Measuring Microsecond Delays

Technical Report TUD-FI03-16-November-2003

Jork Loeser

TU Dresden, Germany
contact@os.inf.tu-dresden.de

Abstract

In this paper we present a setup to synchronize the clocks of two nodes. We especially derive bounds of the achieved clock accuracy and examine implementation aspects.

1 Motivation

In other experiments, we want to measure transmission delays for packets sent over a network between two nodes, as depicted Figure 1. To measure these transmission delays, we use test packets carrying timestamps. A send application at node **A** generates the test packets and adds timestamps. At node **B**, a receiving application compares the timestamps with its local clock and calculates the transmission delay.

We want the delay measurement to be easy to implement and to rely on as little hardware and driver support as possible, as the driver support in our environment is limited. We also want the delay measurement to be efficient, thus we do not want to rely on external clocks which are expensive to read. Our experiments are done on x86-based nodes that offer a *time stamp counter* (TSC), a counter running at CPU speed counting the cycles of the CPU [4]. The TSC can be read within 80 cycles on an Intel P4 1.7GHz, this is about 47ns, and in 32 cycles on other Intel x86 machines (P3, P2). Intels SpeedStep technology, when used, influences the TSC, however in our experiments we do not use SpeedStep.

Due to different CPU speeds and fabrication tolerances the CPU clocks of different nodes run with different speeds[1]. To calculate the delay based on node **A**'s timestamps and node **B**'s time, both nodes must be synchronized. For our experiments we expect packet transmission delays in the order of microseconds to a few milliseconds, and thus the synchronization accuracy has to be a few microseconds. This accuracy can not be achieved by the network that connects nodes **A** and **B**. Therefore we connect node **A** and **B** by an additional parallel cable and use this for the synchronization process.

[1] A 1% deviation of CPUs with the same nominal speed is quite usual.

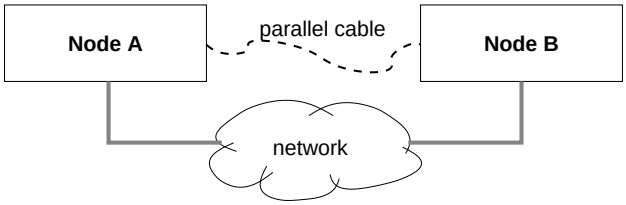

Figure 1: Nodes **A** and **B** connected to a network. Additionally, they are connected by a parallel cable for synchronization.

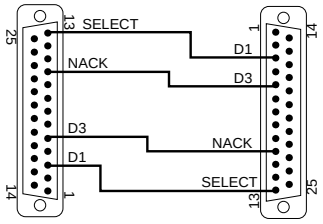

Figure 2: Wiring of the parallel cable (used data pins only)

This paper describes

- how the synchronization procedure works
- how node **B** converts **A**'s timestamps to **B**'s local time
- what accuracy can be achieved
- what resources are consumed by the synchronization process

2 Parallel Port Issues

The parallel port of a PC style computer, originally designed to send data to a printer, has 12 output pins and 5 input pins. The state of the output pins can be set by programmed port I/O, and the state of the input pins can be read by programmed port I/O. One of the input pins (NACK) can be programmed to raise an interrupt if the signal value changes [1].

We connect the parallel ports of both nodes as shown in Figure 2. As a result, we can transmit 2 bits in parallel, and one bit can be used to raise an interrupt at the remote node.

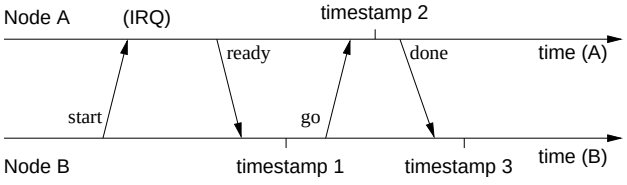

Figure 3: Synchronization process

Signal delays

We did experiments to find out how long it takes that changes at input pins can be seen after the corresponding output pin was toggled. Therefore we used a loop-back dongle connecting output pin D1 of one node to its own SELECT input pin. Then we toggled D1 and measured how long it takes for the SELECT input value to change.

We did the measurements on different PC style computers. The observed delays are quite independent of the nodes CPU speed, as they mainly depend on port-I/O commands of x86 CPUs (there execution times are quite stable among different CPUs) and on the slow internal ISA bus the parallel port is connected to (on modern computers this ISA bus is emulated, but it is still slow). The observed delays are all about 2µs.

3 Synchronization Procedure

The synchronization procedure is very similar to Cristians internal clock synchronization or to that of the server synchronization in [5] used by NTP as defined in RFC 1305: Node **B** periodically raises an interrupt at **A** to start the synchronization process. In reaction, **A** sends a ready-signal and both nodes take timestamps. Later, node **A** sends its timestamp to **B** which uses it to calculate the clock difference and the clock drift.

3.1 Getting the timestamps

The synchronization process is shown in detail in Figure 3 and Table 1. The interrupt is raised by **B** by changing its *D3* output pin from 0 to 1. This sets the *NACK* input pin at node **A** to 1 and raises an interrupt at node **A**. The *ready signal* of node **A** means setting *D3* at **A** to 1. **B** polls its *NACK* pin waiting for the *ready signal*. Once **B** sees the *ready signal* it takes timestamp 1 from its local CPU timestamp counter and sets the *D1* output pin to 1 (*go signal*). This is observed by **A** polling its *SELECT* input pin. **A** takes timestamp 2 and sends the *done signal*. This means setting the *D1* output pin to 1. **B** polls its *SELECT* pin until it becomes 1 and takes timestamp 3 then. Finally, **B** resets *D1* and *D3*.

It may happen, e.g. due to an unplugged parallel cable, that the signals sent by one node are not seen by the other node. Therefore we guard all poll-operations by timeouts. We choose 20µs, this is we ensure that all poll operations together do not run longer than 20µs. It may happen, e.g. due to other interrupts, that one node cannot react within these 20µs. Then, the other node times out and the synchronization process is aborted.

node **A**	node **B**
	start signal: B-D3:=1
A-NACK==1, interrupt *ready signal*: A-D3:=1 A-D1:=0	
	poll: B-NACK==1 timestamp 1 *go signal*: B-D1:=1
poll: A-SELECT==1 timestamp 2 *done signal*: A-D1:=1 A-D3:=0	
	poll: B-SELECT==1 timestamp 3 B-D1:=0, B-D3:=0

Table 1: Synchronization process

3.2 Determining Time and Frequency

3.2.1 Clock Offset

It is easy to see and explained in detail in [5] that the clock offset Θ from node **A** to **B** during the synchronization process is given by

$$diff - jitter \leq \Theta \leq diff + jitter \qquad (1)$$

with

$$diff = \frac{timestamp_1 + timestamp_3}{2} - timestamp_2$$

and

$$jitter = \frac{timestamp_3 - timestamp_1}{2}$$

3.2.2 Compensating different clock speeds

When doing the synchronization process twice, the different clock speeds at **A** and **B** can be compensated. Therefore, we define some variables. Index X refers to values taken from the first synchronization process, index Y refers to values from the second synchronization process.

$$time_X = \frac{timestamp_{1,X} + timestamp_{3,X}}{2}$$

$$jitter_X = \frac{timestamp_{3,X} - timestamp_{1,X}}{2}$$

$$remote_X = timestamp_{2,X}$$

$$time_Y = \frac{timestamp_{1,Y} + timestamp_{3,Y}}{2}$$

$$
\begin{aligned}
jitter_Y &= \frac{timestamp_{3,Y} - timestamp_{1,Y}}{2} \\
remote_Y &= timestamp_{2,Y} \\
conv &= \frac{time_Y - time_X}{remote_Y - remote_X}
\end{aligned}
$$

conv allows to convert timestamps measured at node **A** to the time metric at node **B**. If *remote* is a timestamp value measured at node **A**, then the conversion to **B**'s time *local* is done by:

$$local = conv * (remote - remote_Y) + time_Y \quad (2)$$

The timestamps $remote_X$ (and $remote_Y$) were taken at **A** when the time at **B** was in the interval of length $jitter_X$ centered around $time_X$ ($jitter_Y$ centered around $time_Y$). We define $drift$ as

$$drift = \frac{jitter_X + jitter_Y}{time_Y - time_X} \quad (3)$$

Then, the error of equation (2) due to jitter in the measurements is bound by

$$precision = (local - time_Y) * drift + jitter_Y \quad (4)$$

This error solely covers measurement errors due to signal transmission times, but does not take real clock drifts due to heating or other physical effects into account.

3.2.3 Clock drift

Real numbers on the drift of a processor clock are hardly to find in mainboard or processor specifications. However, quartz oscillators in general are known to drift with rates between 10^{-7} to 10^{-10} (e.g., [3, 6]).

In an experiment we measured the drift rate of a clock in a PC-style node against the clock in another PC-style node. The experiment was done over multiple hours with different load and temperature conditions at the nodes. In this experiment we found the clocks to drift at a maximum rate of 10^{-7}.

We conclude, that an estimation of 10^{-6} for the drift of the clock in node **A** against the clock in node **B** is a safe value. This drift has to be added to that in equation (3).

4 Implementation Aspects and Error Bounds

We implemented the synchronization process on DROPS [2] for measurement purposes. DROPS is based on the L4 micro-kernel offering static priorities. By giving the synchronization programs a high priority we ensured that it is interrupted rarely.

The precision is calculated and returned to the programs as well, so they can verify at runtime that their timestamp conversions deliver sensible results. We consider this an important robustness issue in experimental environments: It is too easy to use a system out of its specifications in the heat of experiments and rapidly changing test programs.

4.1 Integer Arithmetic

An important requirement to the delay measurement is its efficiency, and therefore we avoid floating point arithmetic. Instead, we use 64bit fix point arithmetic, with 32bit precision for storing the scalers *conv* and $drift$. The timestamps and the computed values of *local* and *precision* are stored with 64bit precision. We use `conv` and `drift` to denote the 32bit integer values. We use `local` and `precision` to denote the 64bit integer representations of *local* and *precision*.

We assume the clock frequencies of the nodes **A** and **B** to be between 50MHz and 4GHz. We calculate `conv` as

$$\texttt{conv} = \frac{(time_Y - time_X) * 2^{25}}{remote_Y - remote_X} \quad (5)$$

Conditions for `conv` With the CPU clock frequencies in the range between 50MHz and 4GHz, `conv` is at least 419430, and less than 2^{31}. It can be computed without overflows as long as $time_Y - time_X$ is less than 2^{39}. For a 4GHz node this is about 100s, for a 50MHz node this is over 2 hours.

We calculate `drift` as

$$\texttt{drift} = \frac{(jitter_X + jitter_Y) * 2^{32}}{time_Y - time_X} \quad (6)$$

Conditions for `drift` From Section 3.1 we know that $jitter_X$ and $jitter_Y$ correspond to values less than 20µs. With a minimum pause of 100ms between the two measurements determining $time_X$ and $time_Y$ the value of `drift` is less than 2^{21}.

To convert a remote timestamp taken at node **A** to the time metric at node **B**, we adapt equation (2):

$$\texttt{local} = \frac{\texttt{conv} * (remote - remote_Y)}{2^{25}} + time_Y \quad (7)$$

Conditions for `local` Using the same argumentation as we used for `conv` we see that the computation of `local` does not overflow as long as the time that passed between measuring $remote_Y$ and *remote* does not exceed 100s. In

that case, `local` will be less than 2^{39}, and the numerator in equation (7) cannot overflow.

As `conv` is rounded to an integer, it might be up to 1 off from the real value. Hence, the arithmetic error in (7) due to the integer operation is bounded by $(remote - remote_Y)/2^{25}$. We adapt equation (4) to

$$\texttt{precision} = \frac{(local - time_Y) * (\texttt{drift} + 1)}{2^{32}} + jitter_Y + \frac{(remote - remote_Y)}{2^{25}} \quad (8)$$

The addition of 1 to `drift` covers the arithmetic error in the calculation of `drift`.

Conditions for `precision` We know that `drift`$\leq 2^{21}$. Using a simular argumentation as we used for `conv` we conclude that $local - time_Y \leq 2^{39}$ and hence the first numerator cannot overflow. The other terms of the sum cannot overflow either.

4.2 Errors due to Integer Arithmetic

The error due to rounding of `conv` is $(remote - remote_Y)/2^{25}$. The worst case relative error is achieved with the remote node running at 4GHz and the local node running at 50MHz: The relative error is bound by $4GHz/(2^{25} * 50MHz) = 2.4e-6$ then. However, on nodes with the same speed, the relative error is $2^{-25} = 3 * 10^{-9}$.

The error due to rounding of `drift` is $(local - time_Y)/2^{32}$, which of course is covered by equation (8). The relative error is $1/2^{32} = 2.3 * 10^{-10}$.

For nodes with similar clock frequency, and more than ever for a slower remote node, the arithmetic errors are orders of magnitude smaller than the CPU drift. Consequently we neglect the arithmetic error on node setups like these.

4.3 Errors due to Measurements

From Section 3.1 we know that $jitter_X$ and $jitter_Y$ are less than 20µs, resulting in an absolute error bound of 30µs (equations (3) and (4)). However, almost all of the measurements can be finished within 2µs, and by selecting the good values the error drops to 6µs.

We normally synchronize every 2 seconds. However, if we have to drop a value, or if the process even aborts, we do the next synchronization after 1 second. It turned out in experiments that even in situations with high load no more than 3 attempts were necessary for a successful synchronization. Thus, the error bound is 4s*10^{-6}+6µs=10µs.

5 Summary

We presented an easy-to-use implementation for internal clock synchronization of 2 nodes. By using a parallel cable between the two nodes we are able to achieve error bounds of 10µs.

The synchronization runs not more than every second, and mostly every two seconds. A synchronization does not take longer than 20µs.

The fix-point arithmetic uses integers values scaled for CPUs with clock frequencies between 50MHz and 4GHz. Then, the synchronization must be executed at least every 100s, but not more often than every 100ms. The time between the last successful synchronization and the conversion of a timestamp must not exceed 100s.

References

[1] Jan Axelson. Parallel Port Central. `http://www.lvr.com/parport.htm`.

[2] H. Härtig, L. Reuther, J. Wolter, M. Borriss, and T. Paul. Cooperating resource managers. In *Fifth IEEE Real-Time Technology and Applications Symposium (RTAS)*, Vancouver, Canada, June 1999.

[3] P. Humenn, G. Lewandowski, and D. Zhou. Toward Assured Trusted Time Stamping . In *sixth workshop on distributed objects and components security (DOCsec 2002)*, March 2002.

[4] Intel Corp. *Intel Architecture Software Developer's Manual, Volume 3: System Programming*, 1999.

[5] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[6] Chris Rizos. Principles and Practice of GPS Surveying. available at `http://www.gmat.unsw.edu.au/snap/gps/gps_survey/chap1/132.htm`.