Nanosecond-scale Event Synchronization over Local-area Networks

Steven E. Butner
Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106

Scott Vahey
Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106

Abstract

The research discussed in this paper builds on the success of the Network Time Protocol (NTP) which has been synchronizing clocks on hosts all across the Internet for many years. With a different end goal (NTP seeks to setup and maintain agreement of UTC time in network-connected systems) we augment NTP methods with hardware to improve the accuracy of synchronization that is possible within an ethernet subnet by more than three orders of magnitude.

Our approach involves a study of the sources of non-determinism in NTP exchanges and their removal by special hardware aids. The added hardware has a real-time synchronized clock output and a programmable 'event synch' output that can be used to synchronize attached real-time equipment.

The hardware aids allow us to achieve much better synchronization precision (100 nsec) while also facilitating the convergence to synchronization in a vastly shorter time — about one minute rather than hours or days as required by NTP.

1 Introduction

Much work has already been done on the subject of Internet time synchronization. The Network Time Protocol (NTP) has evolved and matured and is routinely used today on many Internet hosts. The state-of-the-art accuracy for such network-based time synchronization is tens of milliseconds across arbitrary connections with perhaps 0.5-10 milliseconds of accuracy within a local area (LAN) subnet. The approach uses a relatively simple protocol that can estimate round-trip times as well as local clock drift. After achieving lock, the NTP approach can maintain loose synchronization at the above-stated level for long periods of time — even in the presence of network drop-outs, system reboots, and the like. There are three issues with today's NTP approach that we have addressed in this research:

1. Achieving any sort of meaningful frequency lock takes a minimum of several hours and often requires several days(!). This is due to the variability and non-determinism in round-trip delays within a network. Statistics are taken on packet exchanges and these are used to tune a software-implemented phase-locked loop (PLL). NTP packet exchanges are purposely infrequent so as to minimize overhead. This has the consequence that tuning times can be quite long.
2. Even when everything is working perfectly the degree of synchronization achievable is not very good with respect to external application events. This is because the *primary* thrust of NTP is to synchronize a system's notion of Coordinated Universal Time (known as UTC time) rather than to attempt to tightly synchronize real-time events on multiple networked hosts.
3. The resulting synchronization is not directly usable for triggering local (hardware) events since the approach involves encapsulating a system's local oscillator hardware with a software layer that can adjust, upon demand, for frequency drift and phase offset relative to a more authoritative time source. Thereby, one can compute a good estimate of UTC time at any moment by calling the system's `gettime()` routine. It cannot, however, easily schedule local hardware

events so as to ensure that they are in-phase with NTP-calibrated network time. Ideally, one would like a *local* hardware clock that is locked in frequency and phase to other clocks within the network.

This research has sought to improve on the accuracy of achievable synchronization by removing as much as possible of the non-deterministic delay within message exchanges that are part of the network time protocol. In a nutshell, the approach is to identify the sources of non-determinism and/or unpredictable delays in a network carrying NTP messages and utilize additional hardware (resulting in minor changes in the design of a network controller) to remove or greatly reduce these delays. The resulting deterministic delays allow NTP to vastly improve[1] its host-to-host synchronization accuracy as well as its capture-and-lock time with a goal of a few seconds or minutes in the worst case.

Comment on GPS

A reviewer of our original proposal for this project commented that the "standard solution" to time synchronization was to use GPS and that the proposed research was therefore not well motivated. We do not dispute that GPS can indeed be used to achieve host-to-host synchronization. The results reported here should not be viewed as a competition, however. There are applications where GPS may in fact be the best choice but there are also applications where the infrastructure for GPS is simply too costly or where the location of the system precludes GPS use. We have worked to create a *very low-cost* yet highly accurate time synchronization method that can work within existing local area network infrastructure. Note that our modifications to an ethernet interface are minimal, making their integration into a media-access control (MAC) chip feasible. Were this to occur, the incremental cost for using ethernet-based NTP to synchronize real-time distributed systems would be at most a few dollars. The incremental per-node cost of GPS technology, however, is significantly higher.

Having multiple methods of achieving and maintaining time synch is highly desireable. In fact, incorporating a GPS receiver in our "local master" time server is one way that we could perhaps avoid the lengthy initial NTP tuning cycle. We have observed that the introduction of our proposed modifications shorten the tuning cycle appreciably, since individual packet exchanges have essentially deterministic delays that can be acted upon immediately rather than waiting for the convergence of statistical methods. In short, this work does not ignore GPS nor does it preclude its use when appropriate.

[1]Our goal was an improvement in synchronization accuracy over that of NTP by 3-4 orders of magnitude — from milliseconds to approximately 100 nanoseconds.

2 Network Time Protocol

There are several different ways of attaching network-aware devices to form a local-area network. Protocols in use today were developed during an era where the carrier-sense/multiple access (CSMA) Ethernet operated over a shared medium (Figure 1) connecting distinct hosts via a common passive coaxial cable. Because of reliability and cost issues in connecting new systems and in maintaining connections for existing systems to a single passive coaxial medium (10base-2 or 10base-5 style) most users have moved to the use of hubs and switches so that each host is connected to the network via a private attachment between the host and a hub or switch (Figure 2). Hubs and switches have been carefully engineered so that existing protocols still work even though the topology of the network has evolved and diverged from the original shared-medium style.

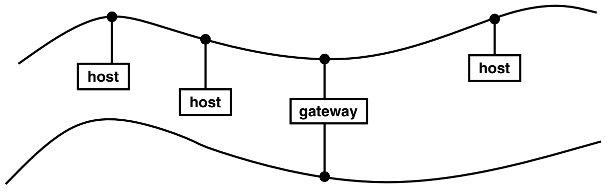


Figure 1: CSMA Ethernet subnet implemented on a passive coaxial cable medium

CSMA-type interfaces sense the presence of carrier to test if the network is in-use. The simple rule is: listen first and only when clear, begin transmitting. On busy nets, collisions happen (multiple hosts make the decision that it is OK to transmit at very nearly the same instant) with the result that their transmissions collide and garble each other. The solution developed under the original CSMA ethernet shared-medium paradigm is random backoff and retry. This strategy has served us well and it is still a crucial part of the ethernet specification today. One side effect of random backoff is the presence of unknown and non-deterministic delays. We simply cannot know how long it will take from the time a network packet is setup for transmission until it actually goes out onto the network medium. Adding to the non-determinism are other significant queuing and scheduling delays in the operating system. Ever since networks became available there have been attempts to use the host-to-host connectivity to establish and maintain a common notion of time. The Network Time Protocol (NTP) is used on many hosts today to achieve and maintain clock synchronization. In addition to NTP there are several other protocols designed to distribute time in LANs: the DAYTIME protocol [5], TIME protocol [6], ICMP timestamp messages [2] and ICMP timestamp options [7]. The NTP approach is based on sampling UTC time stamps from various hosts and using statistical methods to model and make adjustments to an individual host's notion of time based on a local oscillator.

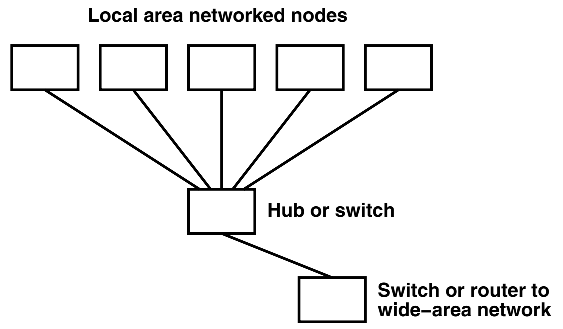

Local area networked nodes
Hub or switch
Switch or router to wide–area network

Figure 2: LAN nodes connected via a hub or switch

Such a setup can achieve fairly good agreement between the time clocks of the participating hosts — accuracy on the order of tens of milliseconds is routine — but network traffic and operating system queuing delays are non-constant. It can take as long as several days of sampling and exchanging NTP messages to achieve and maintain frequency lock. It is also important to know that the computer's local oscillator is not modified but rather a software model encapsulates the hardware time-keeping so that frequency and offset adjustments can be made on demand. This has the consequence that at any moment the present UTC time can be determined by a simple computation but it remains difficult to arrange for a locally-scheduled (hardware) event to happen with a given phase with respect to the network-synchronized time.

When the transmitter interface within a given host sends a message on the ethernet, its locally-generated bit rate clock is used to set the timing of the message. Receivers must synchronize to that bit rate during the frame synch prefix (provided in each ethernet frame for precisely this purpose). Once synchronized, receivers are able to sample the medium and recover the correct bits of the message. At this point there is a unidirectional synchronization of the two end systems with the transmitter's bit rate generator setting the timing for the entire message transmission. This "synchronization" is with respect to the bits of the message being transferred and there is no attempt to maintain any notion of absolute bit rate nor absolute accuracy. The receiver merely matches whatever frequency the transmitter happens to use (within a pre-defined specification range, of course) and a Manchester coded transmission format is used to maintain bit synchronization throughout the message. The level of precision that is maintained for the duration of the message is quite high — for a 100 Mbps network an individual bit time is 10 nsec and thus the phasing of the sampling of each individual bit must be maintained as a small fraction of that overall bit cell time. Indeed, the clock jitter specification for the physical layer chip we use is less than 1.4 nsec [1].

Both communicating systems have free-running local oscillators and software that is trying to keep track of each system's own version of UTC time. But there is never any attempt to link the local version of time to a transmitted frame nor to adjust a system's local time based on a received frame except for the statistics-based PLL implemented via Network Time Protocol exchanges. Through the use of hardware aids we can establish excellent time synchronization within a single LAN subnet. By similar modifications to hubs and switches one may in the future to be able to bring better synchronization capability to a larger intranet.

2.1 Understanding NTP Methods

Since the beginning of the project we have acquired, installed, and measured various versions of NTP running on SUN, PC, and StrongARM-based hosts under Solaris, LINUX, and NetBSD operating systems. As reported elsewhere we have been able to achieve agreement of UTC time across all of our hosts within 20-30 milliseconds. This work has provided a base dataset to begin our investigation.

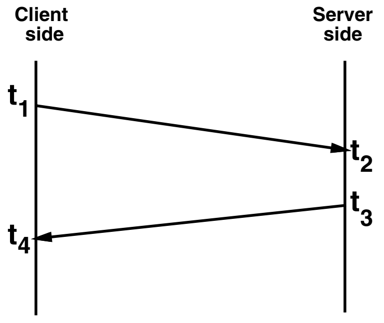


Figure 3: Time stamps in a typical NTP exchange

The project began by trying to understand how NTP works with a particular focus on which aspects of NTP serve to limit its accuracy. Figure 3 shows the four time stamps that are collected as a client's NTP request packet travels to and returns from a time server. With these four values, one can compute an estimate of the average path delay between client and host. Obviously, if the true communications delay between client and server were known (and constant), one could make the client's clock agree perfectly with the server's. The best one can do using this technique, is to compute an *average* of the two path delays (equation 1). It is not a bad assumption within a subnet, however, to assume that the client-to-server delay is approximately equal to the server-to-client delay.

$$t_{path} = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \qquad (1)$$

It can readily be verified that the calculation above yields its result in spite of differing clock settings on the client and server machines since any such host-to-host clock offset cancels. What isn't so obvious is the accuracy (or lack of it). In order to get a handle on that, one must look more deeply into how and when the timestamps are placed in the packet as it travels from client to server and back. Though not discussed in detail here we know that task priority and scheduling in each host, operating system queuing delays, and non-constant delays in accessing the ethernet cause large variations in time between the moment when the timestamp is sampled by the NTP daemon process and the time that the packet actually goes onto the ethernet. Similar variations in time delay occur between the arrival of an incoming NTP packet and the attachment of its time-of-arrival timestamp. It is because of these large time variations[2] that NTP can never act directly on the measurements contained in any *single* NTP packet Instead, each such packet merely contributes to on-going statistical calculations. These calculations add a great deal of complexity (and delay) to the programs that implement NTP. The delay arises because the statistics must be accumulated over a long enough period of time to yield meaningful results.

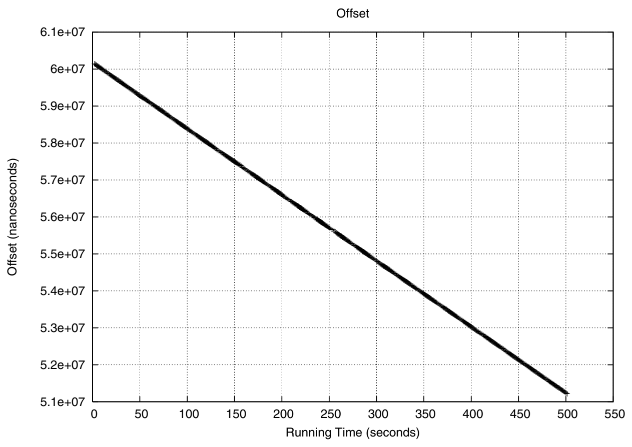


Figure 4: Raw time offset data — NTP not running

2.2 Measured Data

To establish a baseline, we captured timestamp data on packet exchanges both with and without NTP running. Figure 4 (above) depicts measured time offsets between two subject nodes on a local subnet. These same nodes will later be used as the local net's time server and client. The data in Figure 4 was taken by exchanging NTP packets but without processing them, i.e. the two nodes simply exchanged timestamps using NTP-style packets. No NTP daemon was running; no protocol processing happened. It is clear that the two system's local clock oscillators are, predicably, not running at precisely the same frequency. Thus, the offset between the machines changes by approximately 18 ppm (i.e. by 18 microseconds per second).

Continuing with the baseline data, we see that when NTP runs (Figure 5) it gradually tunes its statistics

[2]At this moment, the NTP daemon one of our SUN workstations reports an average path delay to our campus' time server of 0.03732 sec with a dispersion in that measurement of 0.03319 sec.

and learns how to take out the clock difference and make up for the frequency offset between the two systems. The data shown below was taken over a period of approximately 12.5 hours. In that interval NTP completed its convergence in 20000 seconds (5.55hrs) and shifted into more of a tracking mode after that. Even after waiting 24 hours we normally see time agreement only on the order of 500 microseconds. The accuracy of NTP's synchronization seems to wander quite substantially, due presumably to the variations inherent in NTP packet exchanges. Actually, since there is *no hardware output* that one can use to *directly* measure the true machine-to-machine synchronization, we simply use the raw timestamp data from the NTP packet exchanges to infer the clock offset.

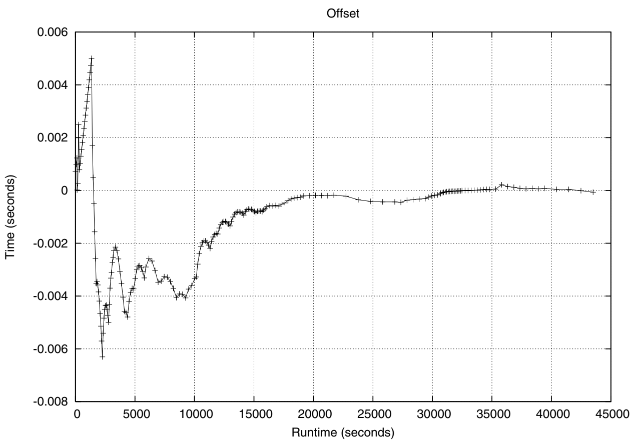


Figure 5: Time offset data — NTP tuning

2.3 Hardware Aids

As hypothesized in the early part of our project we have verified that it is the non-determinism in network and operating system latencies that ultimately limits achievable time synchronization. Accordingly, we developed hardware aids — modifications to a standard ethernet interface — that attempt to remove nearly all such non-determinism. A block diagram of our special ethernet interface is shown in Figure 6.

The strategy behind our design is to keep track of time *in hardware* on the ethernet card, using the 64-bit NTP format. By positioning our time register on the ethernet card it becomes available not only to the host and its software but also to the ethernet MAC chip. We have chosen to implement the

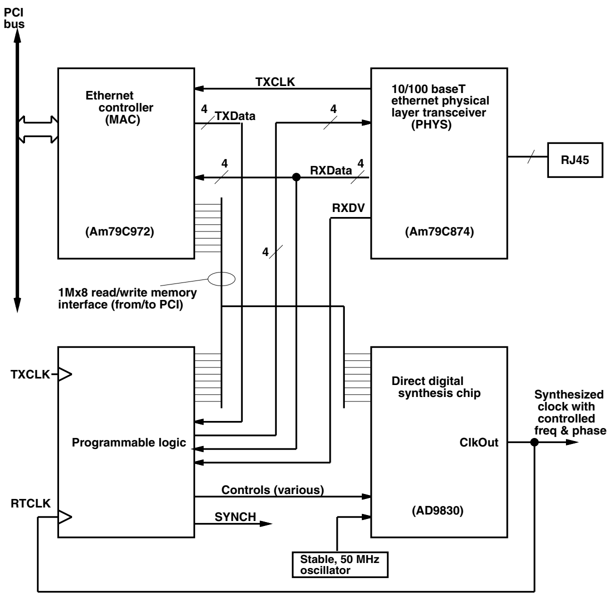

PCI bus
Ethernet controller (MAC)
(Am79C972)
TXCLK
TXData
RXData
RXDV
4
10/100 baseT ethernet physical layer transceiver (PHYS)
(Am79C874)
RJ45
1Mx8 read/write memory interface (from/to PCI)
TXCLK
RTCLK
Programmable logic
Direct digital synthesis chip
ClkOut
(AD9830)
Synthesized clock with controlled freq & phase
Controls (various)
SYNCH
Stable, 50 MHz oscillator

Figure 6: Block diagram of modified ethernet design

time register in programmable logic. The key relation to NTP is via interception of outgoing NTP packets, modification of their XMT timestamp field (to contain an updated time from our 64-bit time register), and latching of the time register to capture the time of arrival of incoming NTP packets. We have selected a particular media access controller chip (the Am79C792 made by AMD) and physical layer chip (the Am79C874) because they implement an industry-standard media-independent interface (MII) that provides a relatively easy intercept point. We pass the nibble-wide transmitted data stream through our programmable logic so that we can recognize and appropriately modify NTP packets while leaving all other packets unchanged. The NTP packet is shown in Figure 7 (the fields involved in recognizing it are highlighted and the two fields we modify are indicated in black). In addition to the fields shown in black, the ethernet packet's CRC-based frame check sequence (FCS) field located at the end of the transmission must also be recomputed and replaced.

Two key observations should be made about our transmit intercept technique. First, as long as our programmable logic keeps up with the 25MHz nibble clock, no timing changes are required — our logic is invisible to the MAC and PHY layer chips. Second, if our logic does not recognize a given packet as NTP, then no modifications are made and that packet goes

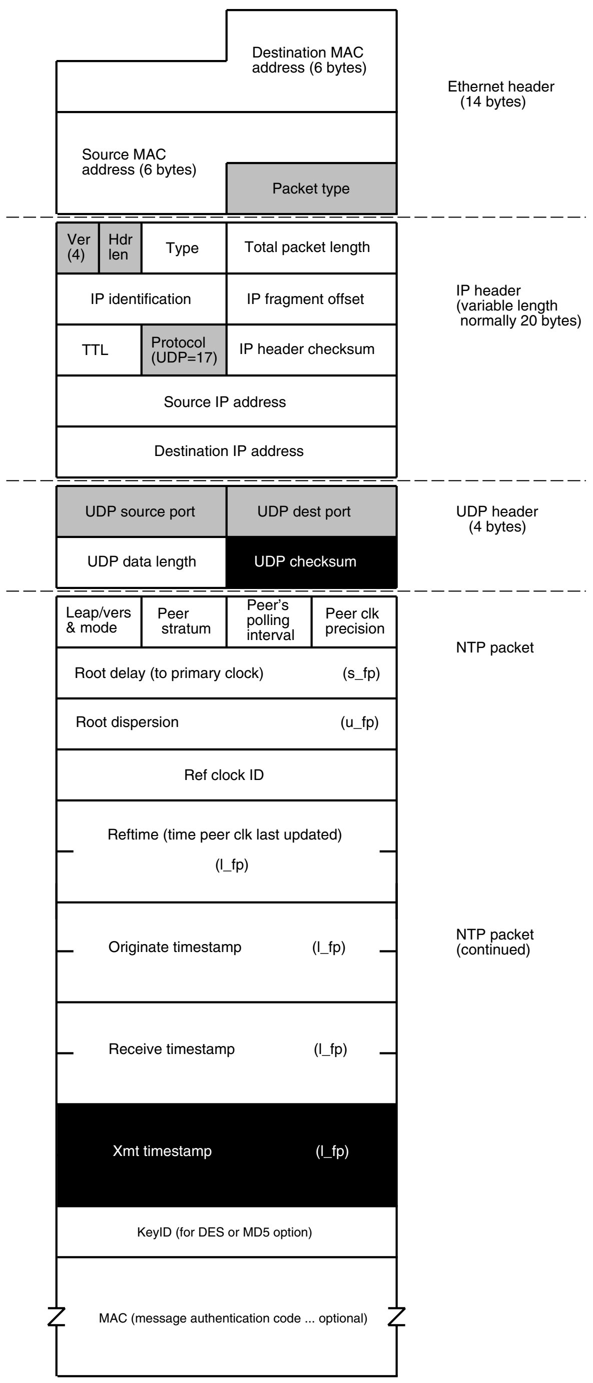


Figure 7: NTP packet layout. Highlighted (gray) fields are used to recognize the packet. Black fields are modified.

on its way without delay. Only NTP transmissions are modified and our modifications to those packets are merely selected updates to overwrite the current time within a field that already exists in all NTP packets. We have not modified the layout of the NTP packet in any way.

In the receiving direction, we had originally thought that it would be necessary to intercept and modify incoming NTP packet data as well. It is not strictly necessary to do that, however. It is sufficient merely to latch the time register when each NTP packet arrives. Ethernet drivers already contain code to fetch the operating system's notion of the current time and logically attach it to each packet. We modified the NTP client and server programs to fetch the latched version of the packet's arrival time from our ethernet card instead. This guarantees that the arrival-time timestamp no longer contains variations due to operating-system latency.

Through these relatively simple changes in hardware and software we have ensured that the outgoing and incoming timestamps on NTP packets have a very small, and *completely fixed latency* which an NTP-like algorithm is able to subtract out. With this approach we have achieved our goal of 3-4 orders of magnitude improvement in ethernet-based time sychronization (from milliseconds to hundreds of nanoseconds). The figure below shows the results of synchronizing with EtherSync. The system has achieved agreement in synchronization within 1 microsecond after only 90 seconds of running time.

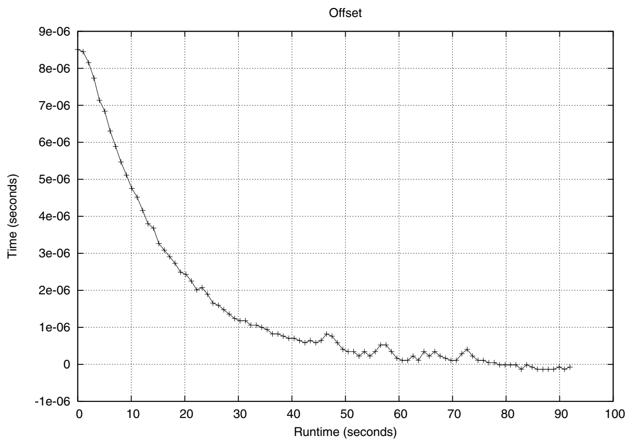


Figure 8: Time offset data — EtherSync tuning

2.4 Clock Adjustment

A direct digital synthesis (DDS) chip has been included in our modified network interface (shown at lower-right of Figure 6). DDS technology is the perfect fit to our adjustable timebase application. This chip makes it possible to enact very small, precise changes in the frequency and phase of the locally-generated timebase in order to bring it into close agreement with the timebase on one master unit. To see how this works, consider how our system initializes:

1. Just as in standard NTP, when a system boots it requests and receives the current time from a time server outside the subnet. There are many elaborate controls that make the choice of time server robust in spite of various network outages or other host downtimes. The idea is to get a rough copy of the present time, with an accuracy perhaps within 100 msec of the time server.

2. Using standard NTP methods we tune the frequency of what will soon be the local subnet's private time reference by tweaking the settings of our local DDS chip. The standard approach to avoid a lengthy NTP tuning cycle is to consult a disk file that contains the history of PLL settings from previous runs and to believe that the settings at this time will be very close to those that existed the last time this host synched up with the same time server.

3. Once the current time has been approximately established in our own accurate local frequency standard, we break the connection with the external time server and move into a highly-accurate local time-synchronization mode. In this mode, one machine on the subnet acts as a master time server. This machine does not make any changes to its local DDS chip. We use that machine's version of time as our reference for the subnet and all other machines on the subnet tune their local hardware clock to match that of the reference in frequency and phase[3].

4. Rather than using *software*-based encapsulation to discipline each host's clock, we utilize each node's local DDS chip to make very fine adjustments in both frequency and phase so as to bring all of the participating local clocks into alignment. We can do this because our modifications for the ethernet interface have removed nearly all of the non-determinism in NTP timestamping. Thus, each measurement can be acted upon directly to further tune the distributed PLL. Because we have chosen to operate our local tunable real-time clocks at a frequency of 8.388 MHz (2^{23} Hz) the resolution of our time synchronization is about 119 nsec. Future versions are planned with significantly higher precision.

We have implemented a vastly simplified version of NTP for use with our customized ethernet interface. The NTP message exchanges are identical to traditional NTP but since we no longer require the sophisticated statistics and clock source selection mechanisms, we have left those out.

3 The UCSB EtherSync Card

Figure 9 shows our implementation of the UCSB EtherSync board. This board is in every way a regular PCI-based 10/100 ethernet interface adapter. It can be used with the standard `pcnet32` driver and is compatible with all MS-Windows operating systems and with LINUX. In order to enable the special synchronization features we have modified the LINUX `pcnet32` driver and added a configuration-time `ethersync` option. We have also made modifications to the time-keeping portion of the LINUX 2.4 kernel. The modifications have been done in such a way that the special ethersync features can be enabled or disabled. The resulting system can be run in traditional non-ethersync mode (with or without the NTP daemon active). Our primary interest is on synchronization so we enable the special ethersync features and execute a custom time synchronization daemon. The custom software is similar to standard NTP but without any filtering or statistics. It is an extremely simple client/server arrangement where each client synchronizes its local clock to match that of the subnet's time server. Changes to the local

[3]Note that this is not nearly as drift prone as it may appear. NTP routinely "coasts" between time exchanges for relatively long periods (on the order of 1024 seconds). Highly stable crystal oscillators are available (for just a few dollars) with excellent frequency stability properties. It is the relative agreement of time between nodes that are part of the local subnet that is most significant. During a data acquisition or other process experiment it is appropriate to have our master time-keeping unit coast on its own highly-stable clock with all of the other units synching to it.

clocks are made by making very small changes in the frequency of the `RTCLK` via commands to the DDS chip.

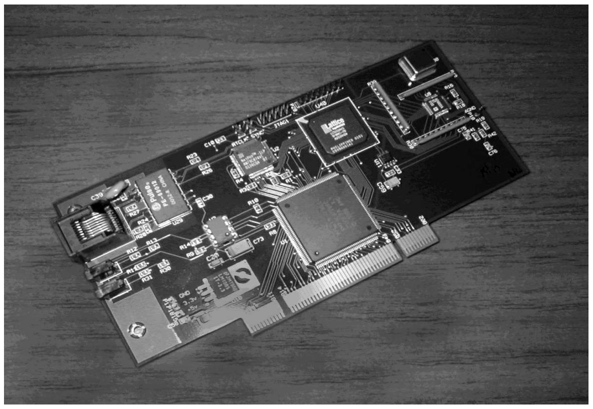
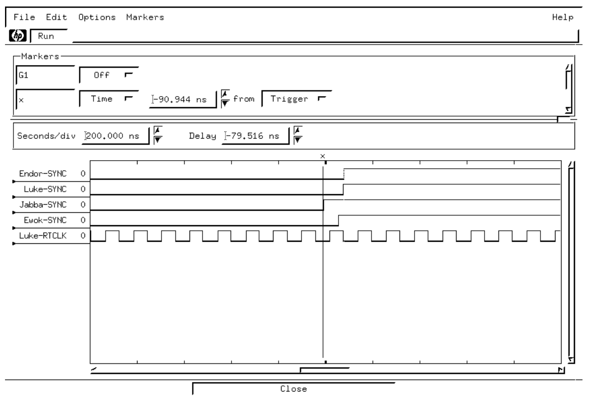


Figure 9: Photo of the UCSB EtherSync board, version 2.0

Figure 10 is captured from a logic analyzer showing the result of four EtherSync-connected systems attempting to synchronize an event. The EtherSync board has a hardware `SYNCH` output signal and it is this signal to which the logic analyzer was connected on all four systems. Each systems' `SYNCH` output was programmed to assert at precisely the same time of day using the standard NTP 64-bit time format. The logic analyzer captured the four `SYNCH` signals and displayed their arrival times. All were found to occur within 90 nanoseconds of each other. On the logic analyzer screen one can also see the real-time clock `RTCLK` output from one of the systems. This clock is always in-phase with that particular board's `SYNCH` output. Though Figure 10 shows only one `RTCLK` output, each board has its own copy. This real-time clock is provided for use by a local application's hardware. Note that the only connection between the four synchronized systems is a 100base-T ethernet connection to a standard hub. The four systems used were all x86 PC's running a modified Linux 2.4.17 kernel. The processors in the four systems were quite diverse in performance, including an Amd-Athlon at 900MHz, an Intel 686 at 450MHz, an Intel 686 at 400MHz and an Intel Pentium at 165MHz. During this test the four systems were exchanging NTP packets (with one system acting as master time server) every 4 seconds.



```
Markers
G1   Off
x    Time   -90.944 ns  from  Trigger

Seconds/div  200.000 ns      Delay  -79.516 ns
```

Figure 10: Four nodes synchronized over ethernet. Event synchronization is within 90 nsec. For reference, the period of the RTCLK waveform (bottom trace) is 119 nsec.

4 Summary

At its most basic level the NTP protocol is extremely simple. It works as well as it does because its statistics and continuous tuning allow each participating system to build a reasonably accurate model of the time offsets inherent in exchanging packets with a time server. Our philsophy (for ethernets) is equally simple: locate and remove as much as possible of the non-constant parts of host-to-host communications delays. We have done this by keeping an NTP clock register on the ethernet card where it can be automatically inserted into outbound NTP packets to ensure that they are accurately timestamped. In the receiving direction we have made provisions to latch a copy of our NTP time register whenever an NTP packet arrives, making it available to the software driver for incorporation into the time offset computation.

Application Scenarios

Consider the challenges of combining data taken by multiple banks of data acquisition equipment with the result from multiple video recordings of an event. Applications like this occur in the military, in medicine, and in many other realms.

In neurophysiology, an experiment might involve one or more cameras observing the motion of a knee joint in response to various stimuli. There would likely be numerous electrodes attached to the muscles around the knee as well as devices measuring various forces. In order to understand the interplay of signals in neurons with the various muscles and to correlate that with the muscles' interaction with the environment it will be necessary to establish a high degree of synchronization between the video camera(s) and the data acquisition systems.

Another scenario (this time from the military) might be several ground-based video cameras observing a drone as a missile or other "disturbance" interacts with it. There will always be many other channels of measurement (e.g. observing and recording signals sent to various control surfaces, rudders and other actuators) and the time synchronization of all of these systems – both video and non-video – will be essential in piecing together the measurements in order to form a cohesive understanding of the event.

Acknowledgements

This project was sponsored by R/C Electronics, Goleta, Ca. with matching funds from the State of California MICRO Program. Many thanks to Richard Renck of R/C Electronics for his participation. The board development work of Jayita Bhojwani and the protocol software provided by James Tandon is also gratefully acknowledged.

References

[1] Advanced Micro Devices, "Am79C874 Datasheet", Publication #22235 Rev. H, Amendment /0, June, 2000.

[2] DARPA, "Internet Control Message Protocol", RFC-792, Sept. 1981.

[3] Mills, D., "Network Time Synchronization", RFC-1129, October, 1989.

[4] Mitra, D., "Network Synchronization: Analysis of a Hybrid of Master-Slave and Mutual Synchronization", IEEE Trans. Communications, COM-28, 8 (Aug 1980), pp. 1245-1259.

[5] Postel, J., "Daytime Protocol", RFC-867, May 1983.

[6] Postel, J., "Time Protocol", RFC-868, May 1983.

[7] Su, Z., "A Specification of the Internet Protocol (IP) Timestamp Option", RFC-781, May 1981.