MEINBERG

Computer Time Synchronization Concepts

Martin Burnicki

Meinberg Funkuhren
Bad Pyrmont
Germany

2014-04-29

Table of Contents

1 Introduction....1
2 Who Needs Time Synchronization?....2
3 Local Time Zones and World Time Scales....3
 3.1 Local Time Zones and Civil Time....3
 3.2 Daylight Saving Time....3
 3.3 Historical Greenwich Mean Time (GMT)....3
 3.4 Atomic Time Scales and Atomic Clocks....4
 3.5 Atomic Time (TAI) And Coordinated Universal Time (UTC)....4
 3.6 Leap Seconds....6
4 How Computers Keep Time....7
 4.1 Basic Concepts....7
 4.2 Resolution of the System Time....7
 4.3 Why the Undisciplined Software Clock Drifts....8
 4.4 Disciplining the System Time....9
 4.5 Computer UTC vs. Local Time....9
 4.6 How to Obtain Current Timezone Information....10
 4.7 Why Not Discipline The Computer's Local Time....10
5 How Do I Know Which Time It Is?....11
6 Network Time Transfer Protocols....12
 6.1 The Network Time Protocol (NTP)....12
  6.1.1 NTP Overview....12
  6.1.2 NTP and Local Time....13
  6.1.3 Computer Platforms Supported by NTP....13
  6.1.4 NTP Naming Conventions: ntp or xntp....13
  6.1.5 The NTP Time Synchronization Hierarchy....14
  6.1.6 NTP Built-In Redundancy....14
  6.1.7 The NTP Drift File....15
  6.1.8 NTP Configuration Overview....15
  6.1.9 NTP Configuration with Upstream NTP Servers....16
  6.1.10 NTP's Local Clock Driver....16
  6.1.11 NTP Configuration via DHCP....16
  6.1.12 NTP Access Restrictions....17
  6.1.13 NTP with Meinberg Refclocks on Unix-like Systems....17
   6.1.13.1 Using Meinberg Refclocks with NTP's Parse Driver....17
   6.1.13.2 The Parse Driver's Trust Time Parameter....18
   6.1.13.3 External Meinberg Refclocks under Unix....19
   6.1.13.4 Meinberg PCI and USB devices under Linux....19
   6.1.13.5 Using Meinberg PCI and USB devices with NTP's SHM driver....20
   6.1.13.6 Accuracy Considerations SHM versus Parse Driver....20
  6.1.14 NTP with Meinberg Devices under Windows....20
  6.1.15 NTP Broadcast Mode....21
  6.1.16 NTP Multicast Mode....21
  6.1.17 Using Hardware PPS Signals with NTP....21
  6.1.18 Getting Started with NTP and Troubleshooting....21
   6.1.18.1 Don't Change the System Time While NTP Is Running....22
   6.1.18.2 Time Sources Need to Be Synchronized....22
   6.1.18.3 Check if the NTP server claims to be synchronized....23
   6.1.18.4 Check if the client synchronizes to the server....23
   6.1.18.5 If the client does not synchronize to the server, check if the NTP packet exchange works correctly....24
  6.1.19 Using NTP in a Windows Active Directory Domain....25

6.1.20 Building NTP from Sources....25
6.2 The Precision Time Protocol (PTP/IEEE1588)....26
6.3 RADclock Daemon....26
6.4 The TIME and DAYTIME Protocols....26
6.5 Time Synchronization using NetBIOS/NETBEUI....26
6.6 General Network Time Transfer Aspects....26
6.7 Latencies due to Network Packet Transfers....27
6.8 Network Latency Compensation by the NTP Protocol....28
6.9 Network Latency Compensation by the PTP/IEEE1588 Protocol....29
6.10 Comparison: NTP versus PTP/IEEE1588....30
7 Time Dissemination by Radio Signals....32
7.1 Time Dissemination by Satellites....32
7.1.1 GPS....32
7.1.2 GLONASS....32
7.1.3 Compass / Beidou....32
7.1.4 Galileo....33
7.2 Time Dissemination by Long Wave Transmitters....33
7.2.1 DCF77 in Germany....33
7.2.2 MSF/Rugby in the United Kingdom....33
7.2.3 WWVB in the United States....33
7.2.4 HBG in Switzerland....33
7.2.5 JJY in Japan....34
7.3 Comparison Satellite Systems vs. Long Wave Signals....34
7.3.1 Signal Reception....34
7.3.2 Signal Propagation Delay Compensation....34
8 Hardware Reference Time Sources....35
8.1 Reference Time Signal Type Considerations....35
8.1.1 Satellite Signals....35
8.1.2 Longwave Signals....35
8.1.3 IRIG And Similar Timecode Signals....36
8.1.3.1 Original IRIG Signals....36
8.1.3.2 AFNOR NF S87-500....38
8.1.3.3 IEEE 1344-1995 and IEEE C37.118-2005....38
8.1.3.4 UTC Offset Discrepancies between IEEE1344-1995 and C37.118-2005....39
8.1.3.5 Modulated vs. Unmodulated (DCLS) Timecode Signals....41
8.1.3.6 Selecting An Adequate Timecode Signal....41
8.2 Access Time Considerations....42
8.2.1 PCI Cards....42
8.2.1.1 PCI Express Limitations....42
8.2.1.2 API Calls available for Meinberg PCI Cards....43
8.2.1.3 Circumventing PCI Access Times....43
8.2.2 Native Serial Port....45
8.2.3 USB Devices....45
8.2.4 Fiber Optic....45
8.3 Time Resolution Considerations....45
8.4 Disciplined vs. Undisciplined Oscillator Considerations....46
8.5 Hardware PPS Considerations....47
8.6 Meinberg's Approach to PTP Client PCI Cards....47
9 Distributing Reference Time to Computers....48
10 Time Synchronization Problems with Virtual Machines....50
10.1 General Information....50
10.2 VMWare....50

10.3 XEN..51
10.4 Microsoft Hyper-V..51
11 Potential RTC Problems on Dual Boot Systems..52
12 Time Synchronization Problems Under Windows..53
12.1 Timer Tick Interpolation Problems..53
12.2 Latency Problems Affecting the Windows System Time..53
12.3 Small System Time Adjustments May Be Lost..54
12.4 Polling Intervals With NTP For Windows..55
12.5 Possible Problems in a Windows Active Directory Domain..57

This document is currently work in progress.
Additional chapters with related topics will be added.

1 Introduction

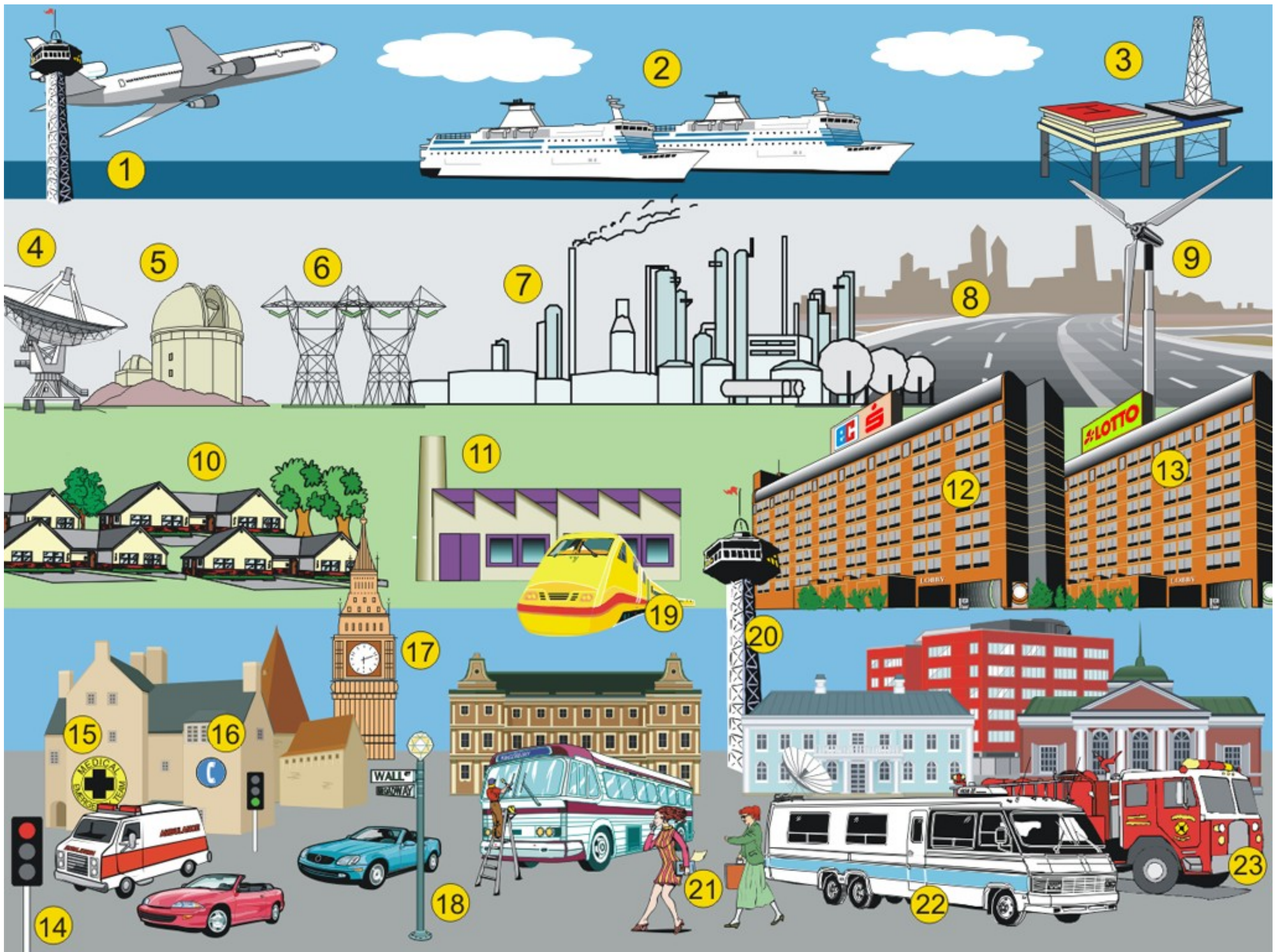
This paper is intended as a primer or reference for people involved with computer time synchronization. It is more a kind of *howto* rather than a scientific article. There are some chapters with basic information to get an overview, but there are also some chapters going into details and trying to make clear why sometimes things may not work as expected by people who are not too familiar with this kind of stuff.

The contents include a short introduction to date and time scales in general, followed by an introduction how computers keep time and how the computer time can be disciplined. This includes basic information on commonly used techniques and protocols as well as advantages and limitations of individual techniques.

Last but not least this paper explains how hardware time sources can be used to yield a better accuracy of the computer system time, and how the achievable accuracy depends on certain characteristics of a hardware time source.

2 Who Needs Time Synchronization?

Time synchronization is a matter of course everywhere in our daily life.
Here are some examples:



1. **Air Traffic Control**
2. **Research Vessels**
3. **Oil Production**
4. **Satellite Communication**
5. **Observatories**
6. **Power Substations**
7. **Power Plants**
8. **Toll Charging Systems**
9. **Wind Energy Plants**
10. **Public Infrastructure**
11. **Production Flow**
12. **Banks, Cash Terminals, Stock Exchange, Data Centers**
13. **Lottery**
14. **Traffic Management**
15. **Operation Coordination**
16. **Event Management**
17. **Wall Clocks**
18. **Lighting Control**
19. **Railway Time Table**
20. **Radio Broadcasting**
21. **Mobile Communication, Call Data Records**
22. **Outside Broadcast Van**
23. **Emergency**

3 Local Time Zones and World Time Scales

3.1 Local Time Zones and Civil Time

Everywhere on the planet human beings expect the time of day to be 12 o'clock noon when the sun reaches the maximum altitude, so several time zones have been defined around the globe.

Basically there are 24 time zones, each of which differs from the next zone by 1 hour and covers a section of the globe which spans about 15 degrees of the geographic longitude. However, since the legal local time is determined by each country's own government, the borders of the time zone often follow the borders of individual countries rather than exactly one of the globe's meridians. There are even a few small regions which define their own time zone differing from the next time zone by 15, 30, or 45 minutes rather than 1 hour.

Today the legal standard time in each time zone is derived from a common world time, which has formerly been called GMT but been replaced by a new definition called UTC. See the next chapters for details.

Some countries like the United States or Russia span across several time zones, so the inhabitants of these countries are usually aware there are several time zones and take this into account in their daily life. On the other hand, there are countries which are completely located inside a single time zone, so the inhabitants of these countries are often not aware there are several time zones until they get in touch with computer time synchronization, or international business relations.

3.2 Daylight Saving Time

Many countries also switch their legal local time to **Daylight Saving Time** (DST, also called Summer Time) during parts of the year. Whether this happens, and at which date and time DST starts or ends is determined by the legislation of each individual country. Some countries follow some general rules for DST, e.g. DST starts at a given time on the last Sunday in March and ends at a given time on the last Sunday in October year by year, whereas other countries follow irregular rules, e.g. Israel, where the beginning of DST depends on the date of a religious holiday in a given year, or Morocco, which temporarily switches back from DST to standard time during Ramadan.

Most countries put the clocks forward by 1 hour during the period where DST is in effect. As a consequence, the length of a day is 23 hours rather than 24 hours on the day when DST begins, since one hour is skipped. Accordingly, the length of a day is 25 hours rather than 24 hours on the day when DST ends since the clocks are put backward and thus 1 hour is passed twice. This needs to be taken into account e.g. if measurements are made in cyclic intervals which start at the same time every day.

3.3 Historical Greenwich Mean Time (GMT)

For historical reasons the zero meridian of the planet Earth has been defined to go through the Royal Greenwich Observatory near London, U.K., and the time measured at that observatory has been used as the base time scale for all other time zones East and West of Greenwich.

This original GMT time scale defined and used mainly by astronomers, so it was based on measurements of the true earth rotation, and a day started a 12:00 noon. In order to adapt to civil habits, the beginning of a new GMT day was changed from 12:00 noon to 12:00 midnight in the

beginning of 1925. However, this caused lots of confusion, so in 1928 the new GMT time scale was renamed to Universal Time (UT). UT is still bound to the earth rotation, and variants of UT (UT0, UT1, and UT2) are still used today by astronomers.

Unfortunately the earth rotation varies over time: it continuously slows down over the long term, mostly due to tide effects, and it can temporarily speed up or slow down due to volcanism, earth quakes, big floods, etc., so obviously a time scale which is associated to the earth rotation also varies over time, and thus the length of a second varies over time. This became unacceptable for technical purposes, and thus a new time scale needed to be invented where the length of a second does not vary. This new time scale was called Coordinated Universal Time (UTC).

Anyway, the old name GMT is still often used in places where UTC would be more appropriate and correct. Especially, GMT is often intermixed with the British civil time which is the same as UTC unless daylight saving is in effect.

3.4 Atomic Time Scales and Atomic Clocks

As mentioned in the previous chapter the length of a second derived from GMT or UT is not constant and thus not very suitable for technical applications. So in 1967 the International System of Units (SI) defined the length of a reference second (SI second) based on a natural constant. It turned out the resonance frequency between selected energy levels of specific Cesium atoms is pretty stable, so the length of a second was defined as a certain number of periods of this frequency.

Atomic clocks are devices which let some of those Cesium atoms oscillate between those specified energy levels and count the number of oscillations in order to determine the length of a second. So unlike the name *Atomic Clock* suggests, these devices have nothing to do with nuclear power plants or nuclear bombs, nor do they emit any radioactivity.

More detailed information on Cesium clocks can be found at
http://tycho.usno.navy.mil/cesium.html

In the last couple of years hydrogen masers have been developed which significantly increase the accuracy from about 10^{-13} to 10^{-15}. This means the time derived from this frequency was only off by 1 second after 10^{15} seconds of operation, i.e. after more than 31 million years!

Such Cesium clocks, hydrogen masers, and similar devices are nowadays installed in metrology institutes and observatories all over the world, building the basis of the modern atomic time scales.

3.5 Atomic Time (TAI) And Coordinated Universal Time (UTC)

In 1972 a new global time scale was defined which is called *Coordinated Universal Time* (UTC, Universal Time, Coordinated). The UTC time scale is based on the SI seconds generated by atomic clocks.

Most industrial countries have their national metrology institutes or observatories where they have one or more atomic clocks installed. The best of those clocks contribute to a common Atomic Time (TAI, Temps Atomique International), weighted depending on the accuracy and stability of each individual clock.

The atomic time TAI is a linear time scale which is used as a base for the common world time,UTC,

which differs from TAI by an integral number of leap seconds which have been inserted over the last decades. See the next chapter.

As the name *Coordinated Universal Time* suggests, the measurements and contributions to TAI (and thus UTC) are coordinated by the *Bureau International de Poids et Mesures* (BIPM): http://www.bipm.org/

All affiliated atomic clocks are compared to each other and steered such that they stay as close as possible to the common UTC time, and in fact most of the clocks differ from the global UTC time by a few nanoseconds only.

Some popular institutes which operate atomic clocks contributing to UTC are

- The German *Physikalisch-Technische Bundesanstalt* (PTB, the German national metrology institute)
  http://www.ptb.de/

- The U.S. *Naval Observatory* (USNO)
  http://www.usno.navy.mil/

- The U.S. *National Institute of Standards and Technology* (NIST)
  http://www.nist.gov/

- The National Physical Laboratory (NPL) in the United Kingdom
  http://www.npl.co.uk/

3.6 Leap Seconds

Since TAI (and thus UTC) seconds have a constant length whereas the earth rotation may slightly vary over time, leap seconds have been applied to the UTC time scale whenever appropriate to keep UTC consistent with the actual earth rotation speed.

In theory leap seconds can be inserted or deleted, but in practice leap seconds have only been inserted over the last decades. The evolution of the earth rotation speed since the introduction of UTC, and the leap second which have been inserted since to compensate this can be seen on this graph:

The graph above has been copied from this web page:
http://hpiers.obspm.fr/eop-pc/earthor/utc/leapsecond.html

It shows descriptively how the earth rotation has changed over time, and how leap second have been inserted to adjust the UTC time scale to the earth rotation.

Historical Information on earth rotation are presented here:
http://www.ucolick.org/~sla/leapsecs/dutc.html#atomic.png

Usually leap seconds are only scheduled for UTC midnight at the last day of June or December, so that they occur at the same moment in time world-wide. However, as a consequence this happens at different local times depending on the time zone you are in.

Whenever a leap second is added or deleted the last minute of the day has 61 or 59 seconds, and thus the minute, the hour, and the whole day is 1 second longer or shorter than usual, which must be taken into account by billing systems, etc.

Actually there is an ongoing discussion whether it makes sense to redefine the UTC time scales without leap seconds. See:
http://www.futureofutc.org

Leap seconds must not be confused with **leap years**, where a whole day is inserted to account for variations of the length of a year instead of the length of a day.

While leap seconds are used to adopt the time derived from technical TAI seconds to the interval the globe requires for a full rotation around its own axis, i.e., a day, leap years are introduced to adopt calendar dates to the interval the Earth requires to circle around the sun, i.e., a year.

4 How Computers Keep Time

4.1 Basic Concepts

Modern computers usually provide a battery buffered real time clock (RTC) chip on the mainboard which keeps the current date and time when the computer is powered off. This RTC chip is often also called *CMOS clock* or *hardware clock*. Since this chip operates from a battery when the computer is powered off the chip is usually driven by a crystal at only a very low frequency in order to minimize consumption of battery power.

When the computer is powered on the operating system starts a so-called software clock or kernel clock which then continues to keep the current time. The software clock works using one of the timer/counter circuits provided by the chipset on the mainboard, which is configured to generate so-called timer tick interrupts in cyclic intervals. Depending on the operating system type and version, the tick interrupt can either be generated by the RTC chip, or by a different timer circuit available on the mainboard.

Whenever a timer tick interrupt occurs a certain tick adjustment value is added to a linear time which represents e.g. the number of seconds and fractions of a second since a given epoch. That linear time can then be converted to a human readable calendar date and time whenever required. This so-called software clock is used by most modern operating systems.

Special problems may arise in virtual machines which rely on virtual computer hardware since the timer tick interrupt is only emulated and thus timer tick intervals may strongly vary in length. See chapter *“Time Synchronization Problems with Virtual Machines”* for details.

Anyway, the time kept by the software clock can drift apart from the time kept by the RTC as long as the computer is up and running. This is why some operating systems write the software time back to the RTC chip when the operating system shuts down.

Linux kernels may even have been configured to write the software time back to the RTC in cyclic intervals while the system is running. The interval is usually 11 minutes, and thus this feature is often referred to as 11-minute-mode, which is used to make sure the RTC is updated even if the system is not shut down properly, e.g. if used in embedded devices.

For considerations of potential problems on Multi Boot systems refer to chapter *“Potential RTC Problems on Dual Boot Systems”*.

For the Linux operating system more detailed information can be found at:
The Clock Mini-HOWTO: How Linux Keeps Track of Time [1]
http://tldp.org/HOWTO/Clock-2.html

4.2 Resolution of the System Time

Most **Unix-like systems** (including Linux, *BSD, Solaris, etc.) provide the system time at least with microsecond or even nanosecond resolution. This means applications can read system timestamps with that resolution, and thus the timestamps from subsequent reads can be distinguished.

The **Windows system time** is only incremented in steps of a timer tick interval and thus provides only very limited resolution. Up to Windows XP and Windows Server 2003 the timer tick interval

is about 16 milliseconds only. This means even though the available API calls support a resolution of 100 nanosecond steps, an application which reads the system time continuously receives exactly the same time stamp during a whole 16 ms timer tick interval, and after the next timer tick the returned time steps by the amount of the timer tick interval.

Under Windows Vista and newer the timer tick interval changes down to about 1 millisecond if an application is running which sets the Windows multimedia timer to highest resolution.

With Windows 8 a new software interface (API function) has finally been introduces which allows applications to read the system time with 0.1 microsecond resolution. However, only applications which explicitely use this API call can benefit from this higher resolution.

Some time synchronization software including the reference implementation of NTP and the Meinberg driver software package for Windows try to increase the resolution of the legacy Windows system time on Windows versions which don't support the new API call with the help of an additional timer used to interpolate the time between two timer tick interrupts. This helps to discipline the system time more smoothly than without interpolation. However, the interpolated system time is not available to other applications using the Windows standard API calls to read the system time.

An alternate way to provide Windows applications with high-accuracy, high-resolution time stamps is having the applications read the time stamps directly from a PCI card provides high accuracy, high resolution time.

If the system time is provided with high *resolution* then this does not necessarily imply the system time also provides microsecond or nanosecond *accuracy*. However, obviously the reverse conclusion is true: If the system time only provides limited resolution then the possible accuracy is also limited. This is the main reason why usually the system time of a Unix machine can be disciplined more accurately than the system time of a Windows machine.

4.3 Why the Undisciplined Software Clock Drifts

The tick adjustment value used by the software clock has been determined such that the counted system time increases exactly if the crystal which drives the counter runs at its nominal frequency. In most cases, however, that crystal does not run exactly at its nominal frequency. Instead, each crystal has its own “native” frequency which is more or less off its nominal frequency. So the amount of time which is added to the system time at each timer tick is a little bit too large or too small, and thus causes the system time to drift away more or less slowly over time.

Since the “native” frequency is unique to the individual crystal, even several mainboards of the same type have their own “native” clock drift. Some time synchronization programs determine the system clock drift during operation, save the determined drift across a reboot, and use the saved value as startup value after a reboot, see e.g. the drift file used by the NTP reference implementation.

Beside the native frequency offset the frequency of a crystal also varies with the ambient temperature, so in addition the clock drift varies with the ambient temperature. The magnitude of this frequency variation depends on the quality of the crystal.

4.4 Disciplining the System Time

The easiest way to discipline the computer system time is to provide a reference time source which is more accurate, and then set the system time in periodic intervals. The disadvantage of this method is, however, that the system time still drifts during each interval, and after the next interval a time step occurs in the range of the time offset accumulated during the last interval.

In order to avoid this problem modern time synchronization applications like implementations of the Network Time Protocol (NTP), the Precision Time Protocol (PTP), and also the time adjust service from the Meinberg driver package for Windows don't just set the system time periodically. Instead, they also try to determine and compensate the system clock drift to discipline the system time smoothly. Modern operating systems usually provide a programming interface which can be used by applications to control the system clock drift. How well this works depends strongly on the type and version of the operating system.

NTP is mostly used to synchronize the time across a network. The reference implementation of NTP which is freely available on the internet can either use an upstream NTP server or a hardware reference clock as a reference time source to discipline the system time. Under Windows the Meinberg time adjustment service should be used to discipline the system time, and the NTP service can be installed to make the disciplined time available on the network.

4.5 Computer UTC vs. Local Time

Modern operating systems keep their system time internally in UTC, and convert the time displayed to the user to local time depending on the time zone settings, which also include settings whether daylight saving time (DST) is observed, and at which date and time DST begins and ends.

Under Windows there is usually only a single global time zone setting. However, on Unix-like systems every user can configure his own preferred time zone, and even single processes can be run with individual time zone settings. All those individual local times are derived from the common system UTC time, so it makes most sense to discipline the system UTC time only.

The operating system also makes sure that always the correct local time is returned, e.g. one millisecond before DST begins the local standard time is returned, and one millisecond after DST switchover the correct daylight saving time is returned. This would not be possible if some time synchronization software which is run e.g. once per second tried to do this job. See also chapter “*Why Not Discipline The Computer's Local Time*”

In most cases applications which present the time to the user read the current local time from the operating system. However, there are also applications which use the current UTC time for timestamping. Using UTC time makes it easier to relate events to each other which have been recorded in different time zones, and it does not suffer from time steps from DST switchovers.

This is why modern data base applications use UTC time to timestamp transactions. Those data base applications can be totally messed up if for example the system UTC time is stepped back so that earlier time stamps are assigned to events which happened later.

So even though it is not obvious at the first glance, a correct, and correctly increasing, UTC time is most important for modern operating systems and applications. The local time can always be tweaked to match the users preferences by simply changing the time zone configuration according the requirements.

4.6 How to Obtain Current Timezone Information

The decision if and when a local time zone should switch to daylight saving is determined by the legislation of each individual country. If the rules are changed by the legislation then the operating systems needs to be updated to account for the modified DST rules. See also chapter *"Local Time Zones and Civil Time"*, and chapter *"NTP and Local Time"*.

Unix-like systems usually come with the so-called Olson timezone database
http://en.wikipedia.org/wiki/Tz_database
to specify local time zones, while Windows keeps the available timezone rules in a set of registry entries. If a timezone specification is changed for a specific country then usually new packages or service packs are made available by the OS vendors/distributors which account for the changed DST rules.

4.7 Why Not Discipline The Computer's Local Time

Good time synchronization software does not care about the computer's local time. Instead it disciplines the computer's UTC time, and the local time which is displayed depends only on the configured time zone settings, based on the rules which come with the OS.

This is the way the Network Time Protocol (NTP) and Precision Time Protocol (PTP/IEEE1588) work, and this is also how the time adjustment service for Windows from the Meinberg driver package for Windows works.

If the computer's local time would be disciplined rather than its UTC time then this might result in severe problems which may not be obvious at the first glance.

Time synchronization software usually compares the system time to some reference time in specified intervals only. If e.g. the time interval is 10 seconds, and a DST switchover would have to occur one millisecond after the last check, then the real DST switchover would occur 10 seconds too late, and thus the wrong local time would be returned for 10 seconds, or in general for the length of the time interval between two checks.

Even worse things will happen if a given time zone has been specified, or time zone "GMT+0" has been specified in the local time configuration, but the user expects to see his true local time rather than "GMT+0" anyway. In both cases the system's UTC time is wrong, or will be stepped back and forth around the time a DST switchover occurs, if the system's local time rather than its UTC time was disciplined.

Even though this might not be visible on the user interface, this may have disastrous consequences for applications which rely on proper UTC time, like database applications. See also the previous chapter.

So it is most important to discipline the computer's UTC time, and this is why especially the NTP network protocol has been designed to deal **only** with UTC time. Forcing NTP servers to send local time rather than UTC results in messed up time on the involved systems, and are considered a **violation** of the NTP protocol specification. See also chapter *"NTP and Local Time"*.

5 How Do I Know Which Time It Is?

Basically there are different ways to get to know what time it is. Each way has specific advantages and maybe disadvantages:

- There's a funny video on Youtube showing how to *do it the Italian way*:
  http://www.youtube.com/watch?v=u9OOjr7odPY ;-)

- *Look at a clock which I have in sight*
  I can look at the clock whenever I want, as often as I want, and know the time immediately
  → Read fast time stamps from a PCI card

- *Listen when the church clock bell sounds*
  Whenever the bell sounds I know it's a full hour
  After I have counted the strokes I know what time it was at the beginning
  I don't know exactly what time it is in between.
  → Wait until serial time string sent automatically, e.g. on second changeover

- *Ask someone else who knows the time*
  The other person looks at its wrist watch and replies to my query earlier or later
  → Network time protocols (NTP, PTP)
  → Querying slow devices (USB)

For computer time synchronization there are similar aspects to be kept in mind:

- *Where do I get the time from? At which accuracy?*
  → Radio clock connected to the PC
  → Time server on the network

- *Which ways exist to get the time?*
  → PCI card: Can get the current time always, immediately
  → Serial: Wait for time string. When sent? Transmission delay?
  → Network: Send query, wait for reply, compensate network delays

- *Resolution of the local system time?*
  → Depends on operating system
  → Windows: 16 ms (XP) or 1 ms (Vista and newer)
  → Unix/Linux/Windows 8: 1 µs or even better

- *Time synchronization software?*
  → Which resolution is supported?
  → Is transmission delay compensated?
  → How is system time adjusted? Set periodically? Smoothly?

These questions will be discussed in the next chapters.

6 Network Time Transfer Protocols

The first public network time transfer protocols have been called **time** and **daytime** protocol (see chapter 6.4) which were published back in 1983. However, those protocols provided only a limited accuracy, and did not try to compensate the network delays. So later in the 1980's the **Network Time Protocol (NTP)** was invented which significantly improved the possible time synchronization accuracy. The reference implementation of NTP was designed to run under Unix-like systems and mainframes.

Under Windows the original way to synchronize times was using dedicated packets of the NETBIOS protocol. The NETBIOS protocol was later extended and renamed to NETBEUI protocol. However, this kind of time synchronization also had some limitations, so current Windows versions also use an NTP (or Simple NTP) implementation provided by the **Windows Time service (w32time)** by default.

The Precision Time Protocol (PTP) has been introduced some years ago in order to improve the possible accuracy beyond the level of accuracy provided by NTP. However, this requires special hardware support to yield the highest level of accuracy.

All the protocols are usually handled by a daemon or service process which runs in the background of the operating system. The next chapters describe some details and characteristics of these protocols.

Nowadays many workstations are shipped with a pre-installed NTP client, so Meinberg also offers various plug-and-play NTP and PTP time servers called LANTIME, with different reference clock options, e.g. built-in GPS or DCF77 PZF receivers. The devices also have a network interface and power supply included, are assembled in a standalone case and ready to operate and provide clients with accurate time.

6.1 The Network Time Protocol (NTP)

6.1.1 NTP Overview

The NTP protocol has been invented in the 1980's by Dave L. Mills at the University of Delaware. The ambition was to achieve the highest possible time synchronization accuracy for computers across the network. The protocol and related algorithms have been specified in several RFCs.

The public domain software package called **NTP** is the reference implementation of this protocol. Since the original implementation NTP has been enhanced and is now widely used around the world. The protocol supports an accuracy of time down to nanoseconds. However, the real accuracy which can be achieved also depends on the operating system and the network performance.

The current **NTP v4** protocol version has being standardized by the IETF, and the basic format of the network packets is compatible with earlier NTP versions, so current NTP implementations can be used together with older versions, unless specific NTP v4 features are being used. In addition to NTP there's also a simplified version called **SNTP** (**S**imple **N**etwork **T**ime **P**rotocol) which uses the same TCP/IP UDP packet structure like NTP but due to the simpler algorithms, it usually provides only reduced accuracy and is thus mostly used for simple clients. The NTP package contains a background program (*daemon* or *service*) which synchronizes the computer's system time to one or more external reference time sources which can be either other devices on the network, or a

hardware reference time source connected to the computer.

Additionally, the NTP distribution contains programs which can be used to control or monitor the time synchronization status, and a complete set of documentation in HTML format.

6.1.2 NTP and Local Time

The decision if and when a local time zone should switch to daylight saving is determined by the legislation of each individual country. If the rules are changed by the legislation then the operating systems needs to be updated to account for the modified DST rules. See also chapter *“Local Time Zones and Civil Time”*, and chapter *“Computer Local Time versus Computer UTC Time”*.

The NTP network protocol has been designed to deal **only** with UTC time. There are no provisions to let NTP handle local time offsets, times for DST switchover, etc. Using something different than UTC in the NTP protocol is clearly a **violation** of the NTP protocol specification. For ways how to upgrade the local time specifications see also chapter *“How to Obtain Current Timezone Information”*.

Also, forcing NTP servers to send local time rather than UTC results in messed up time on the involved systems. NTP programs usually expect UTC time, and they expect the time to increase monotonically. Those programs implement filters to measure and compensate the system clock drift. If the time suddenly steps because the NTP server starts to send DST instead of standard time then the sudden time step would only be accepted after a number of polling cycles, i.e. after several minutes. Then the NTP software had to discard all earlier filter values, step the system time, and restart from scratch.

Beside the requirement to be able to use the NTP protocol safely across the world's time zones, the chapter “*Why Not Discipline The Computer's Local Time*” provides more reasons why NTP uses only UTC time.

6.1.3 Computer Platforms Supported by NTP

NTP's native operating system is **UNIX**. Today, however, NTP runs under many **UNIX-like systems**, and NTP v4 has also been ported to Windows. It can be used under **Windows NT**, **Windows 2000**, and newer Windows versions up to **Windows Vista**, **Windows 7 / Windows 8** and **Windows Server 2008 / 2012**.

The standard NTP distribution can **not** be run under**Windows 3.x and Windows 9x/ME** because there are some kernel features missing which are required for precision time keeping. For Windows 9x/ME and other platforms which are not supported directly by the NTP package there are some NTP or SNTP programs available on the internet. An overview of available programs can be found on the NTP support home page.

6.1.4 NTP Naming Conventions: *ntp* or *xntp*

Each NTP source distribution contains the NTP daemon itself, plus some utility programs. Earlier versions of the NTP distribution and some of the programs included in the package had names starting with ***xntp*** (e.g. xntpd) while other utilities in the same package had names starting with ***ntp*** (e.g. ntpq).

Beginning with NTP version 4, the naming conventions were changed to be more straightforward, so now the name of the NTP distribution itself and the names of all the programs included start with ***ntp*** (e.g. ntpd, ntpq).

Some Unix-like operating systems use **a script to start the NTP daemon** at system start-up. Sometimes the script still has a name starting with *xntp* even though the real name of the daemon started by the script is *ntpd*. This is the case, for example, for some versions of SuSE/openSUSE Linux. Other commonly used names *for the startup script* are *ntp* or *ntpd*.

6.1.5 The NTP Time Synchronization Hierarchy

The NTP daemon can not only adjust its own computer's system time. Additionally, each daemon can be a client or peer for other NTP servers, and act as a server for other NTP daemons at the same time:

- As **client** it queries the reference time from one or more servers.
- As **server** it makes its own time available as reference time for other clients.
- As **peer** it compares its system time to other peers until all the peers finally agree about the "true" time to synchchronize to.

These features can be used to set up a hierarchical time synchronization structure. The hierarchical levels of the time synchronization structure are called ***stratum*** levels. A smaller *stratum* number means a higher level in the hierarchy structure. On top of the hierarchy there is the daemon which has the most accurate time and therefore the smallest *stratum* number.

By default, a daemon's *stratum* level is always one level below the level of its reference time source. The top level daemon often uses a radio clock as reference time source. By default, radio clocks have a *stratum* number of 0, so a daemon who uses that radio clock as reference time will be seen as a ***stratum 1 time server***, which has the highest priority level in the NTP hierarchy. In large networks it is a good practice to install one or more stratum 1 time servers which make a reference time available to several server computers in each department. Thus the servers in the departments become stratum 2 time servers which can be used as reference time source for workstations and other network devices of the department.

Unlike in telecom applications where the word *stratum* is used e.g. to classify oscillators according to their absolute accuracy and stability, the term *stratum* in the NTP context does not indicate a certain class of accuracy, it's just an indicator of the hierarchy level.

6.1.6 NTP Built-In Redundancy

Each NTP daemon can be configured to use several independent reference time sources. Each reference time source is queried (polled) periodically in certain intervals, and the time sources are then classified into groups of time sources which agree about the same time. This allows a group of "good" time sources (*truechimers* in NTP terminology) to overvote a smaller group of "bad" time sources (so called *falsetickers*). The so called *system peer* is then selected from the group of truechimers.

If the time source currently selected as system peer becomes unavailable then a new system peer is determined based on this selection algorithm. The *stratum* level under which a daemon is visible on the network corresponds to the current system peer's stratum level, plus 1.

For details on the selection algorithm see:
David L. Mills, Mitigation Rules and the `prefer` Keyword
http://www.eecis.udel.edu/~mills/ntp/html/prefer.html

For details on the best number of time sources to be used see:
The NTP Support Web, Selecting Offsite NTP Server
http://support.ntp.org/bin/view/Support/SelectingOffsiteNTPServers#Section_5.3.3.

6.1.7 The NTP Drift File

The reference implementation of the NTP daemon can be configured to use a driftfile to save the computed system clock drift compensation value across reboots.

6.1.8 NTP Configuration Overview

The NTP daemon reads its configuration from a file named **ntp.conf**. On UNIX-like systems, this file is located in the **/etc** directory by default.

On Windows platforms, if a recent NTP version has been installed using the GUI installer from the Meinberg NTP download page, the ntp.conf file is located in an etc\ directory below the NTP program directory, e.g. in *c:\Program Files\NTP\etc*.

Earlier versions of NTP for Windows assumed the ntp.conf file to be located in either **%systemroot %** or **%systemroot%\system32\drivers\etc**, where *%systemroot%* corresponds to *c:\winnt* or *c:\windows* in standard installations.

In most installations the **ntp.conf** file contains at least one or more lines starting with the keyword ***server***. Each of those lines specifies one reference time source which can be either another computer on the network, or a hardware reference time source connected to or installed inside the local computer.

Reference time sources are specified using IP addresses, or host names which can be resolved by a DNS name service. If an IP address represents a real node on the network then the NTP daemon assumes another NTP daemon running on a computer with that address. Additionally, NTP uses some pseudo IP addresses to specify special reference time sources.

For example, NTP uses a **pseudo IP address 127.127.8.*n*** to access a Meinberg radio clock installed at the local computer via the parse driver. To access its own system clock, also called the ***local clock***, NTP uses the **pseudo IP address 127.127.1.0**. This IP address must not be mixed up with 127.0.0.1, which is the IP of the *localhost*, i.e. the computer's loopback network interface.

Attention: Some older versions of NTP have problems with DNS name resolution under Windows if support for MD5 authentication has been compiled in. In this case all TCP/IP addresses in the **ntp.conf** file must be entered in dotted decimal notation (e.g. *172.16.1.1*) rather than DNS name like *host.domain.com*.

6.1.9 NTP Configuration with Upstream NTP Servers

NTP configuration for computers without hardware reference clock is quite simple. For each computer which is to be used as reference time source, a line must be added to the file **ntp.conf**. Additionally, the computer's local clock can be configured to be used by the NTP service if none of the other time servers on the network can be reached. Since the time servers on the network shall be preferred, the local clock's *stratum* should be forced to a high number:

```
server 127.127.1.0            # local clock
fudge 127.127.1.0 stratum 12  # not disciplined

server ntp_server_1 iburst
server ntp_server_2 iburst
server ...
```

where *ntp_server_1*, *ntp_server_2*, etc. must be the real host names or IP addresses of existing NTP servers.

The *iburst* keyword which is supported by recent versions of the NTP implementation should be added to speed up initial synchronization. See chapter *"Getting Started with NTP"* for details.

6.1.10 NTP's Local Clock Driver

Additionally, there can be an entry for the local clock which can be used as a fallback resource if no other time source is available. Since the local clock is not very accurate, it should be fudged to a low stratum:

```
server 127.127.1.0            # local clock
fudge 127.127.1.0 stratum 12
```

6.1.11 NTP Configuration via DHCP

In order to be able to configure a large number of NTP client machines in an easy way some DHCP client programs (e.g. the ISC's dhcpcd on Unix-like systems) can query the IP addresses of one or more upstream NTP servers via the DHCP protocol and possibly overwrite an existing ntp.conf file with a new configuration.

This feature can significantly simplify administrative efforts in large networks, since the administrator only has to specify one or more NTP server adresses on the DHCP server.

On the other hand, this can strongly confuse a user who tries to set up a machine with an individual NTP configuration which is then occasionally overwritten by the settings received via DHCP.

NTP server information is only sent by the DHCP server if a DHCP client requests this type of information, and the DHCP client program needs to evaluate this information and update the NTP configuration accordingly, so this is outside the scope of the NTP programs, and the way to enable or disable this feature for the DHCP client depends on the operating system type and version, not on NTP. Also, this method is not appropriate to configure an NTP daemon which shall query the time from a hardware reference time source instead of an upstream NTP server.

6.1.12 NTP Access Restrictions

See:
http://support.ntp.org/bin/view/Support/AccessRestrictions

Please note the exact behaviour of given access restriction configuration parameters may depend on the exact version of the NTP daemon.

6.1.13 NTP with Meinberg Refclocks on Unix-like Systems

The NTP package contains several drivers which can be used to let the NTP daemon read the reference time from various hardware reference time sources (refclocks). On Unix-like systems the *parse driver* can be used to read time strings in various formats from external refclocks connected via one of the computer's serial ports.

If the NTP daemon shall use a PCI card or USB device as reference time source then usually an additional driver package is required. Meinberg provides a driver package which can be installed to use Meinberg PCI cards or USB devices with the NTP daemon under Linux. The driver package has originally been used with the parse driver, but there's also a development version of the driver available which can alternatively be used with the *shared memory (SHM) driver* provided by the NTP package.

The following chapters explain how to set up these configurations, and discuss the pros and cons of the different methods.

6.1.13.1 Using Meinberg Refclocks with NTP's Parse Driver

On UNIX-like systems the *parse driver* (type 8) is used to read the time from reference clocks manufactured by Meinberg and connected via a serial port. The parse driver is part of the NTP package, but must explicitely be activated when the NTP package is compiled. This is usually done by the vendor or distributor of the operating system, so most Unix-like operating systems come with a precompiled NTP package where the parse driver has been compiled in. However, e.g. some NTP packages shipped with older Solaris versions have been built without the parse driver.

If the NTP package installed on a system has been compiled without the parse driver but is to be used with a hardware reference time source then the NTP package needs to be recompiled with the appropriate option enabled. See chapter *"Getting started with NTP and Troubleshooting"* for details how to find out if the parse driver has been enabled with a running NTP daemon.

The configuration steps described below for the parse driver must be done by a user with sufficient rights on the system, e.g. user root. The parse driver accesses radio clocks via symbolic links **/dev/refclock-*n***, where ***n*** is an index number in the range 0 through 3 since the parse driver can handle up to four reference clocks in parallel.

Each symbolic link must point to a physical device representing an existing hardware reference time source. In most cases the physical device is a serial port at which a radio clock has been connected externally.

Each of the reference clocks must also be specified in the **ntp.conf** file using a ***server*** line with the pseudo IP address **127.127.8.*n***, where ***n*** must correspond to the index numbers used with the symbolic device names **/dev/refclock-*n*** mentioned above.

The pseudo IP address must be followed by a **mode *m*** parameter which specifies the type of radio clock represented by the device. The table below lists mode values which can be used with Meinberg devices connected via a serial port:

Mode Number	Device / String Format / Oscillator Quality	Trust Time
mode 0	Meinberg PZF clock with TCXO	12 hours
mode 1	Meinberg PZF clock with OCXO	4 days
mode 2	Meinberg Standard Time String with 9600, 7E2	30 minutes
mode 7	Meinberg GPS with OCXO, 19200, 8N1	4 days

Originally the mode number was also used to specify the trust time for the hardware clock, depending on the quality of the oscillator provided by the hardware clock. However, with current versions of the NTP implementation the trust time can be specified in the ntp.conf file. See below.

So nowadays in most cases **mode 2** is used for all Meinberg PCI and USB devices, and serial devices which send the Meinberg Standard time string at 9600, 7E2, and **mode 7** is used for all devices which send the Meinberg Standard time string or the so-called Uni Erlangen time string at 19200, 8N1.

For example, if a single radio clock is connected to the serial port /dev/ttyS0 then a symbolic link for the clock must be set up using the command

```
ln -s /dev/ttyS0 /dev/refclock-0
```

In the next step the file **ntp.conf** must be edited to configure the NTP daemon and tell it which reference clocks to use. The file should include a *server* line for the refclock-0 device created above. If the radio clock sends the Meinberg standard time string at 9600 baud and framing 7E2 then, as can be seen from the table above, the mode for refclock-0 must be set to 2. Also, if plug-in card is used under Linux then mode 2 must always be used:

```
server 127.127.8.0 mode 2     # standard time string with 9600, 7E2
```

6.1.13.2 The Parse Driver's Trust Time Parameter

If the device is synchronized (mbgstatus Status info: Clock is synchronized) then the time from the card is accepted by ntpd.

If ntpd finds the card is **not synchronized** then the behaviour depends on an additional condition:

- If the card reports "not synchronized" when ntpd is starting then ntpd refuses to accept the time from the card until the card's status changes to "synchronized".

- If ntpd has been running, seen the card synchronized before, but then the card loses sync, then the daemon keeps on accepting the card for a certain trust time. When the trust time20 expires and the card is still not synchronized again then the NTP daemon stops accepting the time from the card.

- Please note also that there may be security tools like AppArmor or SELinux installed which prevent ntpd from accessing the device, unless those tools are explicitely configured to grant access to the card to ntpd.

6.1.13.3 External Meinberg Refclocks under Unix

6.1.13.4 Meinberg PCI and USB devices under Linux

A Linux driver for Meinberg PC plug-in cards and USB devices is available on the Meinberg software download page. This driver neither supports devices connected via a serial port, nor is it required to use devices connected via serial ports with the NTP daemon.

The Meinberg driver package for Linux enables the NTP daemon to use Meinberg PCI cards or USB devices as a reference time source used to synchronize the Linux system time. Additionally the package contains programs which allow monitoring of the device status, e.g. whether the device is synchronized to its incoming time signal, or not, and modifying specific device configuration parameters, e.g. the IRIG frame format for IRIG receivers.

The Meinberg driver package for Linux contains programs which can be used to monitor the device status, e.g. whether the device is synchronized to its incoming time signal, or not, and to modify specific device configuration parameters, e.g. the IRIG frame format of IRIG receivers. However, it also lets the NTP daemon use Meinberg PCI cards or USB devices as a reference time source used to synchronize the Linux system time.

The original approach is to let the kernel module from the driver package emulate a serial interface as expected by NTP's parse driver, and thus allow the parse driver to read the reference time from PCI or USB devices rather than via a real serial port. However, there are some drawbacks with this method, see below, so current development versions of the driver package support a different approach using the shared memor driver (SHM, type 8) supported by the NTP daemon.

If a recent version of the driver package is used then up to 4 devices can be used with NTP, and the required /dev/refclock-* links are by default created automatically by the Linux *udev* system. In older versions of the driver package (before v3.0.0) the link had to be created manually, and the device node to be used for the refclock link was named /dev/mbgntp. If in doubt please see the README file which comes with the driver package.

Be sure an entry for refclock-<n> is included in the ntp.conf file which is usually located in the /etc directory. The lines should look like:

```
server 127.127.8.<n> mode 2              # mode 2 for all Meinberg PCI cards
fudge 127.127.8.<n> time1 0.0            # no systematic delay
fudge 127.127.8.<n> refid GPSi           # informational, dep. on card type
fudge 127.127.8.<n> flag1 1 time2 7200  # optionally, set trust time
```

with <n> matching the index number used for the symbolic link and ***mode 2*** telling the NTP daemon to use the data format of the Meinberg standard time string.

The ***fudge*** lines setup some NTP parameters for this clock. The time1 parameter is a build-in compensation of a constant time delay which should be set to 0 for the plug-in devices.

The ***refid*** parameter is a string of maximum 4 characters which is displayed for example in the output of the ntpq command. We propose to set refid depending on the card type, for example:

```
GPS card      refid GPSi
DCF77 card    refid DCFi
TCR card      refid TCRi
PTP card      refid PTPi
```

The fudge command "flag1 1 time2 7200" can be used to set the so called trust time interval for the card. The trust time is a time interval for which the card is still accepted as reference time source if it has been synchronized but then starts freewheeling, e.g. because the antenna has been disconnected.

Usually the oscillator on the card is much better than the cheap crystal on the PC's mainboard, so if the oscillator has been disciplined before it makes sense to keep on using the card as time source for a while even if it starts freewheeling, instead of discarding the time source with the good oscillator immediately and relying on undisciplined system time.

If the trust time interval is not explicitely configured using the fudge command then the default trust time of 30 minutes is used. In the example above the trust time is set to 2 hours (7200 seconds).

6.1.13.5 Using Meinberg PCI and USB devices with NTP's SHM driver

6.1.13.6 Accuracy Considerations SHM versus Parse Driver

6.1.14 NTP with Meinberg Devices under Windows

On **Windows platforms**, NTP does not currently support most external reference clocks directly. Instead, the Meinberg driver can be used together with most internal and external Meinberg radio clocks to discipline the Windows system time. The NTP service can then be used to make the disciplined Windows system time available to NTP clients on the network.

If NTP is installed using the GUI installer from the Meinberg NTP download page and the setup program detects the Meinberg driver package which has already been installed before then the NTP installer suggests to create an appropriate NTP configuration labelled "Follow Meinberg Time Service".

This configuration should include the following lines:

```
server 127.127.1.0            # local clock
fudge 127.127.1.0 stratum 0   # disciplined by radio clock
```

Since in this case the Windows system time is disciplined by a radio clock, Local Clock's *stratum* should be forced to 0. The NTP server is then visible as stratum 1 server on the network.

For this special mode of operation no **driftfile** should be specified, and if a **ntp.drift** file already exists on the machine, it should be deleted. Otherwise the NTP service might try to correct the system clock drift, thus working against the radio clock driver, resulting in a poor time synchronization quality.

6.1.15 NTP Broadcast Mode

The NTP reference implementation does not use broadcasting of NTP packets by default. Usually clients send request packets to a server, and the server sends a reply packet. This makes it possible for the client to estimate and thus compensate the network delay for each individual packet exchange. See also chapter “*Network Latency Compensation by the NTP Protocol*”.

In broadcast mode there is a one-way network propagation delay which can not be estimated by the client, so the resulting accuracy is worse than with a client/server configuration.

If NTP broadcasts ought to be enabled anyway then one broadcast directive has to be added to the ntp.conf file for each subnet which is to receive NTP broadcast packets. e.g.:

```
broadcast 172.16.255.255
```

if the broadcast address is 172.16.255.255 according to the current network settings.

NTP cients which are to receive NTP broadcast packets also need to be explicitely configured as broadcast clients by adding the folowing directive to the ntp.conf file:

```
broadcastclient
```

Please note authentication should be used for broadcast mode in order to prevent broadcast clients from accepting NTP broadcasts from any node on the network. Otherwise clients might accept broadcast packets from any device on the network which sends NTP broadcasts intentionally or unintentionally.

6.1.16 NTP Multicast Mode

6.1.17 Using Hardware PPS Signals with NTP

6.1.18 Getting Started with NTP and Troubleshooting

Basically the NTP service can work both as server or client. If a Meinberg PCI card or external reference clock has been installed on a computer then this computer can be configured as a time server which makes its accurate time available on the network.

The way to configure the NTP program to use the card as reference time source is a little bit different e.g. for Windows and Linux, (see chapters “*NTP with Meinberg Refclocks under Unix-like Systems*”, and “*NTP with Meinberg Devices under Windows*”) but beside this the basic way it works is identical.

6.1.18.1 Don't Change the System Time While NTP Is Running

NTP has not been designed to correct sudden time steps immediately. It has a filter where the results of several queries to the NTP server(s) are evaluated, "spikes" due to queued network packets are sorted out, etc. The filter doesn't only compute the current time offset, it also determines how fast the system time drifts away from the real time, so that the drift can be compensated. Once the filter has been filled NTP starts to adjust the system time smoothly to compensate the time offset and drift determined by the filter.

NTP expects the system time to run monotonically, so it can do its work. The system time should **never** be changed manually while NTP is running. If this happens then the filtered time offset computed by NTP suddenly steps, and NTP refuses to follow this time step (the changed time offset) immediately. Also, the computation of the system clock drift by the filter is totally messed up in this case. NTP accepts the new time offset only if it persists for several polling cycles, and only if it is not too large. This can take up to more than 15 minutes. Then it sets the system time, discards all filter data and restarts polling/filtering from scratch.

If the system time is changed manually by more than about 1000 seconds while NTP is running then NTP will even abort itself with a log message saying something like: "The system time has been changed significantly. This can only have been done by the administrator, who should know what he's doing, so I'm giving up."

To check how NTP disciplines the system time the system time should be changed **before** NTP has been started, and NTP should be started thereafter. Running "ntpq -p" periodically in a command line window should show how the reported time offset (in milliseconds) decreases over time until it stays at some minimum. Since the system time is disciplined continuously the offset should stay around this minimum and not increase again. NTP can generate some statistics files which can be evaluated to check the synchronization performance over time.

6.1.18.2 Time Sources Need to Be Synchronized

When the NTP daemon (ntpd) receives the time from a reference time source then it also checks whether that reference time source is synchronized, or not. If the reference time source is **not synchronized** then it is **not accepted** by the NTP daemon. If the NTP daemon does not have any reference time source which claims to be synchronized then it does not start to discipline the system time.

When ntpd is running then you can use the command "ntpq -p" to to check the status of the NTP service on the local or on a remote machine to see whether ntpd accepts the configured time source, e.g.:

```
# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*GENERIC(0)      .GPSi.     0 l   18   64  377    0.000   -0.004   0.005
```

In the example above the reach column reads 377 which means the last 8 queries to read the time from the ref clock have been successful. If the refclock can not be accessed **or** the card is not synchronized then the reach value may stay at or go back to 0.

You can run the command "ntpq -p" in a command line window to check the status of the NTP service on the local or a remote machine.

In the examples below ip.addr.of.server is the server's IP address or hostname, and ip.addr.of.client is the client's IP address or hostname:

6.1.18.3 Check if the NTP server claims to be synchronized

On the NTP server machine run "ntpq -p", maybe repeatedly, and check the output:

```
ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*LOCAL(0)        .LCL0.       0 l   14   64  377    0.000    0.000   0.002
```

If the output looks like above, i.e. there's a '*' at the beginning of the line, then the NTP service is synchronized to its own system time (the "local clock") which is served to the network, and NTP clients should accept this server as reference time source.

However, as long as there is no '*' as in the example below then the clients won't accept the server:

```
ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 LOCAL(0)        .LCL0.       0 l   14   64  003    0.000    0.000   0.002
```

This may happen during a few minutes after the NTP service has started. In the example above the "reach" column is 003, which indicates the service has just been started a moment ago.

6.1.18.4 Check if the client synchronizes to the server

The NTP service on the client machine should have been configured to query the time from the NTP server. Run the command in a command line window on the client:

```
ntpq -p ip.addr.of.client
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*ip.addr.of.server .LCL0.     1 u    9   64  377    0.227   -0.659   0.402
```

This means the NTP service on client ip.addr.of.client is synchronized to ip.addr.of.server, which in turn is synchronized to its local clock. If there is no '*' at the beginning of the line, and the "reach" column is "000" then the client is unable to or does not synchronize to the server.

This may be the case if

- the server does not claim to be synchronized
- the client's request network packets don't arrive at the server, or the server's reply packets don't arrive at the client, which may be a firewall issue, if UDP port 123 is blocked for incoming or outgoing packets on the client or server.

6.1.18.5 If the client does not synchronize to the server, check if the NTP packet exchange works correctly.

The ntpdate program which is part of the NTP package can be used to make sure there is no firewall between the client and the server which blocks NTP packets.

The ntpdate program can be run in a command line window on the client. By default the program sends 4 requests to the NTP server specified on the command line (ip.addr.of.server in the example below), and thus expects 4 replies. The -d parameter lets the program print some details:

```
ntpdate -d ip.addr.of.server
 7 Sep 10:51:58 ntpdate[29435]: ntpdate 4.2.0a@1.1190-r Sat Mar 19
19:20:11 UTC 2005 (1)
Looking for host ip.addr.of.server and service ntp
host found : ip.addr.of.server
transmit(ip.addr.of.server)
transmit(ip.addr.of.server)
transmit(ip.addr.of.server)
transmit(ip.addr.of.server)
transmit(ip.addr.of.server)
ip.addr.of.server: Server dropped: no data
server ip.addr.of.server, port 123
stratum 0, precision 0, leap 00, trust 000
refid [ip.addr.of.server], delay 0.00000, dispersion 64.00000
transmitted 4, in filter 4
reference time:    00000000.00000000  Thu, Feb  7 2036  7:28:16.000
originate timestamp: 00000000.00000000  Thu, Feb  7 2036  7:28:16.000
transmit timestamp:  d0307bb1.f33b7521  Tue, Sep  7 2010 10:52:01.950
filter delay:  0.00000  0.00000  0.00000  0.00000
         0.00000  0.00000  0.00000  0.00000
filter offset: 0.000000 0.000000 0.000000 0.000000
         0.000000 0.000000 0.000000 0.000000
delay 0.00000, dispersion 64.00000
offset 0.000000

 7 Sep 10:52:02 ntpdate[29435]: no server suitable for synchronization found
```

In the example above there are transmit lines but no receive lines, which means the client does not receive any replies. Since there are no replies the output says: "Server dropped: no data", and at the end: "no server suitable for synchronization found".

If no replies are received then there may be a firewall on the client or on the server which blocks the request and/or reply packets, the NTP server could be down, or the NTP program on the server may not be running.

In the example below there are 4 transmit lines for the request packets followed by receive lines indicating that replies are being received:

```
ntpdate -d ip.addr.of.server
 7 Sep 10:50:51 ntpdate[29417]: ntpdate 4.2.0a@1.1190-r Sat Mar 19
19:20:11 UTC 2005 (1)
Looking for host ip.addr.of.server and service ntp
host found : ip.addr.of.server
transmit(ip.addr.of.server)
receive(ip.addr.of.server)
transmit(ip.addr.of.server)
receive(ip.addr.of.server)
transmit(ip.addr.of.server)
```

```
receive(ip.addr.of.server)
transmit(ip.addr.of.server)
receive(ip.addr.of.server)
transmit()
server ip.addr.of.server, port 123
stratum 1, precision -19, leap 00, trust 000
refid [LCL], delay 0.02582, dispersion 0.00069
transmitted 4, in filter 4
reference time:    d0307b60.b14a9302  Tue, Sep  7 2010 10:50:40.692
originate timestamp: d0307b6b.45f17a9d  Tue, Sep  7 2010 10:50:51.273
transmit timestamp:  d0307b6b.4633482b  Tue, Sep  7 2010 10:50:51.274
filter delay:  0.02582  0.02588  0.03105  0.02864
         0.00000  0.00000  0.00000  0.00000
filter offset: -0.00113 -0.00119 -0.00378 -0.00256
         0.000000 0.000000 0.000000 0.000000
delay 0.02582, dispersion 0.00069
offset -0.001136
```

The replies are also checked to see if the server would be accepted as time source, and if everything is OK the estimated current time offset between the server and the client is displayed, which is -1.136 milliseconds in the example above.

In this case there NTP client service should be able to synchronize to the upstream NTP server without problems.

There may also be a case where reply packets from the NTP server are received but the ntpdate program says anyway "no server suitable for synchronization found".

In this case the packet exchange works correctly, but the NTP server may not claim to be synchronized, in which case it is not accepted by the client. See chapter "*Checking the NTP status*" above.

6.1.19 Using NTP in a Windows Active Directory Domain

If you have a Windows Active Directory domain installed then the Windows time service (w32time) running on the domain controller (PDC) marks that PDC as authoritative time source for the domain, so all the domain members autodetect the PDC as authoritative time source and synchronize their time to the PDC automatically.

If you would replace the w32time service on the PDC by the NTP program then the PDC would not be detected automatically as authoritative time source anymore, and thus the domain members be able to synchronize their time automatically anymore.

So you better let the w32time service do its work on the PDC and set up a different PC or server as time server. Simply install the card, driver, and NTP package as described above on that PC, and then let the “Internet time server" configured on the PDC point to that special PC.

6.1.20 Building NTP from Sources

6.2 The Precision Time Protocol (PTP/IEEE1588)

6.3 RADclock Daemon

Keypoints:

•Invented by Julien Ridoux
http://www.cubinlab.ee.unimelb.edu.au/radclock

•“Feed-Forward” approach

•Uses NTP network packets

•Easier to be used in virtualized systems

•FreeBSD kernel support since 5.3

6.4 The TIME and DAYTIME Protocols

6.5 Time Synchronization using NetBIOS/NETBEUI

6.6 General Network Time Transfer Aspects

Early network time protocols transferred the time only with limited resolution, e.g. to the second, and didn't try to figure out how long a packet had been travelling on the network. As a result the network delay could not be determined, and thus could not be compensated, which resulted in a time offset error on the client.

The NTP protocol was the first attempt to determine the network latency and thus increase the possible accuracy for the client which queries the time from a server. It uses four time stamps to achieve this:

- t1: Client sends request packet to server
- t2: Server receives request packet from client
- t3: Server sends reply packet to client
- t4: Client receives reply packet from server

So this yields four timestamps from one packet exchange, but 2 of these timestamps refer to the time on the client, while the other 2 timestamps refer to the reference time on the server.

So the client can use a simple algorithm to compute from these four timestamps:

- What's the offset between server time and client time?
- How long did the request and reply packet travel on the network?

This kind of computation works fine as long as the network packet propagation delay is the same for both directions, to the server and back to the client. However, the network delay depends on several conditions, so on real networks, it is not constant. In fact the packet delay may vary for each packet exchange. This means filtering is required to compute the mean delay and filter out spikes.

At the server side a request packet is timestamped when it comes in from a client, and another timestamp is added after the request has been processed, i.e. when the reply packet is sent back to the client. This is pretty easy.

Only the client has all the 4 timestamps available after a packet exchange, so only the client can determine the time offset and network delay, and try to filter out spikes in the network delay.

As a consequence, the accuracy achievable on a client does not only depend on the accuracy of the server, it depends strongly on the implementation of the client software.

So when talking about NTP or PTP we need to distinguish between the protocol, i.e. the format of the network packets, and the implementation running on the client which evaluates the received packets.

6.7 Latencies due to Network Packet Transfers

If the time difference between two computers is to be determined across the network then the first step is to send a network packet with the current time from one computer to the other computer. Unfortunately there are a number of delays when a packet is sent across the network:

1. The sending program picks up a time stamp and puts it into a network packet.
2. The network packet is then passed to the network protocol stack where it is passed down from the sending user space application to the network drivers which partially run in kernel space, where it finally ends up in a send queue. This introduces a delay which depends on the CPU power and the system load (interrupt requests).
3. After the packet has been sent by the transmitting program, but before the packet actually goes onto the wire, the transmitting process or drivers may occasionally be preempted in a multitasking system. This means other processes may run for a certain time interval before the packet processing continues and the packet goes out to the network. The time interval where the transmitting process is waiting to continue can be huge compared to other latencies.
4. The network driver waits until the network wire is unused and starts to transmit the packet. If there's a collision on the wire then transmission is aborted and retried after a unknown, random delay.
5. Once the packet is on the wire the propagation delay is pretty constant, depending on the length of the wire. If there's a network hub between the sender and the receiver then this also

introduces an additional delay, which is pretty constant, though. If there's a router or switch between the two nodes then the packet may be queued for an undetermined amount of time, which also results in an unknown delay.

6. If the packet arrives at its destination then the network driver generates an interrupt request to let the packet be fetched by the protocol drivers. It also takes an unknown amount of time until this is done, depending on the CPU power, whether there are higher-prioritized interrupts just being handled, etc.

7. Current Gigabit NICs support a feature called **interrupt coalescing**. This means the card does not generate an interrupt request for every single received packet. Instead, several packets can be queued on the card. An interrupt request is only generated after several packets have already been received, so the NIC driver can retrieve several packets from the queue at once. This feature increase the maximum throughput for a network card, but it is bad for timing applications since it inserts additional, unknown delays.
See also: http://www.google.com/search?q=interrupt+coalescing

8. Finally the packet is moved up the protocol stack, moved from kernel space back to user space, and passed to the application which then takes a time stamp of its own system time in order to compute the difference to the time stamp from the incoming packet.

9. After the packet has come in from the wire, but before it actually arrives at the waiting application which takes a receive timestamp, the waiting application or the protocol drivers may occasionally be preempted in a multitasking system. This is similar to preemption at the transmitting side and causes similar latencies.

The application which receives the packet can compute the time difference between the time stamp in the received packet and its own current time when the packet has been received. However, there's a bunch of unknown transmission delays and latencies, and unless there are additional techniques the receiving application has no chance to distinguish which amount of the computed time difference is due to the transmission delays, and which amount is the real time offset between both machines.

6.8 Network Latency Compensation by the NTP Protocol

The Network Time Protocol (NTP) tries to determine the network delay by sending pairs of request/reply packets. This means a NTP client sends a packet to the NTP server and the NTP server sends a reply back to the client. In NTP terminology this is called polling. Each polling event yields four time stamps:

- the time of the client when the request packet is transmitted
- the time of the server when the request packet is received
- the time of the server when the reply packet is transmitted
- the time of the client when the reply packet is received

After the client has received the reply packet it can evaluate the four time stamps to compute the overall turnaround time. The client then assumes the one way delay is half of the turnaround time, so the remainder of the computed time difference mentioned above must be the time difference between both computers.

This works well if the packet delays for the request packet and the reply packet are similar.

However, there are cases where the request packet is delayed and the reply packet is not, or vice-versa, which results in an asymmetry of the propagation delays. The NTP client evaluates data from several sequential polling cycles using statistical methods, tries to detect such asymmetries and discards the timestamps from such packets as outliers.

Also, asymmetric network connections like ADSL lines introduce a systematic asymmetry for the propagation delays due to the different transmission speeds in both directions. Such systematic delays can not be determined by the NTP protocol and thus cause a systematic time offset on the client.

Anyway, in most cases the statistical methods to evaluate the polling results in client/server mode yield quite good results without requiring special network cards or switches. However, since many latencies are unpredictable you can **not guarantee** a certain accuracy.

NTP can also be configured to work in **broadcast mode**, i.e. the NTP server sends broadcast packets in periodic intervals. These packets can be received and evaluated by all NTP clients which have been configured accordingly.

The problem with this setup is that the network delay is not determined, or it is only determined once when the client starts to receive broadcasts. If e.g. the network route changes and thus the network delay varies this is not detected by the clients, and thus the network delay is not compensated correctly.

So NTP broadcast mode can not yield an accuracy better than the standard client/server mode. In most cases broadcast accuracy is significantly worse. On the other hand, if a huge number of clients shall be synchronized and the accuracy is sufficient, using broadcast mode can be a good option.

6.9 Network Latency Compensation by the PTP/IEEE1588 Protocol

The PTP protocol has been developed many years after the NTP protocol, so it supports an enhanced way to measure network latencies. In the chapters above we have seen that the variable network latencies are due to the execution time in the transmitting or receiving node. On the other hand, the pure cable delays depend only on the cable length, but do not vary over time.

In order to compensate the receive delay (i.e. when a packet comes in from the wire until it arrives at the application) packets can be time stamped when they come in from the wire. This is done by a **time stamp unit (TSU)** which includes a pattern matcher which has to identify incoming PTP packets in the bit stream from the wire, and take a time stamp if such a packet is detected. Both the network card driver and the application have to provide a way (an API call) to let the application retrieve that time stamp from the NIC driver and assign it to the associated network packet. This way the application can compute the difference between the time it has received the packet and the time the packet has arrived from the wire, and account for that delay.

Obviously the same has to be done for outgoing packets, i.e. determine the time interval from when the packet is sent by the application until it really goes onto the wire. The calculated delay has to be passed to the receiver which has to account for that delay. Unfortunately the time stamp can only be taken when the packet goes out onto the wire, so when the time stamp is available the packet has already been sent. The PTP protocol accounts for this situation by sending a so-called follow-up packet which contains the time stamp of the previous packet. The receiver then gets the original packet which is time-stamped when coming in, plus the follow-up packet which contains the transmission delay and can thus account for both the delays. Using a point-to-point connection

between the transmitting and the receiving node you can yield an accuracy down to a couple of nanoseconds by hardware time stamping.

However, there's still a last delay which is not yet known by our server and client. This is the propagation delay across intermediate nodes like switches and routers. For example, if a switch receives an incoming packet at one port, and the outgoing port is just be used by another packet then the incoming packet is queued internally in a FIFO. This can take up to several tens of milliseconds (!), depending on the type of switch, the network load and the queue depth. The problem is that neither the transmitting nor the receiving node can determine whether a packet has been passed on directly, or has been queued, and for how long it has been queued.

So even if both endpoints provide a way for hardware time stamping, a single standard switch between them can screw up the accuracy. The only way to avoid this are either to use "dumb" hubs which just duplicate the packets without queuing them, or to use special switches which are aware of PTP packets and handle them in a special way.

The PTP protocol defines a special "transparent" or "boundary” clock which can be implemented in switches or routers in order to handle the PTP packets in a special way which compensates the switch's delays.

The statements above also explain why it is nearly impossible to get full accuracy with PTP over a wide area network (WAN): The network nodes between two locations are usually owned by a service provider, and the customer does not even know which route the network packets take from one location to the next, nor does he know which devices are passed along that route.

6.10 Comparison: NTP versus PTP/IEEE1588

The main difference between NTP and PTP is that the statistics implemented by the NTP algorithms yield quite good results even over WAN connections, without requiring special hardware.

PTP **can** achieve much better accuracy than NTP, but this is **only** the case if **only** special hardware is used which explicitly supports PTP.

As a conclusion you can say that

1. NTP can achieve pretty good accuracy in both small and large networks where you don't know which routes the packets take, and you can't and don't have to rely on special hardware support for the protocol.

2. PTP can yield very high accuracy provided that the network infrastructure fully supports the protocol. Obviously this is easier to implement in a closed network where the administrators have full control over the network infrastrucure.

3. If the special hardware support for PTP is not available, PTP suffers from the same limitations as NTP, i.e. the unknown delays occurring during the transport of a network packet. Under these conditions NTP can even yield better results due to the statistical methods it uses.

Meinberg has made some tests with NTP using the same hardware time stamping methods as PTP, and the results showed that NTP can yield the same accuracy as PTP if the basic conditions are similar.

The problem here is that the current specification of the NTP protocol does not provide a method to send a follow-up message to the client in order to let the client know when the original packet really made its way onto the wire. If such methods are added to the NTP protocol then this breaks compatibility with existing implementations of NTP.

Anyway, NTP can yield pretty good accuracy on Unix-like systems, even without dedicated hardware. The example below is from a Linux machine which synchronizes to one server on the local LAN and some other servers over the internet (times are in milliseconds):

```
# ntpq -p
   remote         refid    st t when poll reach   delay   offset  jitter
=======================================================================
*gateway.py.mein .GPSi.    1 u   18   64  377    0.153    0.033   0.036
+ptbtime1.ptb.de .PTB.     1 u    8   64  377   17.342   -0.710   1.032
-ptbtime2.ptb.de .PTB.     1 u   14   64  377   17.725   -0.780   1.026
+tick.usno.navy. .USNO.    1 u   62   64  377  113.213   -0.896   3.455
-tock.usno.navy. .USNO.    1 u   66   64  377  113.597    0.920   1.523
```

Server gateway is a Linux PC with a GPS PCI card on the local LAN. See the low packet delay and the determined time offset of 33 microseconds only. Ptbtime1 and ptbtime2 are public NTP servers of the German PTB (i.e. the German counterpart of the U.S. NIST). Please note the determined time offset is still below 1 millisecond, though the packet delay is about 17 milliseconds. Tick and tock are the public USNO servers, and the determined time offset is still below ± 1 millisecond even though the packet delay is 113 milliseconds, and the network route goes from one continent to another.

This impressingly shows the capabilities of NTP. However, the chapters above should also make clear where the limitations can be found.

7 Time Dissemination by Radio Signals

7.1 *Time Dissemination by Satellites*

7.1.1 GPS

The Global Positioning System (GPS) is a satellite-based radio-positioning, navigation, and time-transfer system. It was installed by the United States Department of Defense and provides two levels of accuracy; the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). The SPS has been made available to the general public, but the PPS is encrypted and only available for authorized (mostly military) users.

GPS operates by accurately measuring the propagation time of signals transmitted from the satellites to the user's receiver. A nominal constellation of 21 satellites together with several active spares, in six orbital planes at about 20000 km altitude, provides a minimum of four satellites in view 24 hours a day at every point on the globe. Four satellites must be received simultaneously to determine both the receiver position (x, y, z) and receiver clock offset from GPS system time. All satellites are monitored by ground control stations which determine the exact orbit parameters and the clock offset of the satellites' on-board atomic clocks. These parameters are uploaded to the satellites and become part of a navigation message which is retransmitted by the satellites and passed to the user's receiver.

The high precision orbit parameters of the satellites are called ephemeris parameters, and a reduced precision subset of the ephemeris parameters is called a satellite´s almanac. While ephemeris parameters must be evaluated to compute the receiver's position and clock offset, almanac parameters are used to check which satellites are in view from a given receiver position at a given time. Each satellite transmits its own set of ephemeris parameters, and almanac parameters of all existing satellites.

GPS system time differs from the universal time scale (UTC) by the number of leap seconds that have been inserted into the UTC time scale since GPS was initiated in 1980. The current number of leap seconds is part of the navigation message supplied by the satellites, so a receiver's internal real time can be based on UTC. Conversion to local time and handling of Daylight Savings Time each year is done by the receiver's microprocessor once these parameters have been programmed by the user.

7.1.2 GLONASS

7.1.3 Compass / Beidou

7.1.4 Galileo

7.2 Time Dissemination by Long Wave Transmitters

7.2.1 DCF77 in Germany

The DCF77 long wave (also Low Frequency, LF) transmitter is located in Mainflingen near Frankfurt, Germany. The LF transmitter disseminates the Legal Time of the Federal Republic of Germany which is either the Central European Time, CET (in German: Mitteleuropäische Zeit, MEZ) or the Central European Summer Time, CEST (in German: Mitteleuropäische Sommerzeit, MESZ). The DCF77 signal can be received in large parts of Europe.

The DCF77 frequency and signal is derived from the atomic clocks of the Physikalisch-Technische Bundesanstalt (PTB) in Braunschweig, Germany, the national institute for science and technology and the highest technical authority of the Federal Republic of Germany for the field of metrology and physical safety engineering. Transmission is controlled by the PTB's Department of Length and Time. The coded information includes the current time of day, date of month, and day of week in coded one-second pulses. The complete time message is transmitted once every minute.

At the beginning of every second the, amplitude of the precise 77.5 kHz carrier frequency is reduced by 75% for a period of 0.1 or 0.2 sec. The length of these time marks represents a binary coding scheme using the short time mark for logical *zeroes* and the long time mark for logical *ones*. Data representing the current date and time and some parity and status bits are encoded in the time marks from the 15th to the 58th second of every minute. The absence of the time mark at the 59th second indicates that a new minute will begin with the next time mark.

In order to increase the accuracy of the demodulated time marks, the carrier of DCF77 is also modulated with a pseudo-random phase noise. The pseudo-random sequence has a length of 512 bits, and is transmitted in the interval between the AM marks. Due to the pseudo-random characteristic of the sequence the mean deviation of the carrier phase is zero. The phase modulated carrier can be received with a larger bandwidth receiver. Correlation algorithms also used with satellite transmission techniques allow PZF receivers to determine the correct time with an accuracy of microseconds, which is far superior to the accuracy achieved by standard AM receivers.

7.2.2 MSF/Rugby in the United Kingdom

7.2.3 WWVB in the United States

7.2.4 HBG in Switzerland

HBG was a long wave time signal transmitter located in Switzerland near Lake Geneva (Lac Léman, Genfer See). It was put into operation in 1966, with a 75 kHz carrier frequency and 25 kW transmission power.

The transmitter was last operated by the Swiss Metrology Institute METAS but was put out of operation at the end of 2011. See
http://www.news.admin.ch/message/?lang=de&msg-id=28671

7.2.5 JJY in Japan

7.3 Comparison Satellite Systems vs. Long Wave Signals

7.3.1 Signal Reception

Satellite receivers usually require an outdoor antenna with clear view to the sky for reliable operation whereas long wave signals can eventually be received inside buildings. However, modern buildings are often reinforced concrete constructions with much metal inside the walls and metal-coated window panes, so the original signal is very weak inside such buildings, and thus an outdoor antenna is often recommended or even required for proper operation of long wave receivers.

On the other hand, long wave signals are usually very susceptible to electrical noise which may superimpose the original signal and thus inhibits reliably decoding of the timing signal. Satellite signals are usually based on some enhanced broadcast technology like spread spectrum technique, so they are much less susceptible to electrical noise than the simple long-wave signals.

7.3.2 Signal Propagation Delay Compensation

Time signal transmitters usually start to transmit a well-defined signal at a given time. When the signal arrives at timing receiver the receiver knows at which point in time the signal has been transmitted, but it has taken some time for the signal to propagate from the transmitter to the receiver, so the signal arrives too late. Thus, if the receiver is to provide accurate time it needs a way to determine the signal propagation delay, and compensate it.

In most cases a long-wave time signal is only broadcasted by a single station, so the signal propagation delay depends basically on the receiver's distance from the transmitter.

For satellite-based navigational systems it is anyway required that receivers can track several satellites at the same time, and exact measurement of the signal propagation delay is a prerequisite for accurate computation of the receiver's position. So the signal delay is compensated automatically, and very accurately, and thus timing based on navigational satellite systems yields a very high accuracy without requiring any manual intervention depending on the receiver position.

8 Hardware Reference Time Sources

Hardware reference time sources are hardware devices which are connected to the local computer in order to provide an accurate time. Such devices can be PCI or PCI Express cards installed inside the computer, USB devices plugged in externally, or even radio clocks connected via a serial RS-232 interface.

Such hardware reference time sources can be used to discipline the system time, and additionally or alternatively they can be used directly by applications which require accurate timestamps.

In order to discipline the system time a piece of software is required which reads both the reference time and the system time in cyclic intervals, computes the difference and applies an adjustment to steer the system time such that the difference becomes as small as possible.

How good the system time can be disciplined using a hardware time reference depends on:

- the accuracy of the reference time source
- the access time required to read a time stamp
- the granularity of the data structure used for transport a time stamp
- the characteristics of the operating system and its timekeeping
- the implementation of the control loop in the time synchronization application.

Meinberg provides devices which get the reference time either from the GPS satellites, from an IRIG generator, from one of several public long wave transmitters, or from a PTP grandmaster. There are also devices available which have several receivers built in and thus can evaluate several reference time signals.

8.1 Reference Time Signal Type Considerations

8.1.1 Satellite Signals

8.1.2 Longwave Signals

8.1.3 IRIG And Similar Timecode Signals

The frequently-used term *IRIG signals* usually refers to a whole group of serial timecodes which use a continuous stream of binary data to transmit information on date and time. The individual time code formats can be distinguished by the signal characteristics, e.g. modulated versus unmodulated signals, by the data rate, and by the kind of information included in the transmitted data, so which specific timecode should be used preferably for an application depends on the specific requirements of that application.

Back in 1956 the **TeleCommunication Working Group (TCWG)** of the American **Inter Range Instrumentation Group (IRIG)** was mandated to standardize different time code formats, resulting in *IRIG Document 104-60* which was published in 1960. Over the years there have been a number of revisions and extensions to the original specification. The current version of the document is ***IRIG Standard 200-04*** which was published in 2004 and is available for download on the U.S. Range Commander Councils publications web page:
https://wsmrc2vger.wsmr.army.mil/rcc/PUBS/pubs.htm.

8.1.3.1 Original IRIG Signals

Timecode signals defined by the various versions of *IRIG Standard 200* are classified by a letter indicating the basic format, plus a three digit numeric code specifying the signal characteristics according to the following table:

First letter: **Format / Data Rate**	A B D E G H	1000 pps 100 pps 1 ppm 10 pps 10000 pps 1 pps
1st digit: **Modulation**	0 1 2	DC Level Shift (DCLS), pulse width code Sine wave carrier, amplitude modulated Manchester modulated **(Note 1)**
2nd digit: **Carrier Frequency, Resolution**	0 1 2 3 4 5	No carrier / index count interval 100 Hz / 10 millisecond resolution 1 kHz / 1 millisecond resolution 10 kHz / 100 microsecond resolution 100 kHz / 10 microsecond resolution 1 MHz / 1 microsecond resolution
3rd digit: **Coded Expressions**	0 1 2 3 4 5 6 7	TOY, CF, SBS TOY, CF TOY TOY, SBS TOY, YEAR, CF, SBS **(Note 2)** TOY, YEAR, CF **(Note 2)** TOY, YEAR **(Note 2)** TOY, YEAR, SBS **(Note 2)**

Note 1): Extension added in IRIG Standard 200-98 from 1998
Note 2): Extension added in IRIG Standard 200-04 from 2004

The **Coded Expressions** mentioned in the table above provide a timecode receiver with the following information:

- **TOY**: Time-Of-Year, including hours, minutes, seconds, and day-of-year. This information is included in every code type defined by the IRIG standards.

- **CF**: Control Field segment which can optionally be used to transport application-specific information. Some newer IRIG codes and codes defined by other standardization organizations use parts of the original control field segment to define well-known extensions providing additional useful information like year number, UTC offset, etc.

- **SBS**: Straight Binary Seconds, i.e. second-of-day. This field is optional, and the transmitted information can also be computed from the transmitted hours, minutes, and seconds. Probably this field has been useful in the pre-computer era when it was hard to do such computations in IRIG receivers.

- **YEAR**: A 2 digit year number, i.e. year-of-the-century. This field is useful to determine the calendar date unambiguously, see chapter *Selecting An Adequate Timecode Signal*. However, for the original IRIG signals this field has only be specified in IRIG Standard 200-04 from 2004.

Some popular legacy IRIG codes include IRIG-B122 and IRIG-B002. According to *IRIG standard 200* the characteristics of these signals are:

IRIG-B002:
- B: 100 pps data rate
- 0: DCLS pulse witdh code not modulated onto a carrier
- 0: no carrier frequency
- 2 includes hours, minutes, seconds, and day-of-year

IRIG-B122:
- B: 100 pps data rate
- 1: amplitude modulated sine wave
- 2: 1 kHz carrier frequency
- 2 includes hours, minutes, seconds, and day-of-year

IRIG-B126:
- B: 100 pps data rate
- 1: amplitude modulated sine wave
- 2: 1 kHz carrier frequency
- 2 includes hours, minutes, seconds, day-of-year, and 2 digit year number

So for example the IRIG-B126 code is similar to the popular IRIG-B122 code, but in addition transports a 2 digit year number which allows unambiguous conversion of the day-of-year to a calendar date.

In addition to the original IRIG codes there are some other popular timecodes which have been defined by other standardization organizations in order to meet the requirements of new applications. Those timecodes are often very similar to the original timecodes but provide some useful extensions.

8.1.3.2 AFNOR NF S87-500

In 1987 the French standardization organization **Associacion Française de Normalisation (AFNOR)** published a French standard AFNOR NF S87-500 which defines a timecode signal similar to IRIG signals. That AFNOR code also transports the current time and day-of-year, but has been defined to always include a 2 digit year number.

This timecode can be used as an unmodulated signal, or amplitude modulated onto a 1 kHz carrier frequency, so today it is basically similar to IRIG-B006/B126. However, these IRIG signals have only been defined in 2004, i.e. some years after AFNOR NF S87-500 has been published.

Optionally the AFNOR signal can also include the day-of-week number (1..7, 1 = Monday), plus the calendar date (month and day-of-month), plus an SBS number like the IRIG codes.

Unlike the original IRIG standards the AFNOR standard also defines the electrical signals to transport the unmodulated timecode, which should be according to RS-422 / RS-485 specifications.

The publication can be purchased following the links on the ANOR home page:
http://www.afnor.org

8.1.3.3 IEEE 1344-1995 and IEEE C37.118-2005

In 1995 a working group of the **Institute of Electrical and Electronics Engineers (IEEE)** published **IEEE standard 1344-1995** about *Synchrophasors for Power Systems* which also includes the specification of a timecode. This timecode is based on a IRIG-B122/B123 code but uses the control field (CF) to transport some additional information useful to overcome some limitations of the original IRIG time codes. These so-called IEEE 1344 extensions include:

- A 2 digit year number to be able to determine if a year is a leap year or not, thus allowing for a unambiguous conversion of the day-of-year number to a calendar date.

- Leap Second Information:
  - An announcement bit which is set up 59 seconds before the leap second event
  - A leap second indicator which is set during the leap second

- Daylight Saving Time (DST) Information:
  - An announcement bit which is set up to 59 seconds before DST status changes
  - A DST indicator which is set while DST is active

- UTC offset information. If local time is transported by the IRIG frame then this parameter can be used to determine UTC time.

  Warning: there are discrepancies both in the different standards, and in existing implementations, about the way the UTC offset has to be applied. See the next chapter for details.

- A 4 bit Time Figure Of Merit (TFOM) code which allows the IRIG generator to pass an accuracy specifier to IRIG receivers, where TFOM 0 means “highest accuracy”, and TFOM 15 (0F hex) means “unsynchronized”.

In 2005 the IEEE standard 1344-1995 was revised and the revised version was published as **IEEE standard C37.118-2005**. The revised standard defines the same extensions for the timecode as the

original IEEE 1344 standard, but unfortunately the way to handle the **UTC offset was completely messed up** in the revised version, which can possibly cause confusion and can result in a wrong UTC time computed by IRIG receivers. See the next chapter for details.

The publications of these standards can be purchased following the links on the IEEE home page: **http://www.ieee.org**

8.1.3.4 UTC Offset Discrepancies between IEEE1344-1995 and C37.118-2005

Care must be taken if the IEEE 1344 or IEEE C37.118 timecodes are used to transfer local time instead of UTC.

Both standards contain two pieces of text describing how to handle the UTC offset parameter:

- A table describing the assignment of the control bits, including an explanation how to handle the time offset bits

- A chapter of descriptive text explaining how UTC offsets are to be handled, including an example for the computation

In the IEEE standard 1344-1995 both pieces of text consistently and unambiguously explain how to handle the UTC offset.

In the IEEE standard C37.118-2005 the chapter of descriptive text has been modified and now defines the UTC offset just with the **reversed sign** compared the original IEEE standard 1344-1995. The associated computation example has been modified accordingly. While it is in general not wise to change specifications in this way, things are even worse since the explanation text in the control bit assignment table has **not** been updated accordingly and is still the same as in the original standard from 1995.

So **the IEEE C37.118-2005 standard contains two pieces of text which define two contradictory ways to handle the UTC offset.**

IRIG devices manufactured by Meinberg implement the IEEE 1344 timecode as specified in the IEEE standard 1344-1995. Some of these devices can alternatively be configured to use the C37.118 code, which is similar to the IEEE 1344 setting but evaluates the UTC offset with reversed sign as specified in by the modified text in IEEE standard C37.118-2005.

Attention:

There are 3rd party IRIG devices out there which **apply** the UTC offset as specified in the modified C37.118 text, but **claim to support** IEEE 1344 extensions. So if local time is transmitted in a timecode with IEEE 1344 extensions then **care must be taken that the UTC offset is evaluated by the IRIG receiver in the same way as output by the IRIG generator**. Otherwise the UTC time computed by the receiver may be absolutely wrong, and the system time of a PC synchronized by an IRIG receiver may be set in a wrong way.

To substantiate the statements above here are some quotes from the original and revised IEEE standards. In both documents the explanation for the time offset bits in the control bit assignment table F.1 reads:

```
Offset from coded IRIG-B time to UTC time.
IRIG coded time plus time offset (including
sign) equals UTC time at all times (offset will
change during daylight savings)
```

This is consistent with the descriptive text in IEEE standard1344-1995:

> ***F.3.4 Local time offset***
>
> *The local time offset is a 4 b binary count with a sign bit. An extra bit is included for an additional 1/2 h offset used by a few countries. The offset gives the hours difference (up to ± 16.5 h) between UTC time and the IRIG time (both BCD and SBS codes).* ***Adding*** *the offset to the IRIG-B time using the included sign gives UTC time (e.g., if the* ***IRIG-B time is 109:14:43:27*** *and the* ***offset is -06*** *given by the code 0110 (.0), then* ***UTC time is 109:08:43:27****). The local time offset should always give the true difference between IRIG code and UTC time, so the offset changes whenever a daylight savings time change is made. Keeping this offset consistent with UTC simplifies operation of remote equipment that uses UTC time.*

In IEEE standard C37.118-2005 the explanation for the time offset bits in the control bit assignment table F.1 is exactly the same as in IEEE 1344-1995, but the descriptive text says:

> ***F.1.4.4 Local time offset***
>
> *The local time offset is a 4-bit binary count with a sign bit. An extra bit has been included for an additional 0.5 h offset used by a few countries. The offset gives the hours difference (up to ± 16.5 h) between UTC time and the IRIG-B time (both BCD and SBS codes).* ***Subtracting*** *the offset from the IRIG-B time using the included sign gives UTC time. [For example, if the* ***IRIG-B time is 109:14:43:27*** *and the* ***offset is –06*** *given by the code 0110 (.0), then* ***UTC time is 109:20:43:27****.] The local time offset should always give the true difference between IRIG code and UTC time, so the offset changes whenever a DST change is made. Keeping this offset consistent with UTC simplifies operation of remote equipment that uses UTC time.*

So obviously the original standard consistently says:

```
IRIG time + UTC offset = UTC
```

whereas the revised standard from 2005 states in the descriptive text:

```
IRIG time – UTC offset = UTC
```

which leads to a wrong UTC time if the IRIG generator applies to the original standard and the IRIG receiver to the revised standard, or vice-versa.

8.1.3.5 Modulated vs. Unmodulated (DCLS) Timecode Signals

The raw time code is a continuous stream of binary data transmitted at a given rate. Optionally this binary data stream can be used to modulate a sine wave carrier of a defined frequency.

Since the logic levels of the raw data stream are usually represented by DC voltage levels, the unmodulated code frames are also called DC Level Shift signals, or DCLS signals.

Depending on the signal characteristics there is a wide range of applications for specific IRIG codes. For example, timecodes modulated onto a tone frequency carrier signal can be transmitted over a telephone line, or be recorded on a magnetic tape. On the other hand, DCLS signals can easily be transmitted by digital transmission lines like RS-485 or fiber optics.

Also the accuracy of a decoded timecode signal can depend on the signal characteristics. Due to its nature as digital signals DCLS timecodes have well-defined slopes, and the propagation delays of digital line drivers and receivers are usually pretty constant. It is pretty easy to generate an accurate trigger signal from a DCLS slope and thus yield a high accuracy from a received DCLS timecode.

For modulated signals the exact start of a signal frame is bound to the zero-crossing of the carrier signal, i.e. the carrier phase. On the receiver side it requires much more effort to detect the exact zero crossing point of a modulated sine-wave signal than capturing a digital slope. Also, modulated timecodes often use filters, transformers, automatic gain control (AGC) circuits, etc., in the signal transmission path which delay the analog signal and thus affect the carrier phase. It requires much effort to compensate such signal delays and yield an accuracy from a modulated timecode signal which is in the same range as the accuracy of a DCLS signal.

8.1.3.6 Selecting An Adequate Timecode Signal

Accuracy requirements or the availability of signal transmission infrastructure (cables or similar) can often help to prefer some modulated or DCLS timecode. However, also the kind of information transported by a specific code has to meet the requirements of an application.

All types of IRIG signals contain the day-of-year number and the current time of that day. This is sufficient to drive wall clocks which simply display the time transmitted by the IRIG signal, but this may not be sufficient e.g. to synchronize the computer time. For example, if the system time of computers is to be disciplined in a reliable way using a timecode signal then the timecode receiver needs to be able to derive both the correct calendar date and the current UTC time from the incoming signal.

Unfortunately most basic IRIG signal frame types like B002 or B122 neither include the year number, nor do they provide any information telling whether the transmitted time is UTC, or some local (DST or standard) time with a certain offset to UTC.

If the day-of-year number from a received timecode signal is to be converted to a human readable calendar date then obviously the conversion is ambiguous after February 28, since the next day can either be March 1, or February 29, depending on whether the current year is a leap year, or not. Thus the timecode receiver needs to know the current year number, either by configuration, or preferably by using a timecode format providing the year number.

Also, if the time transmitted by a timecode signal is to be converted to UTC in order to discipline a computer's UTC time then the receiver needs to know whether the transmitted time is UTC or local time, and if it is local time it needs to know the UTC offset. Of course it is easily possible to

configure the UTC offset on the receiver side, but at the beginning or end of DST the UTC offset will jump back or forth by the amount of the DST offset, in which case the converted UTC time would also step back and forth, and so would the system UTC time of the computer to be disciplined, which may have huge impacts e.g. on database applications.

Newer IRIG formats are usually compatible with the old basic formats, but in addition include some or all of the required information in extensions coded in the control field segment of the IRIG frame.

A reliable way to synchronize the computer time is to use at least a timecode which includes the year number (e.g. IRIG-B126, AFNOR NF S87-500, or one of the IEEE formats), and let the timecode generator transmit UTC time.

If the local time must also be transmitted in the same timecode signal (e.g. to drive wall clocks) then the timecodes defined in IEEE 1344-1995 or IEEE C37.118-2005 are the best choice. The signals are compatible with the popular IRIG-B122 format, but timecode receivers which need to compute UTC time can do this easily since the UTC offset is also transmitted by those signals. Care must be taken, however, that both the timecode transmitter and the timecode receiver handle the UTC offset with the same sign. See chapter *UTC Offset Discrepancies between IEEE1344-1995 and C37.118-2005* for details.

8.2 Access Time Considerations

8.2.1 PCI Cards

8.2.1.1 PCI Express Limitations

Even though PCI Express has been designed for a high throughput, there is an overhead if single data items need to be transferred. This it due to the serial data transmission used with PCI Express.

If an application accesses a register or memory location on a PCI Express card then the read command plus address from which to read needs to be serialized on the mainboard, transmitted serially to the card, be converted back to a parallel address, the data be read parallel and then serialized, the serialized data transferred back to the mainboard, and finally be converted back to parallel to be read by the CPU.

The high throughput with PCI Express can only be achieved if large amounts of data are transferred using DMA, in which case the steps above are pipelined, e.g. while one data item is transferred to the mainboard, the next data item is already serialized, etc. so there is a constant data stream at very high speed.

If you need to read 2 x 32 bit data words to get a 64 bit timestamp, e.g. including seconds and fractions of a second, then all of the above has to happen twice, and pipelining is not possible, so the two read accesses take 4 to 5 microseconds to execute.

Unfortunately it does not make sense to use DMA just to transfer 64 bits of data whenever an application requires this. The overhead to control the DMA transfer, plus the following DMA transfer took longer to execute than the 2 subsequent simple read accesses.

In addition, the PCI bus is subject to bus arbitration, so even a DMA transfer can be delayed if there is another ongoing transfer using the same partial bus.

This effectively limits the maximum access rate to read time stamps directly from a PCI Express card. The problem is not specific to a specific PCI card, mainboard, or operating system.

8.2.1.2 API Calls available for Meinberg PCI Cards

Meinberg PCI cards support different API calls, some of which require interaction with the microcontroller on the card, so there are execution times from a few microseconds up to about a millisecond on consecutive reads, depending on the type of microcontroller on the card. Please note the returned time is latched at the beginning of the call, so the returned time is associated to the entry of the call. This means you can still get high accuracy to discipline the system time, which usually only requires reading time stamp in intervals such as once per second, but there are some limitations for applications which need to read high resolution time at a very high rate.

Meinberg PCI Express cards (i.e. the ...PEX cards) provide a different PCI interface chip where the card's time counter chain is accessible through memory mapped registers. This means the API calls are as fast as possible and do not require interaction with the on-board microcontroller.

The execution time to read a 64 bit time stamp from our PEX cards is about 5 microseconds, but the limitation is due to the PCI Express bus, not due to the cards.

Unfortunately the PCI interface chip on the Meinberg standard PCI cards does not support the memory mapped registers, so PCI Express cards should be used preferably, if possible.

A separate whitepaper is available from Meinberg which describes the Meinberg driver policy and API concepts.

8.2.1.3 Circumventing PCI Access Times

E.g. under Windows the reference implementation of the NTP program uses the Windows QueryPerformanceCounter (QPC) API to interpolate the system time between 2 timer ticks. The Windows Hardware Abstraction Layer (HAL) uses one of the available timer circuits on the board to implement the call. If the HAL uses the power management timer (PM timer) which is simply a register in the chipset then it also takes about 3 microseconds to read the timestamp, simply because the register is in the peripheral chipset which is connected via a local bus. If the HAL uses the CPU's TSC registers instead then it takes just a couple of nanoseconds to read a timestamp.

So a way to get time stamps at a higher rate is to use an interpolation scheme where the PCI card's time stamp plus an associated TSC count are read in periodic intervals, e.g. once per second, then the application reads only the current CPU's TSC count for timestamping, and converts that TSC count to a real time stamp by relating it to the latest TSC/time stamp pair.

However, if the TSC is being used then there may be different problems, depending on the specific CPU type installed on the mainboard. E.g. the TSC clock frequency may change whenever the CPU clock frequency changes for power saving efforts (e.g. Intel's Speedstep, or AMD's Cool'n'Quiet), or the different TSC counters in different cores of the same CPU may not be synchronized, in which case the retrieved time stamps may not be consistent if a piece of code can be executed on different

CPU cores.

This may be fixed if the application is alway scheduled to run on the same CPU (process affinity / thread affinity) or if it is made sure the there are CPUs installed on the mainboard where the TSCs are always synchronized.

The mbgxhrtime example program from our Linux driver package implements the interpolation scheme described above. It starts an own thread to read the TSC/time stamp pairs once per second, calculates the real TSC frequency from the first 2 readings, and then generates the following time stamp using interpolation.

You also need to take into account that an API call which reads a time stamp may at any time be interrupted by a hardware IRQ, or be delayed if there is an ongoing DMA transfer from a different card on the same bus, which may fudge the time stamp you application receives.

Unfortunately it is very hard to implement high resolution, high accurate timestamping which works reliably, and potential errors have to be taken into account.

8.2.2 Native Serial Port

1.) how accurate is the internal time's second changeover:
better than 1 microsecond for GPS, a few milliseconds for non-PZF DCF77

2.) how close to the second changeover is the serial time string sent:
a few microseconds for GPS162, about 1 millisecond for C51

3.) how accurately can the serial time string be received
depends on whether the serial port is a real serial port, or whether RS232-to-USB or RS232-to-LAN converters are involved which introduce additional latencies depending on the converter type and model

4.) how long does it take until the receiving software becomes aware of the incoming timestring:
depends on the serial port driver. Todays UARTs usually have receive FIFOs which notify the driver/application for incoming characters only after several characters have been received, thus introducing a delay which is in most cases unspecified, unless the application can determine or change the recive FIFO trigger level.

5.) which effort does the receiving software take to compensate the transmission delay of the serial string, which depends on the baud rate and framing:
E.g. at 19200 baud it takes 52 microseconds to transmit a single bit, but for 2400 baud it takes 417 us. Framing 8N1 requires 10 bits per character, but 7E2 uses 11 characters per byte. So transferring a single character takes 0.52 or 0.57 ms at 19200, but takes 4.17 or 4.58 ms at 2400.

All the delays mentioned above can be added in a worst case configuration, thus making the resulting timing accuracy for an application much worse than the accuracy of the internal time inside the clocks.USB-to-Serial Converters

8.2.3 USB Devices

8.2.4 Fiber Optic

8.3 Time Resolution Considerations

8.4 Disciplined vs. Undisciplined Oscillator Considerations

Each oscillator generates an output frequency which is more or less off its nominal frequency. E.g., if the nominal frequency is 10.000000 MHz then one particular oscillator can run at 10.000001 MHz or 10.000002 MHz, and a different oscillator of the same type can run at 9.999998 MHz. The current real frequency also varies over time due to **ageing**, but also due to **variations in the ambient temperature**.

The better the quality of an oscillator, the less are the offsets from the nominal frequency, and the less is the drift. A high quality quartz oscillator which comes with a temperature compensation to minimize frequency variations due to temperature is called **Temperature Controlled Xtal Oscillator (TCXO)**. In most cases, however, the frequency is even more stable if the built-in crystal is kept at a constant temperature. So there are oscillators with come with a built-in oven, and thus such an oscillator is called **Oven Controlled Xtal Oscillator (OCXO)**.

The disadvantage of high quality oscillators is the power consumption which is usually higher than with cheap crystals, especially for OCXOs due to the built-in oven. This may be important if there's only limited power available, e.g. for USB devices which are only supplied via the USB connector.

Anyway, even the best oscillator may have a frequency offset from its nominal frequency, even though this offset is usually much smaller than the offset of a cheap oscillator or crystal.

Disciplining an oscillator means the frequency offset is determined by help of the incoming reference signal, and the oscillator is steered such that its output frequency is tweaked to the nominal frequency as good as possible. In order to generate a stable 1 PPS signal you just have to count exactly 10000000 cycles, and in order to shift the PPS output to start a little bit earlier or later you simply increase or decrease the oscillator frequency for a certain interval.

The advantage of this approach is also that the oscillator can continue to be used as reference time source if the reference signal temporarily fails, but the oscillator has been properly disciplined before when the signal was still available. This is usually referred to as holdover mode. The maximum holdover interval depends on:

- Accuracy requirements, i.e. the maximum allowed time offset after a given holdover interval.
- The quality and stability of the oscillator, i.e. how good the oscillator keeps its output frewquency and thus the derived time using the last recent disciplination value.
- How good the oscillator has been disciplined before when the reference signal was still available. This may depend on the systematic accuracy of the reference signal type.

Disciplined oscillators are usually part of more expensive reference time sources, whereas cheaper reference time sources come with cheaper oscillators or crystals, similar to the cheap oscillator usually installed on computer mainboards. Also, cheap oscillators often don't provide a way to discipline the output frequency.

For example, on the more expensive cards manufactured by Meinberg there is often a 10 MHz high quality oscillator which is disciplined by the incoming reference signal. The oscillator frequency is then made available as an output signal, and it drives a counter chain which generates the 1 PPS output signal.

On the cheaper cards and the USB devices manufactured by Meinberg there is no high quality oscillator which can be disciplined but a standard crystal which drives the microcontroller clock. That crystal is not even necessarily running at 10 MHz, but at a frequency required by the microcontroller, and each individual crystal on each individual device of the same type has its individual frequency offset, which is, by the way, similar to value reported in the NTP daemon's drift file. See also chapters "*Why the Undisciplined Software Clock Drifts*" and "*The NTP Drift File*"

The PPS output signal is then generated by counting the cycles of the crystal at its real frequency, e.g. if a crystal runs at 10.000002 MHz then the PPS signal starts every 10000002 cycles, and if the crystal runs at 9.999999 MHz then the PPS signal starts every 9999999 cycles. If the PPS signal needs to be shifted to occur earlier or later then one cycle is inserted or skipped in the counter chain.

8.5 Hardware PPS Considerations

8.6 Meinberg's Approach to PTP Client PCI Cards

In order to yield highest accuracy with PTP hardware timestamping of PTP network packets must be supported on **every** network node involved in the packet exchange.

Of course the LANTIME PTP grandmasters manufactured by Meinberg provide this support on their PTP network ports. However, most high performance switches usually installed in data centers are not PTP-aware. Finally there are some points to keep in mind if a PTP program is to run directly on a machine:

- There must be an implementation of the PTP daemon/protocol for the host operating system.
- To yield highest accuracy, timestamping support is required for the NIC possibly assembled onto the mainboard, the NIC driver for the host OS must be able to queue these timestamps, and a software interface must be implemented in the host OS and drivers which allows the PTP daemon to retrieve the timestamps from the driver.
- If the system time kept by the operating system provides only limited resoluton (e.g. 1..16 ms under Windows) there's hardly a chance to get the most possible accuracy out of PTP.
- Last but not least the timestamps from the NIC need to be related to the system time of the host OS, so the achievable accuracy depends on the clock source used for timestamping by the NIC, i.e. the stability of the oscillator, if it can be disciplined, or not, and whether TSU clock is derived from the same clock source as used for system timekeeping, which may depend on the mainboard or NIC chip.

For Meinberg as a manufacturer of PCI cards which are to be used on different computers with different NIC chips, and different operating systems, it would be hard to provide and maintain a general solution, so Meinberg has made a different approach with their PTP client PCI card PTP270PEX.

The PTP270PEX card provides an own LAN port which is not visible to the host PC, but is used by a small single board computer (SBC) also installed on the PCI card, running the PTP protocol stack

and disciplining a high quality oscillator which drives the on-board clock.

The advantage of this approach is that all the timing critical stuff runs on the card in a known, stable environment, and does neither depend on the hardware of the PC in which the card is installed, nor on the operating system type and version, since the complete PTP specific stuff is handled on-board the PCI card.

The on-board clock is implemented as a counter chain which also generates a 1 PPS output signal that can be compared to a 1 PPS signal from a PTP grandmaster to verify the computed accuracy.

Applications can directly read high accuracy, high resolution UTC time stamps from the on-board clock using the standard API calls provided by the Meinberg driver package for the host operating system. This works in the same way as with other PCI cards manufactured by Meinberg. So from this point of view the PTP270PEX's LAN port is similar to an antenna connector of a GPS PCI card.

9 Distributing Reference Time to Computers

If several computers shall be equipped with hardware reference time sources then the reference time signal has to be distributed to each of the devices in a way which depends on the signal type.

In order to provide several GPS cards with an antenna signal, one or more antenna diplexer(s) need to be installed, and antenna cables need to be wired to every single card. Since the GPS signal delay from the antenna to the receiver needs to be compensated to yield highest accuracy, each single GPS receiver needs to be configured individually according to the antenna cable length and the number of antenna diplexers between the antenna and the receiver. The accuracy of the on-board time of a GPS card is a few hundred nanoseconds.

If IRIG cards shall be used then a diplexer for an IRIG signal may also need to be installed, and a special cabling is required, depending on whether an unmodulated or a modulated IRIG signal is used. The accuracy which can be achieved for the on-board time is about 5 microseconds.

The advantage of PTP is that standard patch cables can be used for the connection. The PTP slave card manufactured by Meinberg has its own network interface, and the PTP protocol stack runs on a single board computer on the card. A PTP grandmaster (LANTIME M600/PTP) and a PTP-aware switch can provide several PTP slaves with the time. The achievable accuracy of the on-board time is 100 nanoseconds.

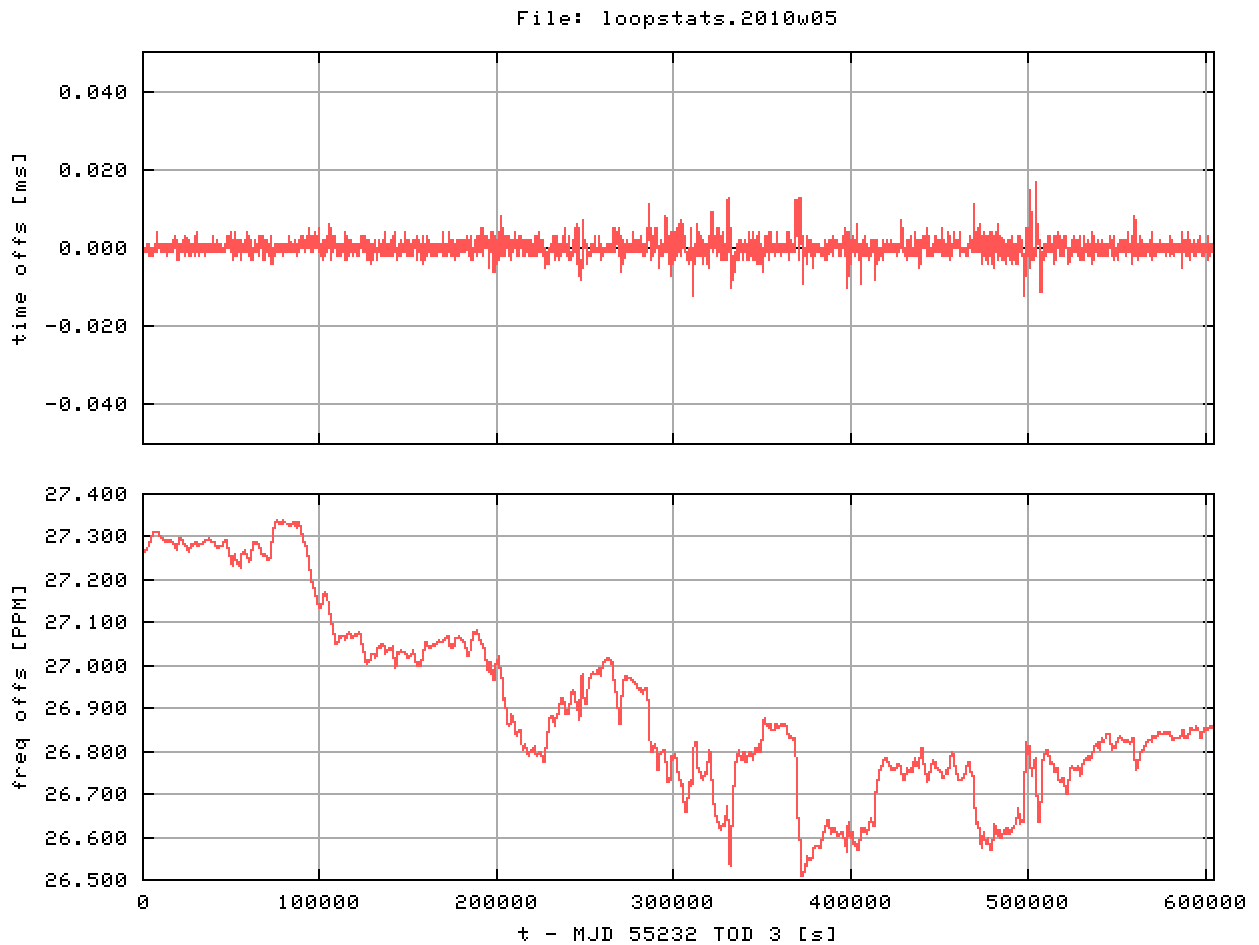
Each of the methods above make a highly accurate time available on a PCI card in a computer.

Individual applications can yield the highest accuracy if they retrieve the time directly from the card via an API call. Since all Meinberg PCI cards support the same set of API calls it makes no difference which card is used for the application, except for the slightly limited accuracy with IRIG cards.

These cards can also be used to discipline the system time. Under Linux the NTP program can be used to do this. In this case the NTP configuration has to be set up in a way that in steead of an upstream NTP server the hardware device is used as reference time source, and thus there are no network delays which need to be compensated.

The graph below shows a loopstats file generated by an NTP daemon running on a Linux server,

using a Meinberg GPS PCI card as reference time source, recorded over one week:



The “time offs” graph displays the offset between the reference time and the system time, with a scale of +/- 50 microseconds. The “freq offs” graph shows the drift compensation applied to the system clock in order to compensate the oscillator's frequency offset, in parts per million (PPM). The frequency offset variations which can be seen are due to variations of the ambient temperature. The mean value of that frequency offset is the crystal's native offset, which is a property of the individual crystal.

10 Time Synchronization Problems with Virtual Machines

10.1 General Information

Timekeeping in virtual machines (Vms) is tricky. If timer tick interrupts are virtualized then the interrupt handler in a particular VM can be delayed when the physical machine is e.g. busy with other VMs, and next time when the physical host is idle, be executed as a batch to catch up.

This means that if the time in a VM is compared to a real external reference clock in regular intervals the time difference observed in subsequent comparisons can jump back and forth even though the reference clock is stable. For example, if 1 second has expired in a VM, the time of a real clock may have gained 1.1 seconds because the timer updates in the VM were delayed, or only 0.9 seconds if the timer updates were batched to catch up.

So how accurately the time in a virtual machine can be disciplined by any synchronization software depends in general on how good the **undisciplined** time in a VM is kept, which in turn depends on the implementation of the virtualization software (i.e., the hypervisor), which has to take care that timer interrupts in each VM are scheduled accurately whenever an associated timer tick interval has expired.

So in fact the performance of time synchronization in a VM depends on the type and version of the operating system running in the VM, and on the type and version of the hypervisor software running on the physical machine, and thus scheduling the timer ticks for the Vms.

Especially, it is usually **not** appropriate to run a VM as time server for other machines.

Some virtualization systems provide their own mechanisms to discipline the time in a VM more or less accurately. Others suggest to install a software like the NTP daemon.

This section provides some links for more detailed investigation

The NTP Support Web: Known Operating System Issues
http://support.ntp.org/bin/view/Support/KnownOsIssues

Especially see: **Xen, VMware, and Other Virtual Machine Implementations**
http://support.ntp.org/bin/view/Support/KnownOsIssues#Section_9.2.2.

10.2 VMWare

VMWare Inc., Timekeeping in VMWare Virtual Machines
http://www.vmware.com/pdf/vmware_timekeeping.pdf
http://www.vmware.com/files/pdf/perf-vsphere-cpu_scheduler.pdf

It has been reported that in recent versions of VMWare ESX time synchronization using ntpd in a virtual Linux machines yields better accuracy than using the time synchronization methods provided by earlier versions of VMWare:
Timekeeping best practices for Linux guests
VMWare KB Article: 1006427, updated: Nov 5, 2011

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1006427

If the system time in a guest runs too fast then this can be due to dynamic CPU clock switching of the host system. This has been determined with VMWare versions which are now outdated. See: http://wiki.ubuntuusers.de/Archiv/VMware (German language)

10.3 XEN

It seems some versions of XEN have similar issues as VMware. See: http://xen.org/files/summit_3/Xen_HVM_SMP.pdf

10.4 Microsoft Hyper-V

Virtual PC Guy Blog: Time Synchronization in Hyper-V
http://blogs.msdn.com/b/virtual_pc_guy/archive/2010/11/19/time-synchronization-in-hyper-v.aspx

Microsoft Knowledge Base: Hyper-V Time Synchronization Doesn't Correct the System Clock in the Virtual Machine if it is more than 5 Seconds ahead of the Host Clock
http://support.microsoft.com/kb/2618634

11 Potential RTC Problems on Dual Boot Systems

Some operating systems expect the real time clock (RTC) chip on the computer's mainboard to run on local time according to a configured time zone, and also write the local time back to the RTC when the system shuts down. Windows in its different versions is a candidate which does so, presumably for backward compatibility with ancient MS-DOS systems. MS-DOS and the first Windows versions which were based on MS-DOS didn't distinguish between UTC and local time, and thus simply used the RTC's local time to initialize the DOS time. See also chapter *“Computer Local Time versus Computer UTC Time”*.

On the other hand, from nowaday's point of view it makes more sense to let the RTC on the mainboard run on UTC, let the operating system set its initial UTC system time directly from the RTC, and let the operating system compute the current local time according to the configured timezone settings.

So most Unix-like systems behave the latter way by default. This may lead to problems on dual or multiboot systems, e.g. if one operating system sets the RTC on the mainboard to the current local time when it shuts down, but a different operating system booting on the same machine expects the RTC to run on UTC.

Fortunately, some operating systems like Linux and BSD variants provide ways to tell the kernel whether the RTC runs on UTC, or on local time. Usually this can be configured during installation, or changed via some management tools which are specific to the distribution, e.g. using the YaST tool on SuSE/openSUSE **Linux**. The **FreeBSD** operating system assumes the RTC runs on local time if a file */etc/wall_cmos_clock* exists. If this file does not exist then the OS assumes the RTC runs on UTC.

If all operating systems installed on the same machine are configured for the same time zone, and all the Unix-like systems are told the RTC runs on local time, then basically everything should be working well.

Unfortunately the RTC chip does not maintain a DST status flag. So a potential problem is still if a DST changeover occurs while the computer is powered off. For example, if Windows is shut down when DST is not yet in effect then the RTC is set to standard time. If a Linux system is booted a few hours later, and DST has started in the mean time then the Linux system expects the RTC to run on DST while it in fact runs on standard time. So the initial system time is off by 1 hour after reboot.

This problem can be fixed if a good time synchronization software is used, and a time source is available at boot time which provides an unambiguous time.

More detailed considerations can also be found here:

Markus Kuhn, IBM PC Real Time Clock should run in UT [2]
http://www.cl.cam.ac.uk/~mgk25/mswish/ut-rtc.html

MSDN Blogs, Why does Windows keep your BIOS clock on local time?
http://blogs.msdn.com/oldnewthing/archive/2004/09/02/224672.aspx

12 Time Synchronization Problems Under Windows

12.1 Timer Tick Interpolation Problems

Some time synchronization software for Windows tries to interpolate the system time between 2 timer ticks using the Windows QueryPerformanceCounter (QPC) API calls. The HAL DLL shipped and installed with Windows determines which of the counters available on the mainboard or in the CPU are used to implement the QPC API, for example the ACPI power management timer (PMTIMER) or the high precision event timer (HPET) implemented in the chipset, or the Time Stamp Counter (TSC) registers provided by modern CPUs of the Intel architecture.

There are multicore CPU types out there where the TSCs in the different cores are not synchronized to each other, so if the time interpolation program is executed on different CPU cores the timestamps can't be related properly.

Also, in some CPU types the TSC clock is derived from the CPU clock, and if the CPU clock is temporarily reduced for power saving then the TSC suddenly increments at a lower rate than usually, which also drives the time interpolation code nuts.

Possible workarounds: Disable Power Saving mechanisms (Intel SpeedStep, AMD Cool'n'Quit, etc.) in the PC BIOS setup, or force Windows to use the ACPI power management timer instead of the TSCs (Windows boot parameter */USEPMTIMER*). See Microsoft support:
Programs that use the QueryPerformanceCounter function may perform poorly in Windows Server 2000, in Windows Server 2003, and in Windows XP
http://support.microsoft.com/kb/895980/en-us

This usually may affect Windows versions up to Windows XP and Server 2003. Newer versions should detect potential problems correctly and work around it automatically.

12.2 Latency Problems Affecting the Windows System Time

Latency problems usually occur if a driver or some other kernel-mode program has not been coded properly, so timer tick interrupts can get lost, other programs waiting for a given timer event are called too late, etc. This can mess up the timekeeping of the operating system itself, but it can also prevent time synchronization software from working properly.

Latency problems have been detected with an ataport.sys driver shipped with Windows Server 2008 R2. This has been detected by using two latency checkers:

This one does not need to be installed, but provides just the latency measurement without assigning which process is to blame:
http://www.thesycon.de/deu/latency_check.shtml

This one gives a little more detail, but requires a full Windows install:
http://www.resplendence.com/latencymon

12.3 Small System Time Adjustments May Be Lost

A bug in Windows Vista and newer is that some Windows versions don't apply small time adjustments at all. For example, if NTP applies an adjustment less than 16 ticks to the Windows time this is simply ignored by Windows. However, NTP expects the adjustment to have some effect, but if there is no effect then the next time comparison yields a much larger difference than expected, and thus causes another adjustment which is probably larger than necessary. As a consequence this can cause large swings in the time adjustment values, and the time offset doesn't settle and converge towards a low value.

The current developer version of the NTP package contains a workaround for this Windows bug. The report and fix are discussed here:
https://bugs.ntp.org/show_bug.cgi?id=2328

The problem is also explained on the Microsoft support page:
SetSystemTimeAdjustment May Loose Adjustments Less than 16
http://support.microsoft.com/kb/2537623

Even though the MS report only mentions Windows 7, the Windows Server 2008 kernel is similar to Windows 7 and has probably the same bug.

An NTP developer version which includes a workaround for this bug can be found here:
http://support.ntp.org/people/burnicki

Alternatively there are even newer precompiled version of NTP for Windows available on the internet which can be installed in the same way as described here.

Stop the NTP service, then unzip the ZIP archive with the new binaries and copy all extracted files over the files in your NTP installation directory (e.g. C:\Program Files (x86)\NTP\bin\). Finally restart the NTP service.

These developer versions have greatly improved the resulting accuracy on Windows 7 and Windows Server 2008 installations, and have not caused any limitations ar drawbacks.

12.4 Polling Intervals With NTP For Windows

Under Windows all upstream servers should be configureed with lines reading:

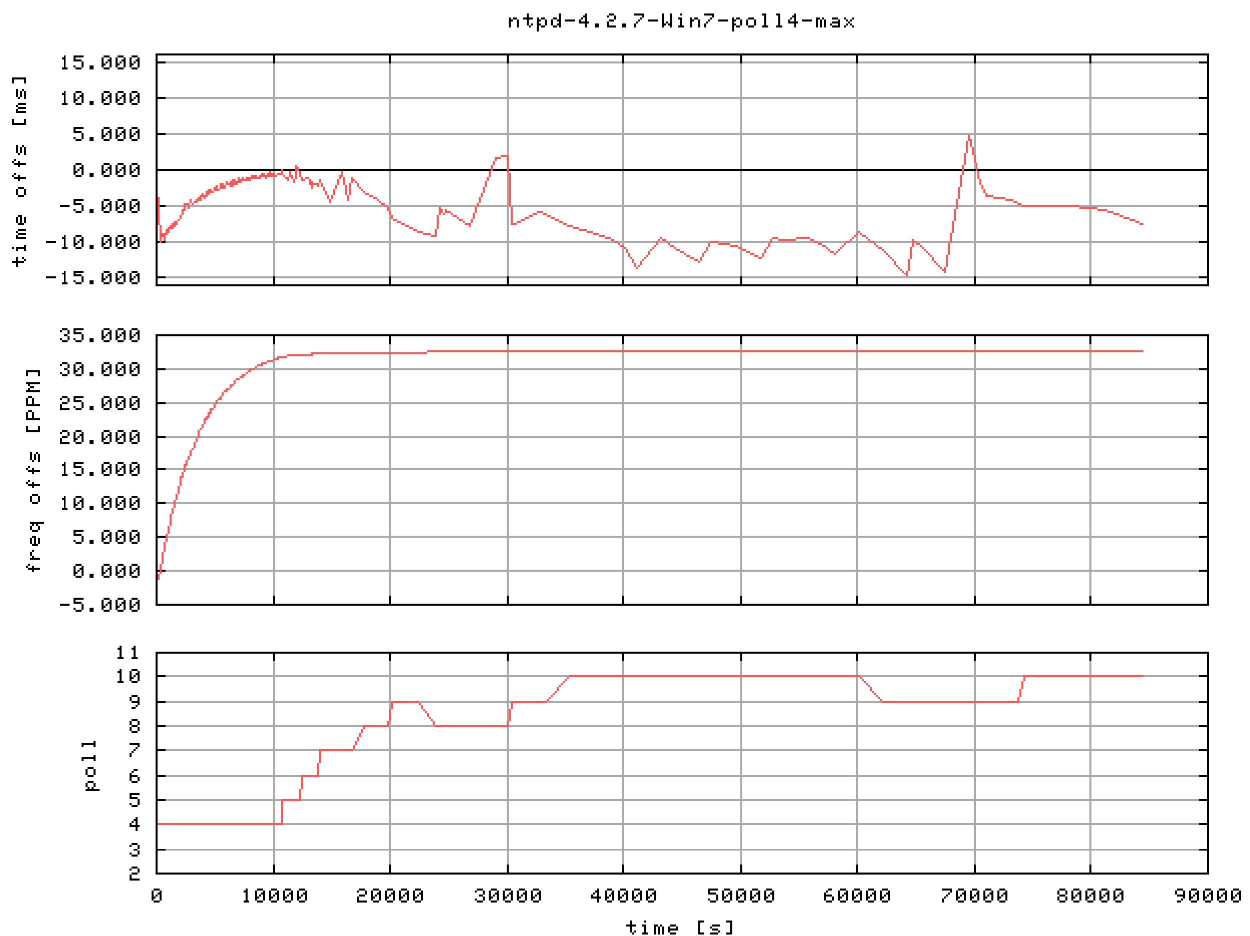
server aa.bb.cc.dd iburst minpoll 6 maxpoll 6

where *aa.bb.cc.dd* has to be replaced with the host name or IP address of your NTP server.

Generally a polling interval as short as possible should be used under Windows to let ntpd apply adjustments quickly, **but**:

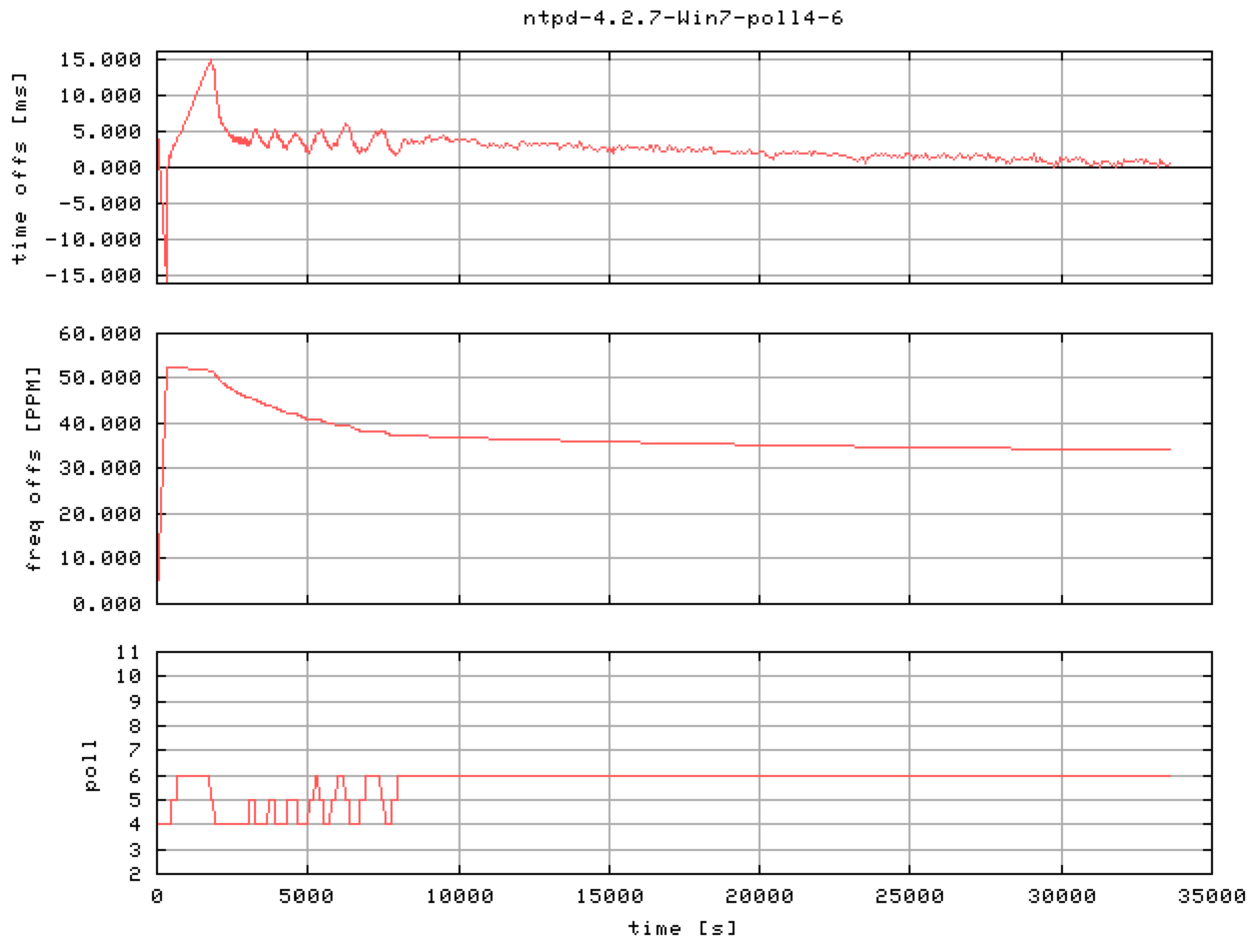
Polling intervals below 6 with the developer version should not be used under Windows since this prevents the workaround mentioned in the previous chapter from working correctly as discussed in the NTP bug report 2328.

Also, higher polling intervals can cause problems under Windows. See:
NTP Bug 2341 - ntpd fails to keep up with clock drift at poll > 7
http://bugs.ntp.org/show_bug.cgi?id=2341



The graph above shows that the time offset first starts to converge as long as the poll interval is short (2^6 s or less) but stops to converge if the poing interval ramps up to 2^{10} s.

The next graph shows that the time offset continues to converge if the polling interval is clamped to 2^6 s:



The graph above also shows that the time offset first converges worse as long as the polling interval is below 2^6 s, as explained in the bug report for NTP bug 2328:
https://bugs.ntp.org/show_bug.cgi?id=2328

So the advice is to use "minpoll 6 maxpoll 6" as indicated in the example above.

12.5 Possible Problems in a Windows Active Directory Domain

If the time in a Windows Active Directory Domain is to be synchronized then it often **not** the preferred solution to install the NTP software package on a PDC, eventually with a hardware reference clock like a GPS receiver or a PCI card. Usually it is better to set up a different server as NTP timeserver and then simply configure the domain controller to synchronize to the external NTP server.

Here are some reasons for this:

- If w32time runs on the PDC then w32time marks the PDC as authoritative time source for the domain, so domain clients can synchronize to it.
- Depending on the w32time version and configuration, it passes time only to its clients if it is synchronized to an upstream time source.
- Even if you have a PCI card plus driver installed, the w32time service is not aware that the system time is adjusted by this driver, so it may assume the system time is not synchronized. Instead, w32time may try to synchronize to some default NTP server, e.g. time.windows.com, and thus work against the PCI card's driver.
- There are some registry setting which should be able to tell w32time the system time is already synchronized by some other service, but it has been found this often does not work reliably. Either the w32time service on the domain controller did not pass the time to its clients at all, or it suddenly stopped doing so after a certain period of time, for example exactly after 1 day of operation.
- The NTP service, on the other hand, can easily be configured not to change the system time but distribute it on the network. However, the NTP service on a PDC does not mark the PDC as authoritative time source for the domain, so clients will not detect it as reliable time source, so NTP may also have to be installed and configured on all the client machines.

As a conclusion and best practice you can say the best solution is to install the PCI card plus its driver plus the NTP packet on a different machine than the PDC, then configure the PDC's w32time service to use that machine as "internet time server", and thus synchronize to that machine via NTP.

In a mixed environment the preferred solution is to set up e.g. a Linux machine as NTP server since it can achieve better accuracy than Windows, but in a pure Windows environment any Windows machine can do the job as NTP server.

In case of an external NTP server w32time can be running as usual on the PDC, has a reliable time source to synchronize to, and the domain clients find their authoritative time source (the PDC) automatically.

All non-domain members can also synchronize directly to the external NTP server.

Bibliography

Ron Bean; The Clock Mini-HOWTO: How Linux Keeps Track of Time
http://tldp.org/HOWTO/Clock-2.html
Markus G. Kuhn; IBM PC Real Time Clock should run in UT
http://www.cl.cam.ac.uk/~mgk25/mswish/ut-rtc.html

Index