relaxed in the simulation experiment at the end of this chapter, where the queuing delays will be distributed according to fractional Brownian motion (FBM), providing a more chaotic nature.

5.2 Online End to End Methods for Time synchronization

Differentiating between frequency skew φ and clock offset is a trick that NTP doesn't do, therefore an improved time synchronization scheme is possible, given that computationally efficient algorithms are implemented so as to estimate φ and θ. We are providing two online algorithms which estimate φ,θ out of network jitter *noise* as samples of the NTP experiment are acquired and one offline approach which is utilized after a buffer of samples (T_1,t_2,t_3,T_4).

5.2.1 Kalman Filter approach

The idea here is extremely simple: if the NTP inter-departure intervals at the NTP client are constant (meaning that the NTP departure stream is periodic) then the inter-arrival times of NTP packets at the NTP server would also be constant if no network jitter noise was present. By comparing the inter-arrival and inter-departure intervals, we could estimate the frequency skew φ between the two clocks. Therefore, the problem now is to estimate dt (Figure 5.2) so as to estimate φ according to the formula (11):

$$\phi = \frac{dT}{dt} \qquad (11)$$

where dt is the de-noised inter-arrival interval at the NTP server and can be estimated by δt_2 from

two consecutive NTP packet experiments (see Figure 5.2). The Kalman filter will be used to de-noise measurements δt_2. As we said above the jitter distribution can not be known in advance, however the jitter energy can be estimated and that is the important parameter to be considered. Jitter energy can be estimated by the measured δt_2 which represents the *dispersion* [Dovrolis] of inter-arrival time of NTP packets at the NTP server. Given N consecutive NTP measurements $(T_{1i}, t_{2i}, t_{3i}, T_{4i}), i = 1..N$, jitter can be estimated according to the following formulas:

$$\sigma_{\delta t}^2 \approx \sigma_{q1}^2 + \sigma_{q2}^2 \qquad (12),$$

since $\delta t \propto x_2 + x_1' = q_2 + q_1' + d_2 + d_1$, d_1, d_2 are constants,

$$jitter \cong \text{var}iance(x_1 - x_2) \Rightarrow \sigma_J^2 = \sigma_{q1}^2 + \sigma_{q2}^2 \approx \sigma_{\delta t}^2 = \frac{\sum_{i=1}^{N-1} (t_2^{i+1} - t_2^i)^2}{N-2} \qquad (13),$$

It is important to notice that the above calculation assume that $t_{3i} - t_{2i} = const$ and the queuing delays are i.i.d. The first assumption can be realized algorithmically in a modified NTP scheme and the second assumption is again a reasonable worst case scenario assumption.

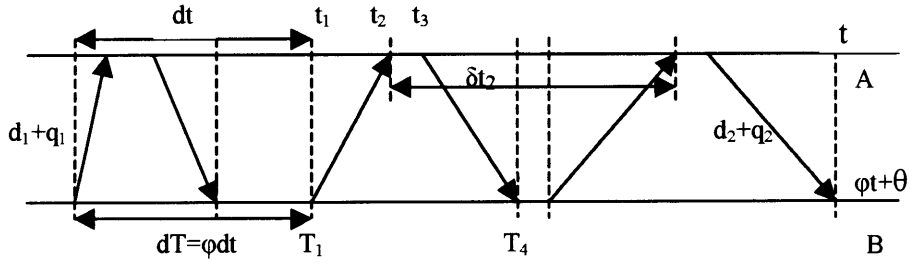


Figure 5.2b Packet departure from client B, every φdt. Time stamps T_1^n, T_4^n are according to B's erroneous clock and timestamps t_2^n, t_3^n are according to time source A's clock. In the presence of network jitter (variation of q1-q2), how can we compute φ,θ?

5.2.1.1 Kalman Filter Notation and its relationship with NTP

Following the figure above, the client B timestamps and sends the nth NTP packet at T_1^n according to each own clock. The NTP server timestamps the packet when it receives it at t_2^n

according to each own clock and sends it back at t_3^n, so as $t_3^n - t_2^n = const$. The client receives the packet at T_4^n. Given a series of N $n \in [1..N]$ NTP packets $(T_1^n, t_2^n, t_3^n, T_4^n)$ ($n \in [1..N]$), the following notation holds:

$dT = T$: the period with which the NTP client is sending NTP packets (according to each own clock).

dt : the period with which the NTP packets would arrive at the NTP server A, according to A's clock (given that they had been sent periodically from the client B) if NO network jitter was present. Therefore, $dT = \phi dt$ (11)

y : the observed difference of arrival times of two successive NTP packets at the NTP server. Because of the existence of network jitter (variation of queuing delays) $y \neq dt$.

Therefore, given two successive NTP packets $(T_1^{n+1}, t_2^{n+1}, t_3^{n+1}, T_4^{n+1}), (T_1^n, t_2^n, t_3^n, T_4^n)$ the following relationships should hold:

$$y_n = \delta t_n = t_2^{n+1} - t_2^n, n = 1..N-1 \qquad (14)$$

$$y_n = dt_n + v_n \qquad (15)$$

$$dt_n = dt_{n-1} = dt \qquad (16)$$

In (15), we explicitly treat jitter as noise v that affects the periodicity of the inter-arrivals of NTP packets at the NTP server A. In (16), we simply state, that the inter-arrival time should be constant from one measurement to the next, when jitter is zero. This is obvious, since packets sent periodically, following the same path and bearing zero or constant queuing delay should arrive also periodically at the receiver.

In the standard Kalman Filter equations [Orfanidis1988, p.188] the problem is defined as finding x_n from y_n where x_n comes from a first order Markov model:

$$x_{n+1} = ax_n + w_n \quad \text{(state model)}$$

with $|a| \leq 1$. The observation signal y_n is related to x_n by

$$y_{n+1} = cx_n + v_n \quad \text{(measurement model)}$$

and it is further assumed that the state and measurement noises w_n, v_n are zero mean, mutually uncorrelated, white noises of variances Q and R respectively:

$$E[w_n w_i] = Q\delta_{ni}, \qquad E[v_n v_i] = R\delta_{ni}, \qquad E[w_n v_i] = 0, \qquad \delta_{ni} = \begin{pmatrix} 1, n = i \\ 0, n \neq i \end{pmatrix}$$

Our only deviation from the above modeling, is only at the state model: we made the assumption that $w_n = Q = 0$, since we are interested on how to remove network jitter noise and we made the assumption that A is the source of true time (therefore node A is equipped with a perfect clock). Practically, w_n has a non-zero variability due to the inherent instabilities of clocks as explained in chapter 2 as well as because of time uncertainties in multithreaded operating systems. In future work, we are going to quantify this additional noise in the time synchronization process. Therefore, in our model:

$$a = c = 1,$$
$$Q = 0,$$
$$R = jitter \cong \text{var}iance(x_1 - x_2) = \sigma_J^2 = \sigma_{q1}^2 + \sigma_{q2}^2 \approx \sigma_{\delta t}^2 = \frac{\sum_{i=1}^{N-1}(t_2^{i+1} - t_2^i)^2}{N-2}$$

The following Kalman filter algorithm was tested [Orfanidis1988], where R, Q are given from the above formulas and the notation

ψ' means the estimate of ψ and

$\psi_{n/n-1}, \psi_{n/n}$ means the estimate of ψ at time n having measurements up to time n-1, n respectively.

5.2.1.2 Kalman filter procedure

0. Initialize $n = 0, x'_{0/-1}, P_{0/-1} = E[x_0^2]$. P is the mean square error between the prediction and we will have a short discussion about P, G after outlining the procedure. We are

inclined to set $P_{0/-1} = \varepsilon$, according to the nominal stability in ppm of the NTP server clock. However, in the experiments below we set $P_{0/-1} = T^2, x'_{0/-1} = T$.

1. At time n compute $y'_{n/n-1} = cx'_{n/n-1}$, $\varepsilon_n = y_n - y'_{n/n-1}$, $G_n = \frac{cP_{n/n-1}}{R + cP_{n/n-1}}$
2. Correct the predicted estimate $x'_{n/n} = x'_{n/n-1} + G_n\varepsilon_n$, and compute the mean square error $P_{n/n} = P_{n/n-1} - G_nP_{n/n-1}$
3. Predict the next estimate $x'_{n+1/n} = x'_{n/n}$ and compute the mean square prediction error $P_{n+1/n} = P_{n/n} + Q$
4. Go to the next instant n=n+1.

The above filter algorithm needs only one estimate of the noise (jitter) power R, which is computed before the above steps. This observation could classify the kalman filter approach at the offline approaches, however we should have in mind that the exact same formulation can be used in time varying models where $R=R_n$ is updated at each step. ***Remarkably, Kalman filtering gives the best estimate of $\vec{x}(n)$ from the history of $\vec{y}(n)$ that can be made by a linear estimator; it cannot be improved by analyzing the entire data set off-line*** [Gershenfeld1999].

The optimal predictor $x'_{x+1/n}$ satisfies the Kalman filtering equation

$$x'_{n+1/n} = ax'_{n/n} = a(x'_{n/n-1} + G_ne_n) = ax'_{n/n-1} + aG_n(y_n - cx'_{n/n-1})$$

(in our case $a = c = 1$)

It can be proved that $P_{n+1/n} = a^2P_{n/n} + Q$ where

$$P_{n+1/n} = E[e^2{}_{n+1/n}], \quad P_{n/n} = E[e^2{}_{n/n}] \text{ and}$$

$$e_{n+1/n} = x_{n+1} - x'_{n+1/n}, \quad e_{n/n} = x_n - x'_{n/n}$$

If we study carefully the above types, we can see that the kalman filter at each step tries to predict not only the state variable x_n but also the observed parameter y_n. Then, the algorithm corrects each estimate of $x'_{n/n}$, proportionally to the error $e_n = y_n - cx'_{n/n-1}$ with a gain factor G_n which depends on the mean square prediction error $P_{n+1/n}$. This specific interplay between what

we observe (y), what we predict to observe (y') and what we are trying to estimate (x) gives to the kalman filtering method its remarkable efficiency.

Therefore the above algorithm gives at least theoretically, an optimum way to extract the frquency skew $\phi = \frac{T}{x_n}$, out of noised samples $y_n = t_2^{n+1} - t_2^n$. Practically, the algorithm could suffer from inaccurate jitter energy estimation (based on (13)).

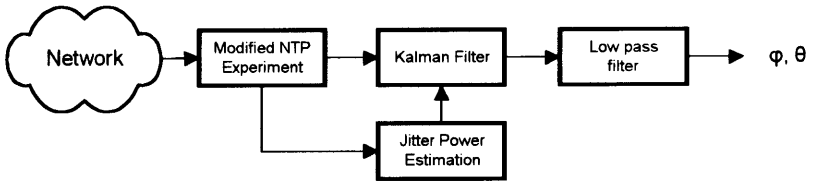


Figure 5.3 The Kalman Filter Approach

5.2.1.3 Experimental Results

Using half-normal distribution for the queuing delays, $d_1=d_2=$ 40 msec, $t_3-t_2=10$ msec =const, dT=200 msec = 0.2 sec (which corresponds to a frequent correction mechanism so as to compensate for errors of 1 nsec happening, on average, every 1 nsec 10^8 =0.1 sec for a clock with Q on the order of 10^8 or uncertainty of 0.01 PPM, which is typical for TCXO clocks as explained in chapter 2). Our goal is to test the time synchronization algorithm for different network conditions (and therefore jitter values) and different frequency skews and offsets due to physical causes. We define the following measure:

$$SNR = 10\log\frac{dT^2}{\sigma_J^2} \qquad (17)$$

which reflects how frequently we run our time synchronization algorithm versus the network jitter energy. dT cannot be indefinitely big, since in that case our initial assumption about zero drift does not hold. Moreover, a time synchronization algorithm which runs infrequently, is useless for obvious reasons. The frequency skew φ was set in the range 100 ppm to 10^4 ppm with one extreme case of φ=2 (δφ=1, δφ/φ=1=10^6 ppm). Jitter standard deviation was set on the order of msecs with SNR (dT=200msec) from 6 dB to 46 dB.

The results in terms of frequency skew estimation were very satisfying. For SNR less than 26 db the error was less than 100 ppm. For smaller values the error could practically reach zero or close to zero ($10^{-8} - 10^{-10}$ ppm). The figures below show a few indicative measurements. Another very satisfying result was the fact that the filter always locked to a steady state after approximately 3-5 seconds (15-25 measurements) which is faster than the software pll filter described in the following section.

ϕ	θ	Jitter std (msec)	$\tilde{\phi}$	$\tilde{\theta}$	Kalman error (msec)	NTP error (msec)
1.0001	1	1	1.0001	1.54	0.07	3.98
1.0001	1	10	1.0000	3.18	0.3	3.1
1.0001	1	100	1.0014	30	7.0	14
1.001	1	1	1.001	0.47	0.1	23
1.001	1	10	1.001	1.8	0.28	20
1.001	1	100	1.0012	3.6	0.26	61
1.01	1	1	1.01	1.7	0.3	229
1.01	1	10	1.0098	7.18	2.0129	190
1.01	1	100	1.0101	0.74	1.3	445
1.01	1	100 (asym)	1.0092	23	5	546
2	1	100	1.9946	75	9.7	26000

Table 5.4 Experimental Results for the Kalman filter algorithm

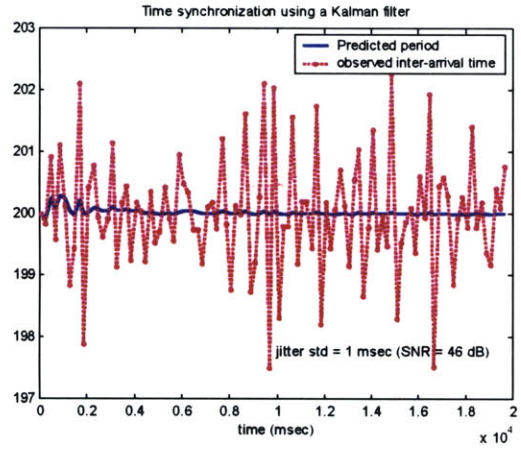


Figure 5.5 Kalman filter Method for skew $\Delta\varphi$=0.0001 (100 ppm) and jitter std =1 msec

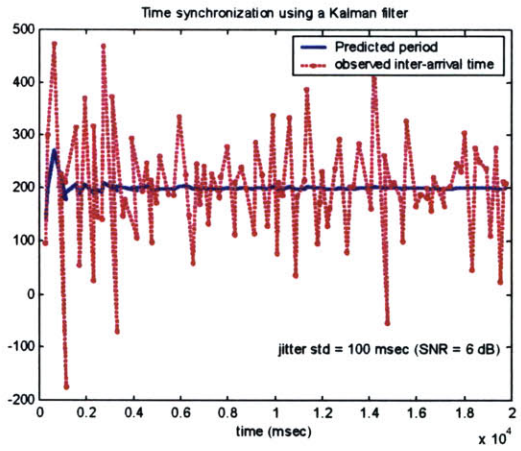


Figure 5.6 Kalman filter Method for skew $\Delta\varphi$=0.0001 (100 ppm) and jitter std =100 msec

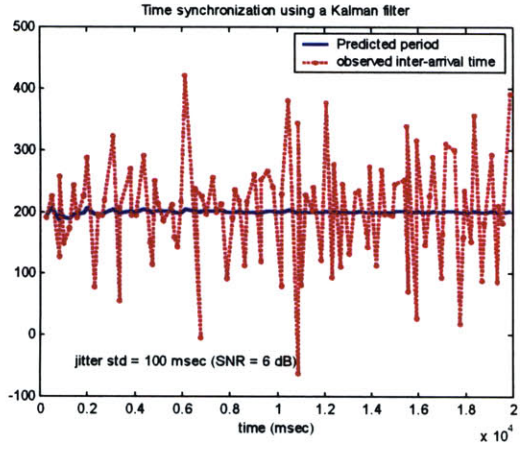


Figure 5.7 Kalman filter Method for skew $\Delta\varphi$=0.001 (1000 ppm) and jitter std =100 msec

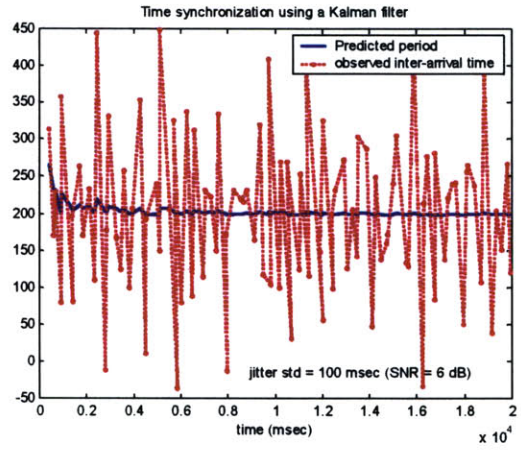


Figure 5.8 Kalman filter Method for skew $\Delta\varphi$=0.01 (10000 ppm) and jitter std =100 msec

We should clarify the way we extract the estimate of dt after the filter has reached the steady state condition: it is clear that a low pass filter is needed. In the plots above, a simple average of the last 20 measurements was used. If we wanted to do more, we could implement a second kalman filter on the steady state output of the first one. Nevertheless, the average filter is simple and efficient.

Another useful observation stems from equation (15):

$$\frac{1}{N_a}\sum_{j=0}^{N_a-1} y_{n-j} = dt + \frac{1}{N_a}\sum_{j=0}^{N_a-1} v_{n-j} \qquad (17)$$

So, instead of working on the samples y_n, we should work on the samples a_n as the variance

$$a_n = \frac{1}{N_a}\sum_{j=0}^{N_a-1} y_{n-j}$$

of noise is being decreased by a factor of N, improving the efficiency of the algorithm for smaller SNR values. Accordingly we should modify formula (13) (dividing by a factor of N). We leave this observation for future work, as it will improve the results quantitatively, not qualitatively. In any case however, it is a useful remark.

Before comparing the kalman filtering approach with what NTP calculates, we should complete the time synchronization scheme: provide a mechanism for the estimation of offset θ, after having a good estimate of the frequency skew φ from the output of the kalman filter. The offset calculation is difficult since it is added to the queueing delays which are unknown to the receiver. From Figure 5.2 we have the following relationships:

$$T_1 = \phi t_2 + \theta - \phi x_1 = \phi t_2 + \theta - \phi(d_1 + q_1) \Rightarrow T_1 - \phi t_2 = \theta - \phi(d_1 + q_1) \qquad (18)$$

$$T_4 = \phi t_3 + \theta + \phi(d_2 + q_2) \Rightarrow T_4 - \phi t_3 = \theta + \phi(d_2 + q_2) \qquad (19)$$

where T_1, t_2, t_3, T_4 are known from the NTP packet experiment. Provided that the same path from A to B and vice versa is kept during the during the experiment, equations (18), (19) have 3 unknowns: θ, x_1, x_2, because $\phi \approx \hat{\phi}$, where $\hat{\phi}$ is provided from the kalman filter algorithm. Therefore, we need to think a little bit more, so as to gain another linear independent (to (18), (19)) equation. We know that the queuing delays are not negative, therefore:

$$(18) \xrightarrow{q \geq 0} T_1 - \phi t_2 \leq \theta - \phi d_1 \qquad (20)$$

$$(19) \xrightarrow{q \geq 0} T_4 - \phi t_3 \geq \theta + \phi d_2 \qquad (21)$$

Therefore from the set of NTP samples $T_1^j, t_2^j, t_3^j, T_4^j$ we should pick the following:

$$j = \arg\max_n (T_1^n - \hat{\phi} t_2^n), \forall n \in [1, N] \qquad (22)$$

$$i = \arg\min_n (T_4^n - \hat{\phi} t_3^n), \forall n \in [1, N] \qquad (23)$$

over all the N NTP samples, hoping that these values will correspond to packets with no queuing delays and therefore:

$$a = T_1^j - \hat{\phi}t_2^j = \theta - \phi d_1 \qquad (24)$$

$$b = T_4^i - \hat{\phi}t_3^i = \theta - \phi d_2 \qquad (25)$$

of course q_1, q_2 won't be in general zero, especially in high network utilization conditions. However, this is not disappointing, since we are looking for ***just one*** packet/NTP sample measurement which is "lucky" enough to experience zero queuing delay across the forward path and ***just one*** "lucky" packet/NTP sample across the reverse path. The error in offset calculation will be influenced by the validity of this assumption. If the the forward and reverse physical path are the same then $d_1 = d_2$ (26). That can be realized by *source routing* the NTP packets alongside the forward and reverse path between the client and the NTP time-server. From (24), (25), (26) the offset estimation formula becomes:

$$\theta = \frac{T_1^j - \hat{\phi}t_2^j + T_4^i - \hat{\phi}t_3^i}{2} \qquad (27), where$$

$$j = \arg\max_n (T_1^n - \hat{\phi}t_2^n), \forall n \in [1, N]$$

$$i = \arg\min_n (T_4^n - \hat{\phi}t_3^n), \forall n \in [1, N]$$

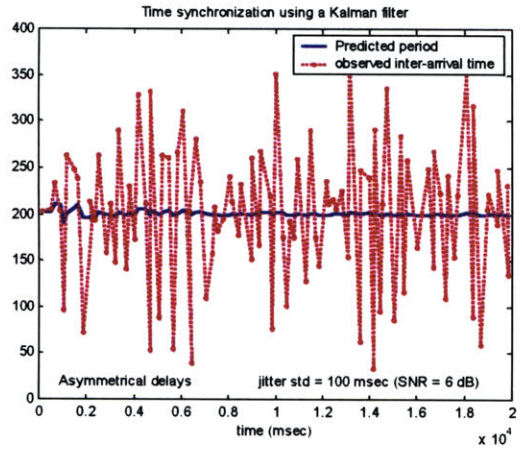


Figure 5.9 Kalman filter Method for skew $\Delta\varphi$=0.01 (10000 ppm) and assymetric jitter std =100 msec

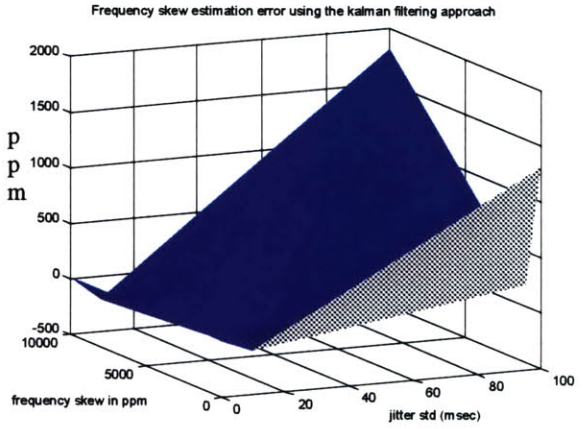


Figure 5.10 Kalman filter error (ppm) in estimates of φ as a function of network jitter and skew

Before laying out the comparison results with the original version of NTP, we should revise the modifications we are proposing to the NTP algorithm:

a) clocks frequency skew should be modeled explicitly and calculated using a filtering de-noising (de-jittering) approach (a kalman filter here, a software pll in the next section or linear programming technique at the following passage).

b) NTP packets should be source routed between NTP client and server, so as the physical communication paths from client to server and vice versa are the same.

c) NTP packet delays at the NTP server (t_3-t_2) should be constant, eliminating additional jitter noise due to NTP server operation system's multi-threaded actions.

d) The frequency of the NTP packets exchange should be more frequent than the current NTP-version 3 bound of 1 NTP packet exchange every 64 seconds (dT<<64 sec). NTP is a time correction scheme, therefore it should be able to compensate for time uncertainties of current clocking structures. For example an uncertainty of 10^{-8} of a TCXO clock imposes an error of 1 nsec every 0.1 sec. Therefore dT should be on the order of hundreds of msecs, for TCXO clocks, to compensate for the nsec error.

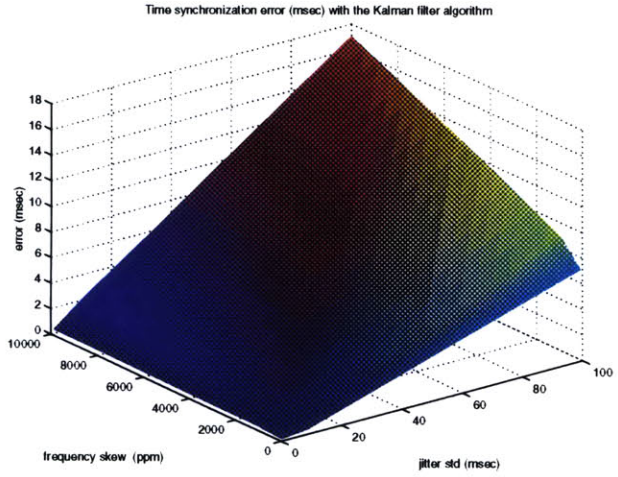


Figure 5.11 Error in time synchronization in ms With the Kalman filter algorithm

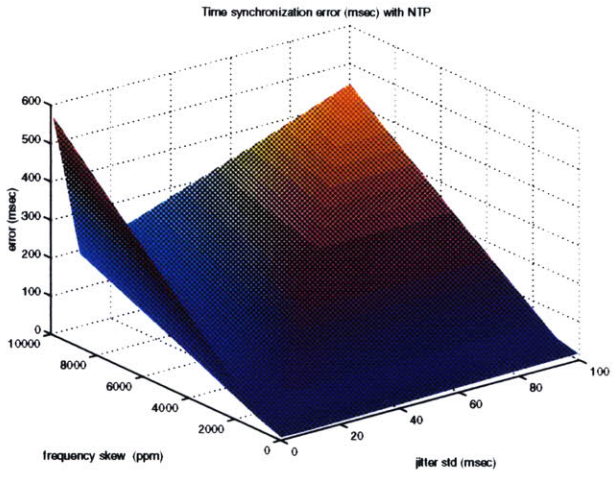


Figure 5.12 Error in time synchronization in ms with NTP-v3

The Kalman filtering approach outperforms NTP as it can be seen from the above figures and table, by two or more orders of magnitude, especially at the cases of small jitter (standard deviation of jitter on the order of 1 msec, corresponding to SNR=46 dB) or the cases of high frequency skew (10^4 ppm). For higher frequency skew (1000 ppm) the difference in time

synchronization error between the 2 algorithms is between one and orders of magnitude. In extreme cases, like in $\varphi=2=10^6$ ppm, the NTP error is extraordinary but the kalman filter error is reasonable. The figures above describe the numerical values of time synchronization in both algorithms. For asymmetric queuing delays the kalman filter performs significantly better (2 orders of magnitude) as expected. The conclusion from the kalman filtering technique with the modifications suggested above, is that the provided synchronization error is approximately two order of magnitude smaller than that of NTP and in many cases three orders of magnitude (for high SNR on the order of 40 dB, or small frequency skew between the two clocks, on the order of 1000 ppm or smaller).

The results above are not trivial. The success of the kalman filtering technique lies on the provable optimality of the de-noising (de-jittering) linear estimator of frequency skew. This is an inherent property of the kalman filtering algorithm used, as explained previously above. In the following passage we are going to show a different frequency skew estimator, which is used in MPEG-2 video decoders, which however does not provide as accurate estimates of φ as the kalman filter does, deterioriating the time synchronization quality compared with the Kalman filtering algorithm (but still providing better time synhronization for high frequency skew values than NTP).

5.2.2 Software Phased Lock Loop (PLL) technique

In this scheme, the frequency skew estimation is conducted using a software phased-lock loop. This is an idea from the MPEG community, where a synchronization mechanism is provided to lock the receiver' time base and perform consistent decoding and play-out of audio-video units. Clock skews and drifts at the encoder or decoder seriously affect the video play-out, so in the MPEG standard there are stringent requirements about maximum allowable clock skew, or drift. For example, MPEG defines the maximum frequency skew of the encoder's clock to 20 ppm =540 Hz/ 27 MHz [MPEG1994].

The de-jittering mechanism that computes the frequency skew φ out of network jitter noise is a software pll. The software pll comes from the original transfer function of an analog second order Phased Lock Loop with an active loop filter:

$$H(s)=\frac{2\zeta\omega_n s+\omega_n}{s^2+2\zeta\omega_n s+\omega_n^2} \qquad (28)$$

$\omega_n = \sqrt{K_i K_v}, \zeta = \frac{1}{2} K_p^2 \sqrt{\frac{K_v}{K_i}}$ are the natural frequency and damping factor respectively, of the frequency response of the above filter.

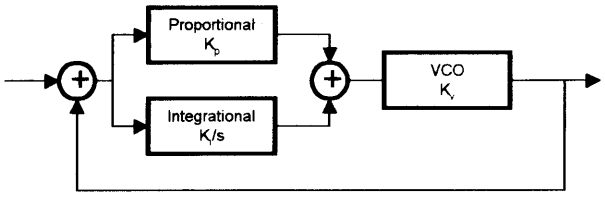


Figure 5.13 Phase Locked Loop diagram

The digital pll comes from (28) by applying a bilinear transformation $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$. The software pll algorithm for computing dt (where $dt = t_2^n - t_2^{n-1} = const$ for zero jitter) (Figure 5.2) is the following:

$$\begin{aligned}
&\hat{t}_2^n = \hat{t}_2^{n-1} + F^{n-1}(T_1^n - T_1^{n-1}), where \\
&e(n) = t_2^n - \hat{t}_2^n, \\
&P(n) = K_p e(n), \\
&I(n) = K_i e(n)(T_1^n - T_1^{n-1}) + I(n-1) \\
&F(n) = K_v(P(n) + I(n)) \qquad (29)
\end{aligned}$$

The same assumption as with the kalman filter must hold: $t_3^n - t_2^n = const$. After the pll has locked, dt is estimated as the average of 20 intervals of $\hat{t}_2$.

$$dt \approx \frac{1}{20}\sum_{j=k}^{k+19}(\hat{t}_2^j - \hat{t}_2^{j-1})$$

Frequency skew is estimated as $\phi = dT/dt$. The frequency offset θ is estimated as in the kalman filter approach, therefore the same assumptions made there (a), b), c), d)), should also hold here.

The above pll approach, due to the fact that there is no easy, direct and structured way to compute the optimal values of K_p, K_i, for specific jitter energy values (as in the kalman filter approach), provided inferior results than the kalman filter. But the most worrying observation, was the time needed to lock to a steady state, which was on the order of tens of seconds, increasing dramatically by one order of magnitude the number of NTP experiments needed to be run (compared with the kalman filter technique) and therefore compromising the initial assumption for zero drift during the experiment.

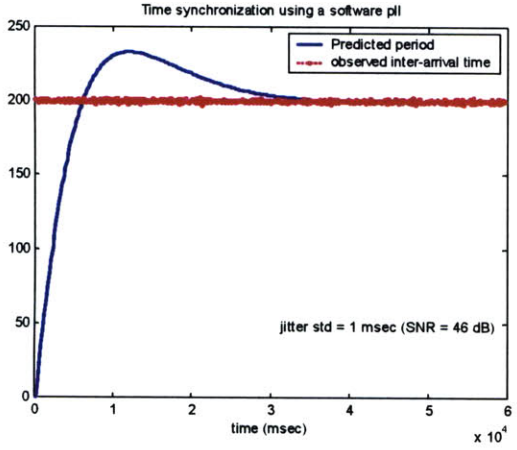


Figure 5.14 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 1 msec, $\delta\varphi$ = 100 ppm).

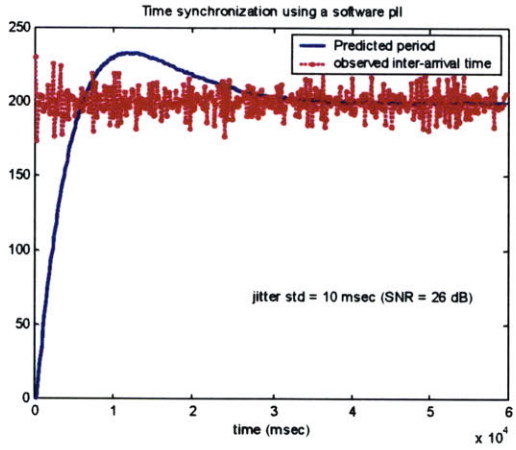


Figure 5.15 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 10 msec, $\delta\varphi$ = 100 ppm).

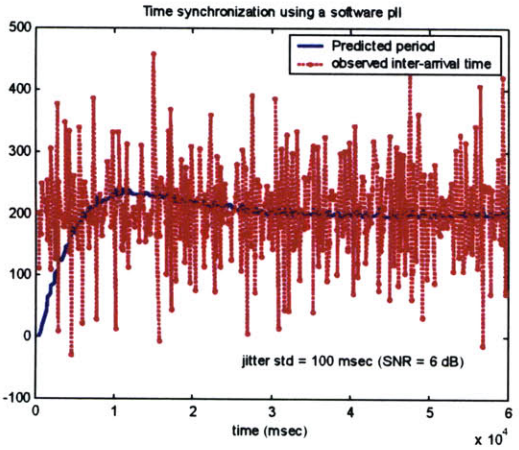


Figure 5.16 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 100 msec, $\delta\varphi$ = 100 ppm).

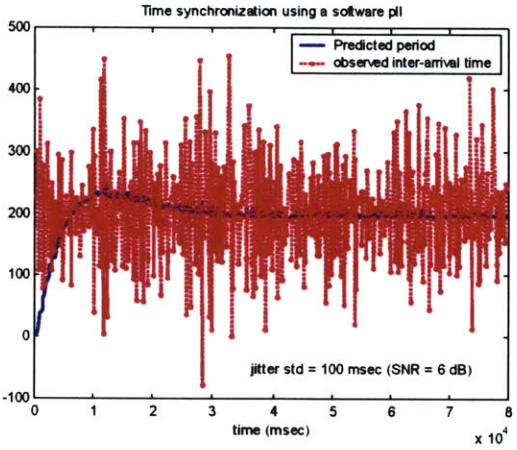


Figure 5.17 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 100 msec, $\delta\varphi$ = 10000 ppm).

Nevertheless, the pll approach performed better than NTP for cases of high frequency skew between client and NTP server clock.

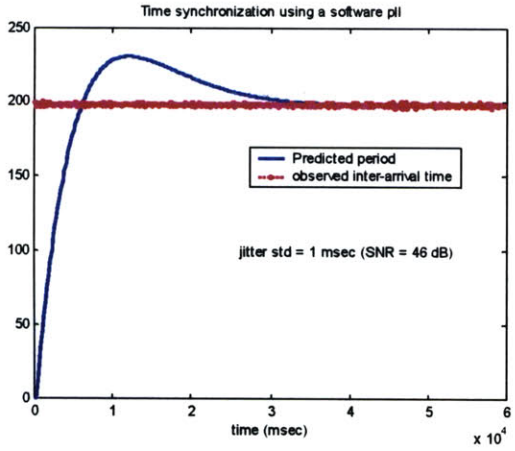


Figure 5.18 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 1 msec, δφ = 10000 ppm).

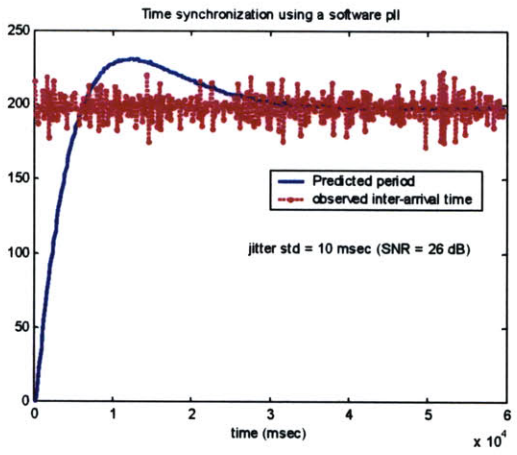


Figure 5.19 De-jittering of inter-arrival time leading to frequency skew estimation using a software pll (std = 10 msec, δφ = 10000 ppm).

ϕ	θ	Jitter std (msec)	$\tilde{\phi}$	$\tilde{\theta}$	PLL error (msec)	NTP error (msec)
1.0001	1	0.1	1.0004	10	3	7.4
1.0001	1	1	1.0004	11	4	3.98
1.0001	1	10	1.0003	6	2	3.1
1.0001	1	100	1.0052	69	13	14
1.01	1	1	1.0102	9	3.5	830
1.01	1	10	1.0104	15	4.6	190
1.01	1	100	1.0086	65	23	445
K_p=0.03 10^{-3}		K_I=0.003 10^{-6}		K_v=1		

Table 5.20 Experimental Results for the PLL filtering approach

We should clarify that the above inferior results of the digital pll used, come from the poor selection of the parameters K_p, K_i. Those parameters should be different for different network conditions. In principle a phased lock loop can be viewed as a rudimentary Kalman filter where

the state vector $\vec{x}$ is represented by the signal phase information and the measured quantity $\vec{y}$ is the signal itself. For example we can design a Kalman estimator for the signal

$$y_n = \sin(\theta_n) + v_n, \text{ where } v_n \text{ is noise and } \theta_n = \omega n + \sin(\omega_2 n), \quad x_n = \begin{bmatrix} \theta_n \\ \theta_{n-1} \end{bmatrix},$$

assuming a simple, non-markovian model $\theta_{n+1} = \theta_n + (\theta_n - \theta_{n-1}) \Rightarrow \vec{x}_{n+1} = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \vec{x}_n$

Of course the above model is non-linear, however there is a similar procedure to compute the filter parameters as in the standard linear case [Gershenfeld1999, p.194, 195].

Therefore, we can design for our purposes a pll using again a structured , kalman filtering procedure. Nevertheless, we will not present experimental results, for this case. The standard kalman filter, introduced in the previous section will be out kalman-based proposal.

5.3 Offline End to End Method for Time synchronization

The above two techniques made the assumption that during a series of NTP packet exchange, no intermediate NTP packet is lost. Practically, this is not the case, therefore the algorithm should add *"dummy"* packets appropriately before the kalman/pll calculations, or repeat the NTP experiment until an adequate number of NTP packets is acquired unobstructed by packet losses. In this passage, we are studying a different approach that doesn't have the above limitation. We saw previously that for a set of N NTP packet exchanges, the following relationships hold:

$$T_1^n - \phi t_2^n = \theta - \phi(d_1 + q_1) \le \theta - \phi d_1$$
$$T_4^n - \phi t_3^n = \theta + \phi(d_2 + q_2) \ge \theta + \phi d_2, \forall n \in [1..N]$$

Let's assume that the kth and mth NTP packets are not delayed (q_1=0) during their trip from the client to the NTP server. Then for those two packets we will have the following relationships:

$$(T_1^k - \phi t_2^k) = \theta - \phi d_1 \ge T_1^n - \phi t_2^n, \forall n \in [1, N]$$
$$(T_1^m - \phi t_2^m) = \theta - \phi d_1 \ge T_1^n - \phi t_2^n, \forall n \in [1, N]$$

By setting $a_1 = \phi, \beta_1 = \theta - \phi d_1$ the above relations become

$$\alpha_1 t_2^n + \beta_1 - T_1^n \geq 0, \forall n \in [1, N] \qquad (30)$$

$$\alpha_1 t_2^k + \beta_1 = T_1^k, \alpha_1 t_2^m + \beta_1 = T_1^m \qquad (31)$$

From (30), (31) we can see that if the set of the NTP experiment samples includes at least two "lucky" packets that incurred zero delay across the forward path, then we can define a line that passes through $(t_2^k, T_1^k), (t_2^m, T_1^m)$ on a (t_2, T_1) plane and leaves all the rest sample points at the same side of the half-plane having as a boundary the defined line. The parameters α_1, β_1 of that line will give the following information:

$$\alpha_1 = \phi \qquad (32)$$

$$\beta_1 = \theta - \phi d_1 \qquad (33)$$

$$(30),(31) \Leftrightarrow a_1, b_1 \text{ maximize the function } f(\alpha, \beta) = \sum_n (\alpha_1 t_2^n + \beta_1 - T_1^n)$$

under the constraint of $\alpha_1 t_2^n + \beta_1 - T_1^n \geq 0, \forall n \in [1..N]$

The problem is a typical case of linear programming:

find a_1, b_1 *which* maximize $\vec{F}^T \vec{x}$, *where* $A\vec{x} \leq \vec{b} \Rightarrow$

$$find\ a_1,b_1\ which\ \text{maximize}\left[\sum_n t_2^n \quad N \quad -\sum_n T_1^n\right]\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix},\ where\begin{bmatrix}-t_2^1 & -1 & 0\\ -t_2^2 & -1 & 0\\ -t_2^3 & -1 & 0\\ \vdots & \vdots & 0\\ -t_2^N & -1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix} \leq \begin{bmatrix}-T_1^1\\ -T_1^2\\ -T_1^3\\ \vdots\\ -T_1^N\end{bmatrix} \qquad (34)$$

A simplex or a computationally efficient ellipsoid method can be utilized in order to solve the above linear optimization program [Bertsimas1997]. Respectively, we can do the same for the reverse path. The resulting formulas follow:

$$\alpha_2 = \phi \qquad (35)$$

$$\beta_2 = \theta + \phi d_2 \qquad (36)$$

$$find\ a_2,b_{21}\ which\ \text{minimize}\left[-\sum_n t_4^n \quad -N \quad +\sum_n T_4^n\right]\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix},\ where\begin{bmatrix} t_3^1 & 1 & 0\\ t_3^2 & 1 & 0\\ t_3^3 & 1 & 0\\ : & : & 0\\ t_3^N & 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix}\le\begin{bmatrix} T_4^1\\ T_4^2\\ T_4^3\\ :\\ T_4^N\end{bmatrix} \qquad (37)$$

Assuming source routing, the forward and the reverse path are physically the same, so the transmission and propagation delays are the same, so $d_1 = d_2$. Therefore, from (32),(33), (35),(36) offset and frequency skew are calculated from:

$$\hat{\phi} = \frac{\alpha_1 + \alpha_2}{2} \qquad (38)$$

$$\hat{\theta} = \frac{\beta_1 + \beta_2}{2} \qquad (39)$$

With the linear optimization technique, we don't care about loses of NTP packets and there is no stringent requirement for dT or $\delta t=t_3-t_2$. However, we make the assumption that at least two "lucky" NTP packet (i.e. packet with zero one way delay) will occur at each path (forward and reverse). Therefore, the accuracy of this assumption will affect not only the calculation of θ as in the previous algorithms but also the calculation of φ. Also remember that in the previous

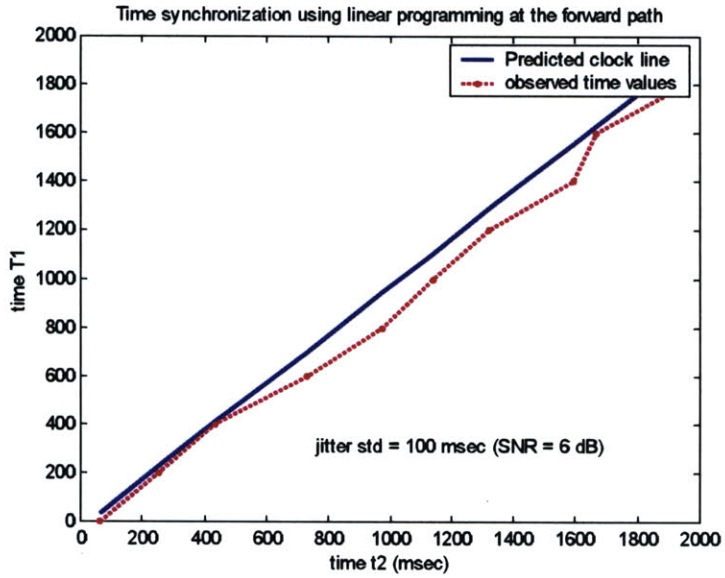


Figure 5.21 Estimation through linear programming of frequency skew at the presence of network jitter. Δφ = 10000ppm, jitter std = 100 msec. Observed values along the forward path.

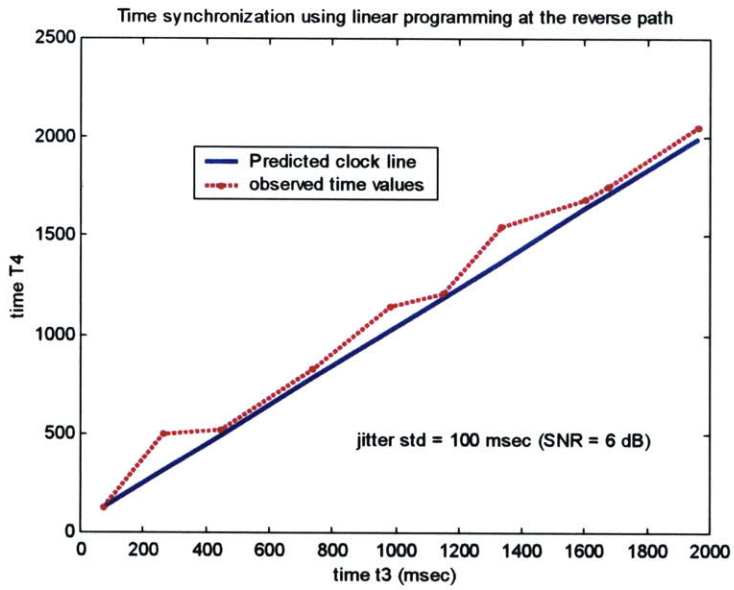


Figure 5.22 Estimation through linear programming of frequency skew at the presence of network jitter. $\Delta\varphi$ = 10000ppm, jitter std = 100 msec. Observed values along the reverse path.

N	ϕ	θ	Jitter std (msec)	$\tilde{\phi}$	$\tilde{\theta}$	Linear optimization error (msec)	NTP error (msec)
100	1.0001	1	1	1.0001	1.01	0.0235	5.8
100	1.0001	1	10	1.0001	0.58	0.4	2.25
100	1.0001	1	100	0.9849	1.0674	17	-32
100	1.01	1	1	1.0097	1.54	0.27	39
100	1.01	1	10	1.0118	0.3891	2.25	26
100	1.01	1	100	1.0121	4.12	5.37	42
100	2	1	100	1.9350	68	12.8	1603
10	1.0001	1	10	1.0001	1.247	0.27	4.3
10	1.01	1	10	1.0095	0.6566	0.78	3

Table 5.23 Experimental Results for the Linear Programming approach

algorithms we needed one "lucky" packet per path (forward or reverse), not two per path as in

here. We tested this algorithm eith the same experimental setup.

Generally, the linear optimization technique improves the NTP accuracy by one order of magnitude (and in low jitter cases by two orders of magnitude). It has comparable results with the kalman filtering approach, although slightly inferior. However, it is affected by the number N of samples being used, primary because of two reasons:

i) The linear programming solution techniques are computationally intensive.

ii) N should be large enough, so the probability of two "lucky" packets per path is adequately high.

Unfortunately, the linear optimization technique relies heavily on two "lucky" packets for the estimation of both offset and frequency skew, deteriorating the efficiency of the algorithm in high traffic loads, as we will see in the following section.

5.3 Simulation with Self-Similar cross traffic

In the previous sections, we analyzed the performance of the proposed time synchronization schemes under the existence of network jitter, defined as the variation of the difference of queuing delays across the forward and the reverse path and provided results for various jitter power values. Although we provided analytical estimates of the jitter variance in the presence of cross traffic, we didn't attempt to model the distribution of network jitter, since a task like that would be over-ambitious and the solutions for the time synchronization problem would be customized for specific jitter distributions. In fact, the solution mostly highlighted, the kalman filter one, needs only an estimation of the variance of differences of queuing delays (jitter power) without any assumption about the underlying probability distribution. The strategy of experimentation above was general enough to cover most cases without biases over specific network loads and topologies.

In this section we will try to analyze the time synchronization schemes proposed above with a less abstract but still realistic way, meaning that the strategy followed will be as close as possible to real world conditions. Recent surveys in Local Area Network traffic measurements showed a remarkable burstiness at any time-scale, proving at the same time that network traffic is *self-similar* [Leland1994], [Erramilli1996], [Taqqu1997]. Therefore, traditional traffic models based on short-range dependent processes, like the Poisson process, cannot describe realistically this chaotic behavior. Based on this observation, we will try to test our time synchronization schemes in network traffic scenarios that exhibit self-similar behavior.

The topology tested, is the following and consists of 5 routers R_i between the client and the NTP server and bi-directional cross traffic flows in every link between two routers that vary the queuing delay of the NTP flow packets. The links between the routers have bandwidth and

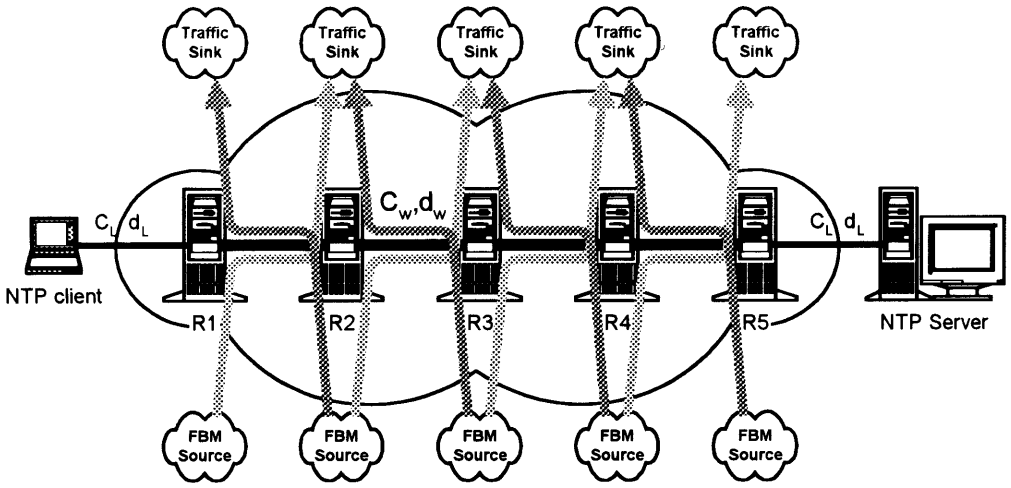


Figure 5.24 Simulation setup

propagation delay C_w, d_w respectively and the local loops of NTP client and server C_L, d_L.

The cross traffic will be based on a fractional Brownian model, inspired by the work on the analysis of storage models with self-similar input found in [Norros1994].

According to [Norros1994], a fractional Brownian motion traffic source can be described by the following formula:

$$A(t) = mt + \sqrt{am}Z(t) \qquad (bits) \qquad (40) \text{ where}$$

$A(t)$ is the total number of bits the source has produced up to time t,

m is the average rate of the source in bits/sec,

a is a modulating parameter and

$Z(t)$ is a normalized fractional Brownian motion with self-similarity (Hurst) parameter $H \in [\frac{1}{2}, 1]$

This process is characterized by the following properties:

i) $Z(t)$ has stationary increments;

ii) $Z(0) = 0, E[Z(t)] = 0,$

iii) $E\left[Z^2(t)\right] = |t|^{2H}$,

iv) $Z(t)$ has continuous paths,

v) $Z(t)$ is Gaussian

Under the above properties it can be proven that $Z(t)$ is a self-similar process, i.e.

$$Z(\beta t), t \in R, \text{ is identical in distribution to } \beta^{H} Z(t), t \in R \text{ for every } \beta > 0$$

For H=1/2, $Z(t)$ is the standard Brownian motion. The traffic model in (40) apart from its self-similar property, exhibits a superposition property: the sum of K independent fractional Brownian motion processes $Z_i(t)$ with a common Hurst parameter H is given by the following formula:

$$\sum_{i=1}^{K} A_i(t) = \sum_{i=1}^{K} (m_i t + \sqrt{m_i a} Z_i(t)) = mt + \sqrt{ma} Z(t), \text{ where } m = \sum_{i=1}^{K} m_i \text{ and } Z(t) \text{ is}$$

a fractional Brownian motion with parameter H.

The above model was coded as a separate traffic agent is ns-2, the standard, open-source simulator used by the network research community [NS-2]. NS-2 is a C++ simulator with a object oriented TCL front-end interface. Simulation scenarios are written is TCL and the simulation is run by C++ shadow objects. Fractional Brownian motion traffic agents were not available, therefore, they were coded form scratch. Of course, Pareto traffic models were available and it has been shown that the aggregation of many Pareto sources can show long range dependence [Taqqu1997]. Nevertheless, the above direct approach was adopted. Moreover, NTP server/client agents were not present either, so they were coded from scratch.

The following section presents the efficiency of the time synchronization algorithms proposed above in a chaotic environment, as today's Internet.

5.3.1 Local area Network scenario

For propagation delays on the order of 5 msec and C_w=C_L=10Mbps the sum of propagation and transmission delay for the topology given above, assuming NTP packets of 128 bytes is the following:

$$d_1 = d_2 = 6*5m\sec + 6*\frac{128*8}{10*1000}m\sec = 30.6144m\sec$$

Each fractional Brownian motion cross traffic has average rate $m_i = 1Mbps$ and the sampling interval of the modified NTP protocol is 200msec therefore, the utilization of each link between two routers is $\frac{1Mbps + \frac{128*8}{200*10^{-3}}10^{-6}Mbps}{10Mbps} = 10.05\%$, therefore the links are underutilized. Our goal is to show that our proposed algorithms perform one or two (or more) orders of magnitude better than NTP. The results follow:

	ϕ	θ	$\hat{\phi}$	$\hat{\theta}$	***Rms error (msec)***
Kalman	1.0001	1	1.0001	1.4791	0.075
	1.001	1	1.001	1.4795	0.7055
	1.01	1	1.01	1.483	0.0755
PLL	1.0001	1	1.0003	-34	5.5
	1.001	1	1.0012	-34.06	5.59
	1.01	1	1.0102	-31.9	5.63
Linear Programming	1.0001	1	1.0001	1.0417	0.0062
	1.001	1	1.001	1.023	0.0045
	1.01	1	1.01	1.0193	0.0026
NTP-v3	1.0001	1			29
	1.001	1			276
	1.01	1			2700

Table 5.25 Simulation results for lightly utilized LAN

The above results show a decreased time synchronization error of the proposed algorithms compared to NTP by at least two orders of magnitude. This is because NTP neglects explicit modeling of clock frequency skew and because the traffic load is low increasing the probability of the existence of "lucky" packets with zero queuing delay.

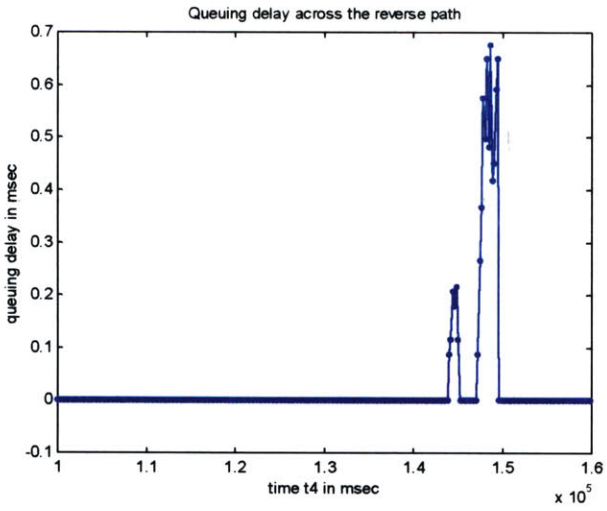


Figure 5.26 Queuing delays across the reverse path for 10% link utilization

Moreover, the local area network is a high-speed network and the queuing delays are smaller that those in WANs, therefore the variation of queuing delays (jitter) is smaller, allowing improved performance of the time synchronization schemes, as explained in previous sections. With a more careful look we should observe that the linear programming technique provides smaller error than the kalman filter. This is because in the kalman filter implementation we used only a FIR LP filter of order 20, at the exit of the kalman filter. In the software pll approach, we used the same integration and proportion parameters as well as the same initialization coefficients as in the analysis discussed previously. Unfortunately, the transient times of the filter until the steady state, was on the order of 140 seconds, making the use of plls unpractical.

We will repeat the above experiment, but this time with higher utilization (90%) setting each FBM source to 9 Mbps.

	ϕ	θ	$\hat{\phi}$	$\hat{\theta}$	Rms error (msec)
Kalman	1.0001	1	1.0001	-0.6	0.21
	1.001	1	1.001	-0.6397	0.2179
	1.01	1	1.01	-0.6544	0.2171
PLL	1.0001	1	1.0004	-42	7.12
	1.001	1	1.0013	-42.3	7.08
	1.01	1	1.0103	-40	6.69

Linear Programming					
	1.0001	1	1.0001	2.0525	0.14
N=50	1.001	1	1.0009	9.07	0.25
N=50	1.01	1	1.0099	9.6668	0.3221
N=300	1.01	1	1.01	0.8831	0.3371
NTP-v3	1.0001	1			29
	1.001	1			276
	1.01	1			2700

Table 5.27 Simulation Results for highly utilized LAN (utilization 90%)

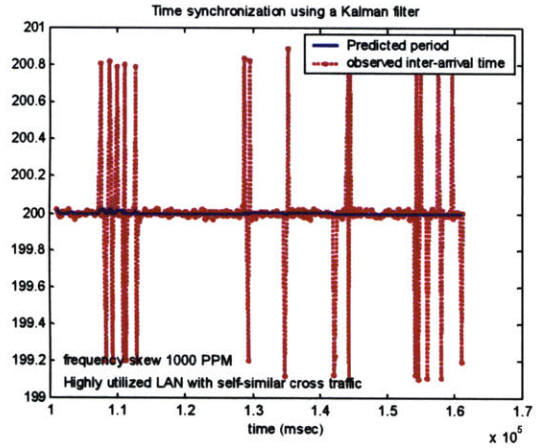


Figure 5.28 Kalman filter approach in highly utilized LAN (90%) with self- similar cross traffic.

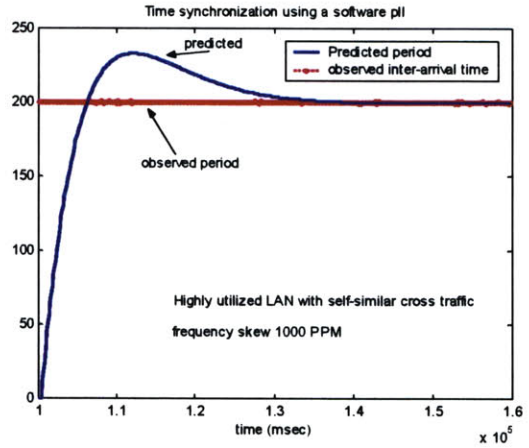


Figure 5.29 PLL filter approach in highly utilized LAN (90%) with self- similar cross traffic.

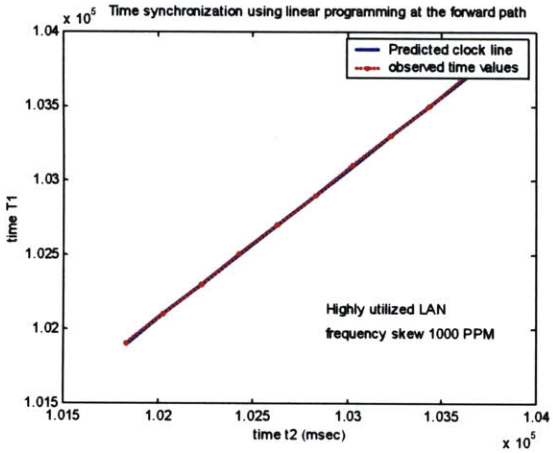


Figure 5.30 Linear programming filter approach in highly utilized LAN (90%) with self- similar cross traffic (forward)

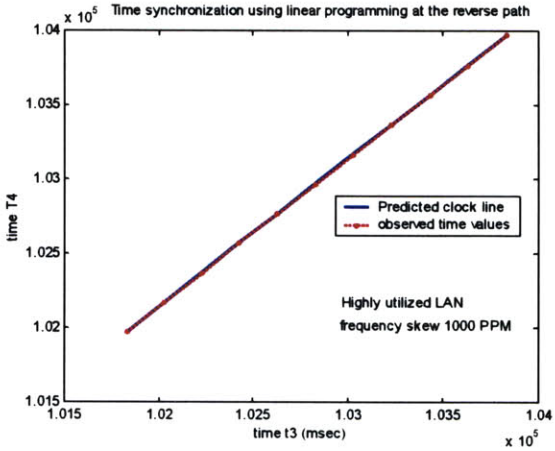


Figure 5.31 Linear programming filter approach in highly utilized LAN (90%) with self- similar cross traffic (reverse)

The results are very satisfying and encouraging. The kalman filtering approach, as well as the linear programming algorithm achieves better time synchronization from NTP, by at least two orders of magnitude, even though the network is highly utilized. Of course, the results are inferior those in the light traffic case, however they are much better than NTP. In this experiment we used N=50 so as to test the quality degradation of the linear programming technique. The pll approach provides good accuracy only after a significant amount of transient time on the order of hundreds of seconds, which is a significant shortcoming.

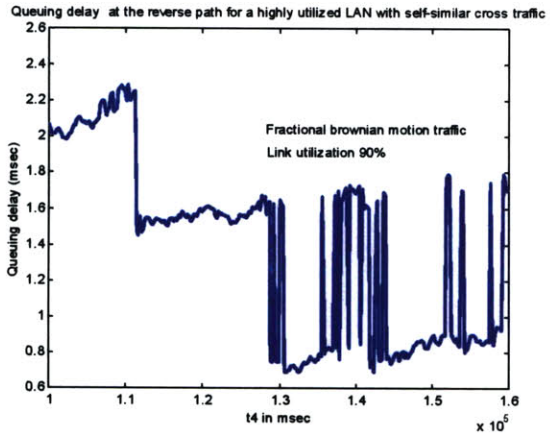


Figure 5.32 (non-zero) Queuing Delay for 90% FBM traffic in LAN

5.3.2 Wide Area Network scenario

We repeated the experiments for Wide Area Networks (WAN) by changing the capacity of inter-router links from C=10 Mbps to C=1.5 Mbps and propagation delay from d=5 msec to d=10 msec. The local loop capacity and propagation delay remained 10 Mbps and 5 msec respectively. The results again showed significant improvement in rms time synchronization error by at least one order of magnitude.

5.4 A note on NTP-version 4

NTP-version 3 is used in this work for baseline comparison. NTP-v3 has indeed implemented a software pll, however not for de-jittering usage but for clock–discipline purposes (for example the clock should preserve monotonicity after the corrections from the NTP experiment). D. Mills, has proposed a revised version of NTP, NTP-v4 which incorporates PLL and Frequency Lock Loops (FLL) for frequency and offset correction. However no structured/optimal algorithm for the

computation of the filter parameters was provided. It remains to be seen if those filters exhibit the same problems as the PLL studied here.

We are ready now to proceed to the conclusions section.

6 Conclusions and Applications

6.1 Conclusions

We identified the main physical causes for differences in time references over a distributed system and underlined the need for a time-synchronization scheme to address explicitly the frequency skew φ and offset θ between the time source (reference) server and the NTP *"to-be synchronized"* client. We analytically explained why and how network packet delay variance (jitter) affects time synchronization and suggested that the new time synchronization scheme should treat network jitter as 'noise' to the synchronization mechanism. Therefore, we tried to devise algorithms that extract the useful timing information out of network noise (jitter). The de-jittering approach that we adopted is based on a kalman filtering algorithm which is provably the optimal linear estimator we can devise. Apart from its optimality (for the linear case), this algorithm provided a structured way to compute the parameters of the Kalman filter, using the NTP-version 3 measurements. The algorithm was tested under various network conditions and clock offsets and *provided rms synchronization error two orders of magnitude smaller than NTP-v3*. The non-zero error of the algorithm was due to the procedure followed for the offset θ estimation which was based on the existence of one "lucky" packet with minimum (close to zero) queuing delay. We don't claim that we have created a time synchronization scheme over packet

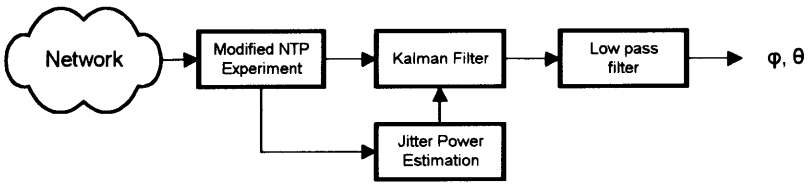


Figure 6.1 The Kalman Filter Approach

switched networks with a GPS (nanosecond) accuracy. We do make the claim that we have implemented a suite of algorithms that can be built on top of NTP and improve its accuracy by two orders of magnitude. Particularly, the Kalman filter algorithm can be built with a direct, easy and nearly optimal way using jitter estimation from the network environment itself. This way for NTP improvement is the major contribution of this thesis.

Moreover, for position estimation in sensor networks, time differences are important as we will show in the applications passage, therefore accurate offset estimation is useless (at least for this cause). Frequency skew φ between the clocks of the base station and the client affects position

estimation applications. If for example the NTP server has a time reference t, and the NTP client a time reference $T = \phi t + \theta$, but the time synchronization algorithm provides $\phi, \bar{\theta}$ then the client would correct its clock by $(T - \bar{\theta})/\phi$. In a Loran-C like algorithm, time differences are utilized for position estimation so $\delta t = \delta T / \phi$, therefore the true offset value is not needed. The kalman filter algorithm, as well as the Linear Programming can compute the value of φ with high accuracy as we have showed previously, in various network conditions.

Nevertheless, very accurate frequency skew φ estimates are critical. The Kalman filtering approach manages succesfully to estimate φ with residual error close to zero ($\approx 10^{-10}$) for SNRs (defined in chapter 5) below 26 dB. This bound can be further improved if special averaging before the kalman filter is utilized. We emphasize the two orders of magnitude improvement in rms error over NTP. Talking about absolute values, the error could be decreased on the order of μicrosecond for low SNRs and low frequency skew values (below 100 ppm). However, we didn't incorporate resolution error in our experimental setup, therefore we expect this number to be higher.

More accurate estimation of θ could be done using *entrainment*, forcing a common heartbeat in the network and using that heartbeat in order to synchronize. This suggestion is not a science fiction scenario, but an observed fact in Routing Internet Protocol (RIP) agents which under specific conditions, managed to synchronize and send their routing messages at the same time point, even though they originally transmitted with a random offset difference. Entrainment occurs since each routing agent postpones the departure of its new routing message update, until it processes the message sent by another RIP agent. This phenomen, and the special conditions required, have been thoroughly analyzed in [Floyd1994].

We contrasted our kalman filter approach with a software phased lock loop di-jittering mechanism as well as with a linear programming technique. The pll approach manages to reduce the rms error however not as successfully as the kalman filter technique. This is because the pll filter doesn't have an easy, quick and structured way for the computation of its optimal parameters for various and changing network conditions. Its transient time until it locks to the correct value can be too high (at the experiments run here, it was on the order of hundreds of seconds while the respective kalman filter transient time was on the order of a few seconds) therefore, it could potentially violate the assumption for zero frequency drift $(T_r'(t) = 0)$.

The linear programming technique de-jittered successfully, however it is computationally very intensive and its accuracy depends on the number of NTP message exchanges, therefore it can potentially violate the zero drift assumption if the NTP experiment duration is high. It is based on at least two "lucky" packets (zero queuing delay) on every path (forward and reverse) affecting not only the offset but also the frequency skew estimation, therefore in principal it is sub-optimal compared with the Kalman filter one. All the three algorithms were tested for various queuing delay variances (jitter) and various link utilizations, assuming traffic with zero dependence (Gausian traffic) or long range dependence (self-similar traffic).

According to the above results, the NTP algorithm, augmented with the kalman filter de-jittering mechanism, can provide the End-to-End time synchronization mechanism we desired, improving NTP rms error by two orders of magnitude. The hardware constraints of the NTP server are basically constant and fixed delay of each NTP packet, while the hardware constraints of the NTP client are conditions that impose zero drift (second derivative of timing function) during the NTP experiment, which is a reasonable assumption if the NTP experiment is short. We underline the fact that our algorithm was tested with frequency skew values ranging from 100 ppm to 10000 ppm and in one extreme case for $\varphi=10^6$ ppm, therefore we don't make any assumption about the frequency skew of the client.

Finally, we should remind another modification in the NTP-v3 protocol, which is the source routing of NTP packets across the forward and reverse path so as to zero the asymmetry of the propagation and transmission delays across the forward (from client to NTP server) and the reverse (form server to client) path, which could also deteriorate the time synchronization performance.

The modifications of the NTP algorithm in parallel with the notion of de-noising using an optimal Kalman filter with parameters calculated form the network environment, are the major contribution of this thesis. This is the answer to the hard question in chapter one: how can we have an accurate End-to-End time synchronization scheme, better than NTP-v3. We provided algorithms that reduced the rms error two orders of magnitude. End-to-End solutions can be incorporated in today's Internet while router modification-based techniques, like those in chapter 4 can be implemented only in custom networks and are (at least theoretically) trivial.

6.2 Applications

We are envisioning Myriad Networks as a new class of packet switched networks where time keeping is a critical service layer, and on top of that all the other important layers like routing (IP), congestion control (TCP) etc. of the seven layer architecture follow. We have connected the time-

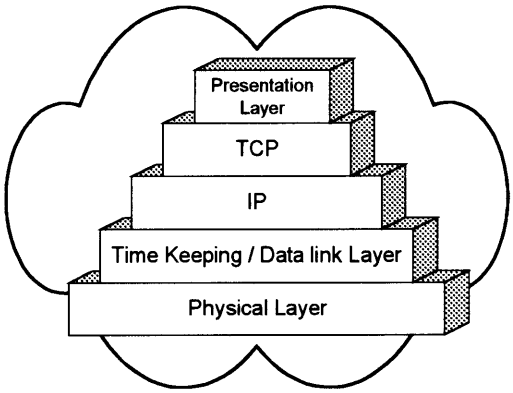


Figure 6.2 Time keeping Layer in Myriad Networks before anything else...

keeping facility with the data link layer, since accurate timing is crucial in all digital communication links (otherwise you would be able to use you Mbps and soon Gbps Ethernet card), however during the layered network design over the last 20 years we tend to neglect this crucial information. The timing layer is responsible for correct time keeping over the network and this is achieved with the adoption of special network architectures (like routers that minimize jitter, lke those presented in chapter 4) or by running the augmented version of NTP with the de-jittering kalman filter, as proposed above.

Now, assuming that we have a common clock in a distributing system (as well as a de-jittering mechanism like the kalman filter) many applications can be envisioned:

6.2.1 Beam-forming from a distributed, ad-hoc set of sensor nodes

Spatial filtering , known as *beamforming* is used to distinguish between the spatial properties of signal and noise and finds many practical uses in radar, sonar, communications, geophysical exploration and biomedical signal processing. A set of independent sensor nodes are placed at different points in space to "listen" to the received, useful signal. In order for the beam-former to work, a common time reference should be available. Moreover, in a scenario where the nodes are

scattered randomly (ad-hoc environment), the relative position of each node to the others should be estimated and this information be used by the beam-forming algorithm. If time keeping and position estimation is not provided by an external source (like a GPS receiver) at each node, could the algorithms devised in this work help solve the position estimation problem?

Let's assume that there is one source of correct time which disseminates this information to all the other nodes, using an NTP like protocol augmented by the kalman filtering modifications proposed above. Let's also assume that S1, S2, S3 will be the reference points according to which

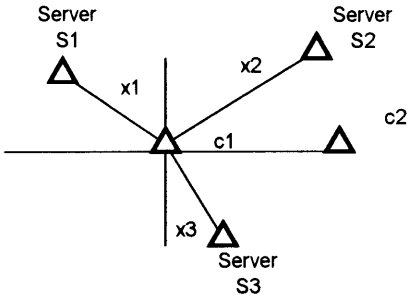


Figure 6.3 Loran-C Position Estimation method

all the other nodes will compute their relative position. Those nodes emit a known waveform (audio or rf) at specific and know intervals of time. To simplify the analysis, those nodes are synchronized to the 'true" time and emit their waveforms at constant intervals. This sequence is S1 at t1, S2 at t2, S3 at t3 with (t2-t1=T), (t3-t2=T). All the other nodes are synchronized to the time reference servers using the kalman-NTP approach, over the network between all the nodes.

In a Loran-C like approach, assuming that the clock of client c1 is $T_{c1}(t) = \phi t + \theta$ and the clock of S1, S2, S3 is $T(t) = t$ (true time) we can compute the position of c1 relative to all the others with the following way: S1 sends its waveform at t1, which is received at c1 after $\frac{x1}{c}$ and S2 sends its waveform at t2, which is received at c1 after $\frac{x2}{c}$, therefore the clock of c1 will show

$$T_1 = \phi(t1 + \frac{x1}{c}) + \theta, T_2 = \phi(t2 + \frac{x2}{c}) + \theta$$

at the time of the arrival of the two waveforms for S1, S2 respectively. Therefore,

$$T_1 - T_2 = \phi(\frac{x1}{c} - \frac{x2}{c} - T) \Rightarrow x1 - x2 = c\frac{T_1 - T_2}{\phi} + cT \qquad (1)$$

if we do the same for the pair S2, S3 and S3, S1 then:

$$x2 - x3 = c\frac{T_2 - T_3}{\phi} + cT \qquad (2), \quad x3 - x1 = c\frac{T_3 - T_1}{\phi} - 2cT \qquad (3)$$

(1), (2), (3) define a set of hyperbolas, therefore it is possible to find the cross section and compute x1, x2, x3 provided that the time keeping algorithm will provide us with ϕ. The kalman- NTP can provide very accurate estimates of ϕ (residual error on the order of 10^{-10}=10^{-4} ppm), for network SNRs above 26 dB as we have already showed. Therefore, it is possible to compute the position of a network node, using the timing information from the network of nodes itself, without the need of an expensive GPS receiver. The question remained to be answered is how often the client should run the NTP-kalman experiment in order to correct its clock. For a common quartz resonator of 10^{-6}= 1 ppm time uncertainty $\frac{\delta t}{t}$, this means an error of 864 msec per day. Since we are interested in frequency skew estimation and we have made implicitly the assumption that clock drift of c1 is zero ($T''_{c1}(t) = (\phi t + \theta)'' = 0$), the NTP-kalman filter intervals of experimentation should be small but frequent enough so as the zero drift assumption is not violated.

The above example is a very concrete example of how better clock modeling can be useful in position estimation. If we had modeled $T_{c1}(t) = t + \theta$, as NTP does, then we would need to follow different strategy than the above in order to compute position. Specifically, we should incorporate very stable clocks, in order to minimize θ. This is what GPS does, using cesium clocks with stability on the order of 10^{-14}, which means an error of 0.864 nanosecond every 24 hours. This high stability is needed, because the satelites can be time-corrected only once every 24 hours, when they become visible from GPS center in Edwards US. Air Force Base.

6.2.2 Improvements in TCP

TCP uses round trip time estimates, exploiting the ACK mechanism, so as to compute the size of the (congestion) window of packets which can be transmitted so as to avoid or relief congestion and at the same time, fully utilize the network. However, this estimate couples the available bandwidth of forward and reverse path which can be asymmetric as it has been shown in [Paxson1997]. Therefore, having separate estimates of queuing delays across the forward and reverse path could imtpove TCP performance. One way delay measurements are needed which means, that the end nodes should be time-synchronized. The time-keeping layer in Myriad Networks can provide this ability. TCP-Santa Cruz [Parsa1999] uses one way delay estimates to compute the queuing delays across the forward and the reverse path, therefore it could benefit (and practically be improved) from our time-synchronization scheme.

Finally our de-jittering approach can simplify real-time video streaming, since current approaches use a buffer to remove network jitter. Further work is needed to discover the appropriate mechanics.

6.2.3 ...and a demo

We finish our work with a networking demo presented at the IPID Media Lab SIG, on May' 01, which tried to make apparent the effects of network jitter.

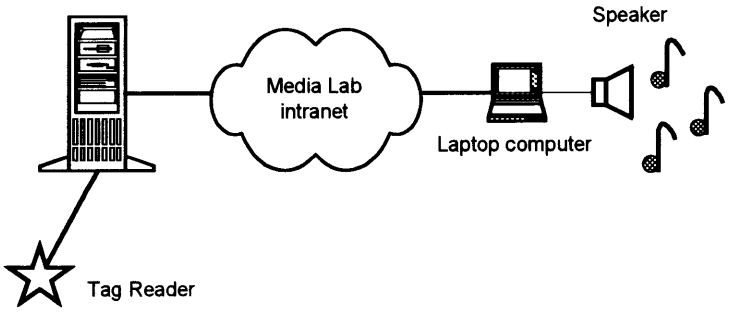


Figure 6.4 Jitter-controlled ipid tone synthesizer (!)

A tag reader was connected through a serial to ethernet interface to a remote server. The tag reader was presented a set of different tags corresponding to different resonant frequencies. The server translated the set of frequencies to id numbers, using a local database and sent those ids

over UDP connections to a remote client, through Media Lab's Intranet. The client was synthesizing tones with an increasing sequence, from low ids to high ids, depending on which ids it had received. This was done in a loop. Therefore, the user at the tag reader could synthesize simple tone patterns, which would be reproduced at a remote host. If no jitter was present, a specific tone pattern was occurring, while in the presence of jitter the tone pattern was no longer canonical. A software knob at the client side, magnified this jitter effect.

This is hopefully, the last sentence in my thesis... Thank you for reading.

References

[Abelson2000] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, R. Weiss. *Amorphous Computing*. Communications of the ACM. Volume 43, issue 5. May 2000.

[Andreotti1995] G. F. Andreotti, G. Michieletto, L. Mori, A. Profumo, *Clock Recovery and Reconstruction of PAL Pictures for MPEG Coded Streams Transported Over ATM Networks.* IEEE Transactions on Circuits and Systems for Video Technology, volume 5. December 1995.

[Bertsimas1997] D. Bertsimas, J.N. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific 1997.

[Carlstrom1999] J. Carlstrom, E. Nordstrom. *Reinforcement Learning for control of self-similar call traffic in broadband networks.* Proceedings of the International Teletraffic Congress – ITC-i6. Edinburgh UK 1999.

[Erramilli1996] A. Erramilli, O. Narayan, W. Willinger. *Experimental Queuing Analysis with Long-range Dependence Packet Traffic.* IEEE/ACM Transactions on Networking, volume 4. April 1996.

[Evans2000] D. Evans. *Programming the Swarm.* University of Virginia, Department of Computer Science. July 2000.

[Ferrari1990] D. Ferrari, D.C. Verma. *A Scheme for Real-Time Channel Establishment in Wide-Area Networks.* IEEE Journal on Selected Areas of Communications, volume 8, no 3. April 1990.

[Floyd1994] S. Floyd, V Jacobson. *The Synchronization of Periodic Routing Messages.* IEEE/ACM Transactions on Networking. April 1994.

[Gershenfeld1999] N. Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press 1999.

[Gershenfeld2000] N. Gershenfeld. *The Physics of Information Technology.* Cambridge University Press 2000.

[Kahn1999] J.M. Kahn, R. H. Katz, K. Pister. *Next Century Challenges: Mobile Networking for Smart Dust.* ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 99). August 1999.

[Leland1994] W.E. Leland, M.S. Taqqu, D.V. Wilson. *On the Self-Similar Nature of Ethernet Traffic.* IEEE/ACM Transactions on Networking, volume 2. February 1994.

[Landry1997] R. Landry, I. Stavrakakis. *Study of Delay Jitter With and Without Peak Rate Enforcement.* IEEE/ACM Transaction on Networking, volume 5. August 1997.

[Major1998] F. G. Major. *The Quantum Beat: The Physical Principles of Atomic Clocks.* Springer New York 1998.

[Marzullo1985] K. Marzullo, S. Owicki. *Maintaining the time in a distributed system.* ACM Operating Systems Review 19. July 1985.

[Matragi1997] W. Matragi, K. Sohraby, C. Bisdikian. *Jitter Calculus in ATM Networks: Multiple Nodes.* IEEE/ACM Transactions on Networking, Volume 5, no. 1. February 1997.

[Mills1992] D. L. Mills. *Network Time Protocol (Version 3) Specification, Implementation and Analysis.* RFC 1305. University of Delaware. March 1992.

[Mills1997] D. L. Mills, A. Thyagarajan, B.C. Huffman. *Internet timekeeping around the globe.* In Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting. December 1997.

[Mills1998] D. L. Mills. *Adaptive Hybrid Clock Discipline Algorithm for the Network Time Protocol.* IEEE/ACM Transactions on Networking volume 6. October 1998.

[Minar1999] N.Minar. *A survey of the NTP Network.* MIT Media Lab. http://nelson.www.media.mit.edu/people/nelson/research/ntp-survey99/html/. December 1999.

[Moon1999] S.B. Moon, P. Skelly, and D. Towsley. *Estimation and removal of clock skew from network delay measurements.* Proc. of 18th Annual Joint Conf. of the IEEE Computer and Communications Societies. (INFOCOM). 1999.

[MPEG-2] Information Technology--Generic Coding of Moving Pictures and Associated Audio Information: Systems. ISO/IEC 13818-1, 13818-2, 13818-3. 1996.

[Norros1994] I. Norros. *A Storage Model with Self-Similar Input.* Queueing Systems, volume 16. 1994.

[NS-2] http://www.isi.edu/nsnam/ns/

[Parsa1999] C. Parsa, J.J. Garcia-Luna-Aceves. *Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media.* Proceedings of IEEE, ICNP '99. Toronto, October 1999.

[Paxson1997] V. Paxson. *Measurement and Analysis of End-to-End Internet Dynamics.* Ph.D thesis. University of California at Berkeley. April 1997.

[Poor1999] R. Poor. *Synchronization in Decentralized Networks.* MIT Media Lab. 1999 http://www.media.mit.edu/~r/academics/PhD/sync_diffuse/index.html

[Simon1969] H. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge Massachusetts 1969.

[Singh1994] R.P. Singh, Sang-Hoon Lee, Chong-Kwoon Kim. *Jitter and Clock Recovery for Periodic Traffic in Broadband Packet Networks.* IEEE Transactions on Communications, volume 42, no. 5. May 1994.

[Taqqu1997] M. S. Taqqu, W. Willinger, R. Sherman. *Proof of a Fundamental Result in Self-Similar Traffic Modeling.* ACM Computer Communications Review. April 1997.

[Tennenhouse2000] D. Tennenhouse. *Embedding the Internet: Proactive Co-mputing.* Communications of the ACM. Volume 43, issue 5. May 2000.

[Troxel1994] G. D. Troxel. Time Surveying: Clock Synchronization over Packet Networks. Ph.D. thesis, MIT EECS, May 1994.

[Verma1991] D.C. Verma, H. Zhang, D. Ferrari. *Delay Jitter Control for Real-Time Communication in a packet Switching Network.* Proceedings of Tricomm'91.

[Weiser1991] M. Weiser. *The computer for the 21st century.* Sci. Am. 265, 3. September 1991.