Synchronous data collection from diverse hardware

Camille Goudeseune, Braden Kowitz
Beckman Institute, 405 N Mathews St, Urbana IL 61801 USA

Abstract

We describe an accurate distributed timestamp service. This open-source C++ package runs on commodity PC's. With no extra hardware, the service correlates sensor data (head- and eye-trackers, biometrics, captured video, driving simulator data) from multiple PC's with sub-millisecond accuracy. PC-driven actuators like motion bases and audio/visual/haptic warning systems are also controlled with the same accuracy. This lets us accurately measure driver response time (brake at a stoplight, direct gaze at a hazard, answer a telephone). Hardware vendors often assume that system integration revolves around their own devices. This service synchronizes devices despite such assumptions. It is orders of magnitude more accurate than conventional clock-synchronizing methods in Microsoft Windows. A Linux master clock provides a stable NTP time base. Slave clocks, Linux or Windows, synchronize to the master clock by several mechanisms. Measuring round-trip ping times corrects for network latency. In the slave's high-resolution clock, drift is predictively compensated for while trapping wraparound and jitter (e.g., from PCI bus contention). Performance degrades gracefully and measurably on heavily loaded networks. Several phase-locked loops, within each slave and between slave and master, guarantee performance.

Résumé

Nous décrivons un service précis d'horodateur distribué, par C++ en source ouvert, pour les PCs ordinaires. Sans matériel supplémentaire, ce service corrèle des données des senseurs (traqueurs de tête et d'oeil, biométrie, vidéo capturée, simulateur) des PCs multiples avec exactitude de moins d'une milliseconde. Les dispositifs conduits par PC comme des bases de mouvement et des systèmes d'avertissement audio/visuel/haptique sont également commandés avec la même exactitude. Ceci nous laisse exactement mesurer le temps de réponse de conducteur (le frein à un feu d'arrêt, regard fixe direct à un risque, réponse à un téléphone). Les fournisseurs de matériel supposent souvent que l'intégration de système tourne autour de leurs propres dispositifs. Ce service synchronise des dispositifs en dépit de telles prétentions. C'est des ordres de grandeur plus précis que des méthodes de synchronisation conventionnelles dans Microsoft Windows. Une horloge principale dans un PC Linux fournit une base stable de temps de NTP. Les horloges esclave, Linux ou Windows, synchronisent à l'horloge principale par plusieurs mécanismes. Mesurant les périodes aller-retour des paquets corrige pour la latence de réseau. Dans l'horloge à haute résolution de l'esclave, la dérive est compensée prédictivement. L'exactitude dégrade mesurablement et avec élégance sur les réseaux fortement chargés.

1. Introduction

1.1 Environment

The Integrated Systems Laboratory (ISL) is a Beckman Institute facility at the University of Illinois at Urbana-Champaign. ISL's mission is to advance scientific understanding of human-computer interactions. We meet this goal in part by integrating advanced technologies so that Institute researchers can conduct experiments in human multi-modal perception and cognition. The ISL operates several Virtual Reality (VR) simulators for the research community including the Beckman Institute Driving Simulator (BIDS), a fully enclosed immersive 3D chamber, and many smaller facilities.

1.2 Problem

Most simulator systems have requirements to precisely measure timing between events. Event types are highly varied and can be anything from an eye movement to a warning buzzer.

Measuring event times is relatively simple when a single computer controls all aspects of the simulation: the controlling computer queries its local clock whenever an event occurs. However, experiments using multiple computers are becoming increasingly common. This change is driven primarily by two factors:

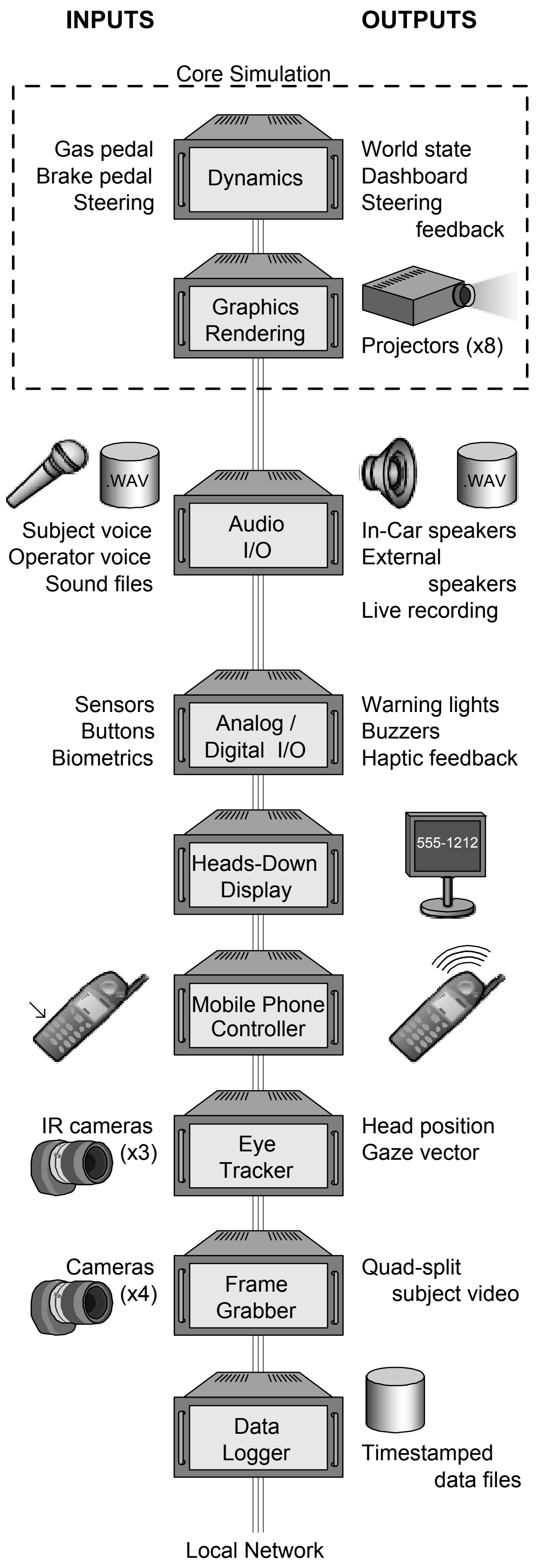

Figure 1. Network of PCs with inputs and outputs used in the BIDS driving simulator.

(1) VR simulations have historically been run on a single specialized graphic supercomputer. Today, networked clusters of commodity PCs deliver the same graphics performance at a fraction of the cost [Francis et al 2003].

(2) The decreasing cost of computational power is driving many specialized hardware components to software implementation. This change in research equipment is especially visible in eye- and motion-tracking systems.

Because of these forces, many research environments now use networked PC's working together to run an experiment. BIDS is an example of such a system where separate computers are responsible for different parts of the simulation (Fig. 1). In BIDS, one computer plays warning sounds while another monitors the brake pedal position. Each computer logs events with a local timestamp for later analysis. But in order to properly correlate data from different machines, all computers need access to a common time source.

1.3 Goals

Our aim is to develop a software component that provides global timestamps using a local area network. All delivered timestamps must be accurate to within 1 msec of each other. In addition, the solution must not require any specialized equipment apart from conventional networking hardware. This timestamp service must easily integrate with a variety of existing proprietary systems.

2. Using the timestamp service

The timestamp service is most useful in measuring a subject's response time (RT). In an experimental trial, subjects are presented with various stimuli and may respond in various ways (Table 1). RT is the elapsed time between stimulus presentation and response initiation. Researchers analyze variation in RT with respect to independent variables such as the subject's age, traffic density, or attention loading with secondary tasks. This necessitates accurate measurements of RT.

Commonly, RT is measured by logging an event with its simulation frame number. VR simulations typically run at 60 frames per second (16.7 msec per frame). It is not unusual for a scenario to have a RT of about 100 msec. If such a scenario's RT is clustered in the interval [80, 130] msec, frame-based timing offers only three distinct values for analysis. A millisecond-accurate timestamp service offers 50 distinct values in this range. The strength of statistical conclusions correspondingly improves.

Table 1. Example scenarios for measuring a subject's response time.

Stimuli	Responses
• Voice prompt "change lanes"	• Turn steering wheel past threshold • Lane deviation exceeds threshold
• Other car "cuts off" subject's car	• Clench hands on steering wheel • Release gas • Depress brake • Direct gaze at car
• Head-down display shows phone number	• Speak the phone number
• Subject's speed exceeds 100 km/h • Voice prompt "please slow down"	• Release gas pedal • Speed falls below 100 km/h

3. Installing the timestamp service

The timestamp service is quite easy to use. It builds and runs under Windows, Linux (Fedora and Debian), OpenBSD, and MacOS X. The time provider server runs on a non-Windows machine as a user level process. We have taken care to ensure ease of use with client integration. The native interface is a C API, with wrapper modules for TCL, Python, Ruby and Java.

We have used the C API to integrate our timestamp service with eye trackers [Smart Eye AB 2004]. However, even novice developers can use the service through its wrappers for scripting languages. For example, integration with our driving simulator [DriveSafety 2004] required only three steps. First, we copied the shared libraries (DLL's) into a system folder. Next, we created a text file listing configuration options like the time server address and error tolerances. Finally, a few lines of TCL code let each scenario access the timestamps.

```
load /usr/local/lib/clockkit_tcl.so
    clockkit
ckInitialize
VTriggerCreate eachFrame {
    SimSetUserData [ckTimeAsString]
}
VTriggerAdd eachFrame 60 Hz
```

4. Previous work

One-time calibration

In an ideal world we could synchronize the system clocks on each machine just once. Unfortunately, affordable clocks do not keep exact time. High-quality quartz crystals are used in PC system clocks. These room temperature crystal oscillators (RTXO's) typically have a thermal drift of about 1 to 2 PPM/°C, though this is not linear and varies considerably from one crystal to the next. Smaller drifts come from mechanical shock, unregulated line voltage, humidity, barometric pressure, vibration, and simple aging. Industry crystals typically advertise a frequency stability of 20 to 100 PPM within a given temperature range. Inexpensive crystals common to budget PCs have tolerances worse than 100 PPM [Agilent 2004, Martinec 2004].

All this means that a one-time calibration cannot keep several clocks running in agreement with each other for any useful length of time. For example, even with an initial frequency calibration, a temperature change of only 2 °C can cause a 1 msec disagreement within 10 minutes. Regular recalibration is needed to compensate for frequency drift.

Radio clocks

One way to provide timestamps is to send a clock signal to each machine. Specialized devices that receive time broadcasts via GPS or CDMA can be attached to each PC. Unfortunately, this specialized hardware costs several hundred dollars per PC and requires antennas within range of the timing signal. Other specialized hardware solutions exist, but since each PC is typically connected to an Ethernet network, it is desirable to use the existing network links to keep the clocks synchronized.

Windows Time Service

When dealing with Windows exclusively, it may be tempting to use the Windows Time Service. This service is built into Microsoft operating systems since Windows 2000 and promises to synchronize system clocks over the network. Unfortunately it "provide[s] clock values that are 'loosely synchronized' across a network. This service is not designed for use by applications that require greater precision" [Brandolini and Green 2001]. This method clearly does not provide the accuracy we need.

Master Clock Ticks

One can distribute timing signals to a computing cluster by propagating master clock ticks over a network link. In this method, one computer generates tick signals at a specified rate. The ticks are then typically propagated over network links as UDP broadcast packets. When a client machine receives such a packet, it recognizes that a tick has occurred. This method has been used to successfully synchronize the timing of distributed graphics rendering [Francis et al 2003, Papelis 2003]. Unfortunately, this method is less reliable than using a local clock. UDP packets can be dropped or delayed in transit, causing timing errors of several milliseconds. Such errors cause a slight amount of jitter or dropped frames, which are nearly invisible in graphics rendering but unacceptable for measuring response time.

Network Time Protocol

In many situations, the Network Time Protocol (NTP) daemon solves the timestamp problem perfectly. The daemon uses network communication to discipline the local system clock to UTC (Universal Coordinated Time). Although NTP runs on all major operating systems, the Windows implementation does have some limitations. Since it is often impossible to avoid Windows when integrating new hardware, we must carefully consider these limitations.

The NTP project claims to keep a Windows clock synchronized to within 0.5 msec [Dietrich and Mayer 2003]. Although NTP can keep the system clock accurate, the clock itself has a very low resolution. It updates the clock value, or ticks, once every 10 to 15 msec. This means that if an event is measured at time *t*, the clock can report a value anywhere in the range [*t* – 15 msec, *t*].

Clearly, we cannot use the Windows system clock to generate timestamps. On other operating systems, such as Linux, the system clock has sub-millisecond resolution; also, applications can request the current time directly from the NTP daemon instead of from the local clock. Doing so increases precision and also provides error bounds via the NTP_GETTIME API along with the current time. Unfortunately, the Windows implementation does not yet support this API [Dietrich and Mayer 2003].

Fortunately most Windows systems provide a high-resolution clock known as the performance counter. This counter cannot be disciplined to the proper time or frequency, and often runs at frequencies very different from that reported by the API. By itself, the performance counter is unsuitable for generating timestamps. It can however be used to interpolate between system clock ticks [Nilsson 2004]. When a system clock is disciplined by NTP, this interpolation method should provide timestamps that are both accurate and precise.

Unfortunately, we have found that performance counter interpolation of an NTP-disciplined system clock does not provide sufficiently accurate timestamps. The offset between NTP time and the system clock is typically ±1.5 msec on our Windows PC's, even with low network delay and jitter (3.5 and 0.2 msec respectively).

Besides insufficient accuracy, there are several other reasons we have chosen not to use the previously described method. Integration is a major consideration for a timestamp service. In order to provide timestamps using NTP, the local clock must be disciplined to UTC. But adjusting the local clock may not be desirable. For example, licensing bugs have at times forced us to set system clocks incorrectly as a temporary workaround. This method is incompatible with NTP synchronization. Furthermore, adjusting a PC's local clock requires root or administrator access, which local policy may forbid to users in a shared computing environment. Instead of changing the local clock, we compute a transform function between the local and reference clocks. When a timestamp is requested in this situation, the transform function generates an accurate timestamp from the local clock. This means that a mere user-level process can provide accurate timestamps, and that separate processes can synchronize to different sources.

NTP may seem to be a simpler solution. However, we must be able to receive accurate timestamps from within various programming languages including C, C++, Java, Ruby, TCL, and Python. Because some of these languages lack built-in support for accessing the system clock with sub-millisecond accuracy, we would need to build modules to support high resolution clock access. In fact, because of its low resolution system clock, Windows would need modules for all languages.

In addition to these concerns, many NTP features are unnecessary to us. Instead of accurate UTC time, we only need precise time relative to participating PC's. We do not need long-term reliability during network failures, since in such a situation the distributed simulation itself fails. Our simulations run on a private network so we need no authentication, security, or handling of byzantine failures. Finally, our facilities have at most a few dozen PC's so overloading a time provider is not a concern.

5. Design of the timestamp service

The system clock on a Linux PC is used as a reference clock. It can optionally be disciplined with NTP to increase its accuracy.

Each client needs access to a stable, high-resolution time source. Windows machines use the performance counter; other operating systems use the system clock.

Cristian's algorithm measures the difference between local clocks and the reference clock [Cristian 1989]. Our implementation of this algorithm sends UDP packets to and from the reference clock. Each clock skew measurement is accompanied by calculated error bounds. These error bounds vary linearly with the observed round-trip time (RTT) of the packets, so a low-latency network improves skew measurements.

A simple phase-locked loop computes the transform between the local and reference clocks. Our implementation uses two variable-frequency clocks connected in series (Fig. 2). The first clock measures phase changes every 30 seconds to discipline frequency, while the second measures phase once per second to adjust for clock skew. The two degrees of freedom provided by the unusual two-clock design lets us independently model (and correct) both frequency and phase.

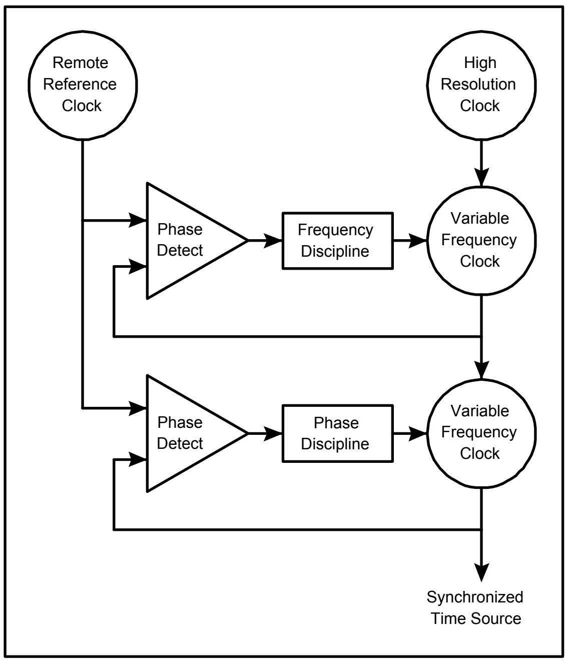

Figure 2. Disciplining the clock.

Since each phase measurement from Cristian's algorithm has error bounds, we can adjust frequency and phase conservatively. If a low-RTT packet arrives, our algorithm uses it to very accurately adjust for clock skew. If the next packet has a high RTT, then the measurement error likely outweighs any information in the packet. This method keeps the clock synchronized during periods of bursty network traffic. Clock frequency is measured over longer periods, and thus is less sensitive to RTT variation.

Error estimates

The generated timestamps would be worthless without guarantees of accuracy. Fortunately, we can compute very good estimates regarding the accuracy of the timestamp service. Four pieces of information provide our error estimates: (1) clock skew at last measurement; (2) clock skew measurement error; (3) frequency measurement error; and (4) assumptions about the variability of the local time source. The worst-case error estimate begins with the skew measurements (1) and (2). Frequency errors from (3) and (4) accumulate over time since the last phase measurement.

Simulation of this algorithm shows that an unloaded 10base-T link supports timestamps with an average error bound of 200 μsec, well within our 1 msec goal (Fig. 3).

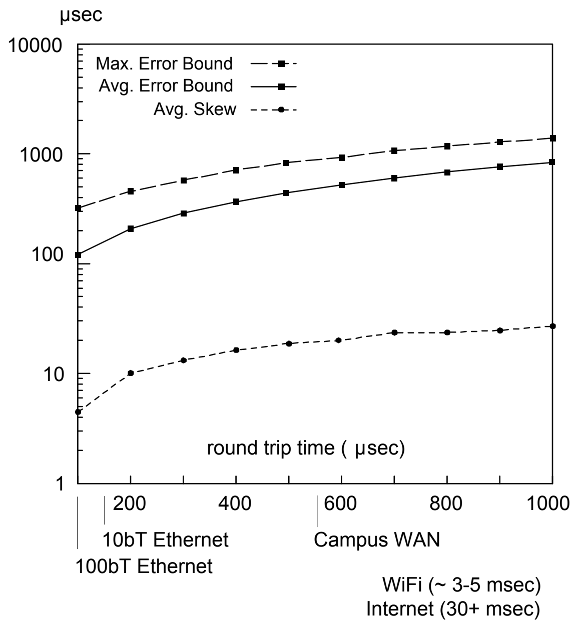

Figure 3. Clock accuracy as a function of packet round-trip time (simulated).

We have tried to make the time-stamp service simple to use. Since researchers already have enough data to analyze without considering timing inaccuracies, error bounds are not supplied by default with each timestamp. Instead, each PC has a configuration file specifying an acceptable error bound. If a PC's timing error exceeds this limit for some reason, the service changes to an "out of sync" state in which it stops providing timestamps until the timing error returns to an acceptable value. With this model, users know that all delivered timestamps are accurate within a guaranteed threshold.

Clock design

The timestamp service code is written in the C++ language, using namespaces to simplify integration. Exceptions are used extensively for sanity checks on conditions like nonmonotonic clock motion.

The source is built on top of the GNU Common C++ library, which provides OS-independent high-level abstractions for threading and networking. These abstractions ease the process of porting to various operating systems. It also helps to keep the size of the program very small. At less than 3000 lines, the source code is easy to inspect for errors.

Finally, we used the Simplified Wrapper Interface Generator (SWIG) to easily create modules for TCL, Ruby, Python, and Java.

6. Conclusions

The timestamp service described here provides a source of accurate timing data to a wide range of PC-based equipment added to a driving simulator. These timestamps are particularly useful for measuring a subject's response time when the stimulus is presented by one PC and the response detected by another.

No special hardware is needed other than local Ethernet connectivity. No unusual software requirements, such as real-time priority or administrator privileges, are needed. Timestamp accuracy can be adjusted to a level supported by the network, typically under a millisecond. Timestamps are guaranteed to be within this accuracy bound.

The implementation is open source, and available from http://www.isl.uiuc.edu/.

7. References

Agilent Technologies. 2004. *Timebase oscillator calibration*. <http://metrologyforum.tm.agilent.com/xtals.shtml>.

Brandolini, S., and D. Green. 2001. *The Windows Time Service*. <http://www.microsoft.com/windows2000/techinfo/howitworks/security/wintimeserv.asp>.

Cristian, F. 1989. "Probabilistic clock synchronization." *Distributed Computing* 3, 146–158.

Francis, G., C. Goudeseune, H. Kaczmarski, B. Schaeffer, and J. Sullivan. 2003. "ALICE on the eightfold way: exploring curved spaces in an enclosed virtual reality theater." In *Visualization and Mathematics* III, H.-C. Hege and K. Polthier, eds., Springer, 2003, 307–317.

Dietrich, S., and D. Mayer. 2003. *NTP 4.x for Windows NT.* <http://www.eecis.udel.edu/~mills/ntp/html/build/hints/winnt.html>.

DriveSafety 2004. *Company website.* <http://www.drivesafety.com>.

Martinec, M. 2004. *Time, with focus on NTP and Slovenia*. <http://www.ijs.si/time/>.

Nilsson, J. 2004. "Implement a continuously updating, high-resolution time provider for Windows." *MSDN Magazine* 19(3). <http://msdn.microsoft.com/msdnmag/issues/04/03/HighResolutionTimer/>.

Papelis, Y. 2003. "Performance evaluation of a framework for distributed real-time driving simulation applications using Windows based PCs." *DSC North America 2003 Proceedings.*

SmartEye AB 2004. *Company website.* <http://www.smarteye.se>.