Part II: Adjustment of System Time

Arno Lentfer, June 2012

Last Update: Version 3.00, March 2018

Windows provides the following simple tools to manage and monitor system time adjustments: The Internet Time GUI and the console application w32tm.exe. These tools are sufficient to obtain an initial rough estimate of the performance of the Windows internet time synchronization.

1. The Internet Time GUI

Synchronization to an internet time server is accomplished directly from the user interface. Windows Vista, Windows 7 and Windows 8 provide the Internet Time Settings window and Windows XP provides the Internet Time tab in the Date and Time Properties window:

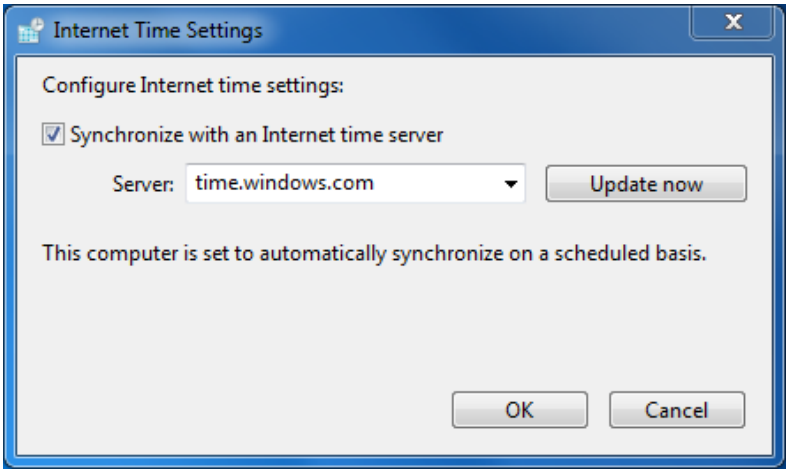

Fig. 1.1: Internet Time Settings window of Windows Vista and higher.

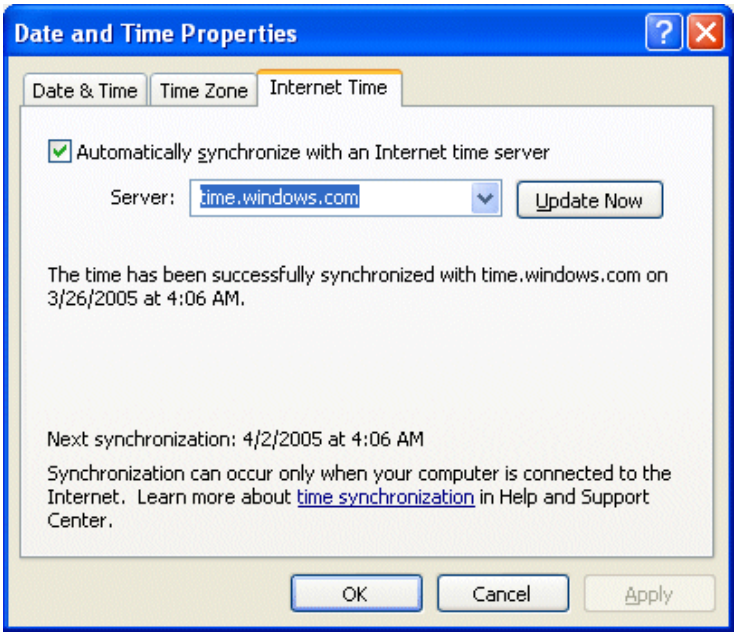

Fig. 1.2: Date and Time Properties windows of Windows XP/Server 2003.

An internet time provider can be chosen from a list or a new NTP server address can be added to the list. It is also possible to add an IP address to the list. Adding an IP address may be advisable when the name represents a pool of servers and the server needs to be explicitly indicated.

The common "Update Now" button will attempt to synchronize the system time to the time server. This allows synchronization to take place or it becomes active upon confirmation. Note: The message "...has been successfully synchronized..." does not necessarily mean that synchronization has finished. It could also mean that a synchronization process was successfully started. Such processes can last for many hours.

2. w32tm.exe

In order to verify the result or progress of the synchronization, another tool has to be run in parallel. The console application w32tm.exe allows monitoring of the offset of the local time to the time of an internet time server.

The easiest way to do this is from a console window with the following set of parameters:

```
w32tm /stripchart /computer:time.windows.com /period:120
```

As a result, the system time and its offset to the time server are dumped to the console every 120 seconds:

```
Tracking time.windows.com [65.55.21.14.123]
08:38:57 d:+00.0417301s o:+00.1024506s  [        *        ]
08:40:57 d:+00.0418632s o:+00.1037897s  [        *        ]
08:42:58 d:+00.0419165s o:+00.1015612s  [        *        ]
08:44:58 d:+00.0417048s o:+00.0985075s  [        *        ]
08:46:58 d:+00.0419394s o:+00.0942827s  [        *        ]
08:48:58 d:+00.0419296s o:+00.0913788s  [        *        ]
08:50:58 d:+00.0418867s o:+00.0883421s  [        *        ]
```

Each line consists of the local time (08:38:57), an internal delay (time difference between the udp package received and udp package sent on the server side, i.e., d:+00.0419394s), the actual offset between the local time and the server time (o:+00.1024506s) and a very basic stripchart of the offset.

The first output line of w32tm will also resolve the name of the time server (time.windows.com) to an IP (UDP port 123 is reserved for NTP). This is important because time.windows.com does not refer to a single server but rather to a pool of servers; therefore, consecutive attempts to synchronize to it may use different physical servers. However, w32tm resolves the IP of the server currently in use with w32tm. This IP can also be chosen as a server for the synchronization. For example, one of the addresses of the time.windows.com pool is 65.55.21.14. The best proof of quality is obtained when the IP address in the internet time GUI described above and the same IP address with the w32tm command are used:

```
w32tm /stripchart /computer:65.55.21.14 /period:120
```

3. Results

The results obtained with w32tm are difficult to interpret. When the offset in time is large (i.e., several seconds), synchronization of the system time seems to happen in one step. In these cases, the remaining offset is typically larger than a few milliseconds. However, when the offset is less than a few seconds, an algorithm gently adjusts the offset in small steps. This procedure can take many hours.

It turns out that obtaining detailed insight into this adjustment algorithm by using w32tm is difficult. A more in-depth investigation may uncover the cause of the behavior observed, however, this requires additional software.

4. Discussion

Applying the scheme described above frequently gives very dissatisfying results. Sometimes the synchronization results in a time offset that is worse than the offset prior to synchronization. In particular, Windows Vista and Windows 7 show strange behavior, e.g., seemingly never-ending adjustments to huge offsets.

A piece of software is necessary to find out the secret of the adjustment algorithm. Actual system time adjustment parameters can be obtained by a call to the function GetSystemTimeAdjustment because Windows performs the system time adjustment through calls to the function SetSystemTimeAdjustment.

```
BOOL WINAPI GetSystemTimeAdjustment(
  OUT PDWORD lpTimeAdjustment,
  OUT PDWORD lpTimeIncrement,
  OUT PBOOL lpTimeAdjustmentDisabled);
```

MSDN: "For each *lpTimeIncrement* period of time that actually passes, *lpTimeAdjustment* will be added to the time of day." Assuming this rule, the adjustment gain can be calculated:

gain = (lpTimeAdjustment - lpTimeIncrement)/ lpTimeIncrement

A simple program can call GetSystemTimeAdjustment frequently while a system time adjustment is active and evaluate the gains for individual values of *lpTimeAdjustment*. The function SetSystemTimeAdjustment allows to initiate and control a system time adjustment:

```
BOOL WINAPI SetSystemTimeAdjustment(
  IN DWORD dwTimeAdjustment,
  IN BOOL bTimeAdjustmentDisabled);
```

System time adjustments occur when *bTimeAdjustmentDisabled* is set to FALSE and *dwTimeAdjustment* is set to some meaningful value. Unfortunately, the influence of the values of *dwTimeAdjustment* depends on the Windows version: The MSDN description of the SetSystemTimeAdjustment function contains the note: *"Currently, Windows Vista and Windows 7 machines will lose any time adjustments set less than 16."* Note: Windows 8 is not mentioned here, the related knowledge base article KB2537623 also does not mention Windows 8.

The update scheme of the system time and also the scheme of system time adjustments depends on the presence of a High Precision Event Timer [HPET]. Intel specifies [hpetspec.dpf]: *"An existing HPET does not replace the RTC Time of Day, the RTC Alarm,*

and the RTC CMOS functionality. The HPET architecture supplements/replaces only the RTC Periodic Interrupt function." The RTC (Real Time Clock) Periodic Interrupt function used to be the heartbeat of the system time update. However, an existing HPET will replace this functionality and remove the system time update activity from the RTC periodic interrupt function. Those systems can typically be identified by a specific value of the update period *lpTimeIncrement*: 156001. HPET and RTC are driven by different hardware. Therefore they are neither synchronized nor are they in phase by default; additionally they may show specific drifts. More information about the evolution of the HPET architecture is given in "Guidelines For Providing Multimedia Timer Support" [MSDN]. Newer systems may provide hardware with an invariant Time Stamp Counter (TSC) as described in section 17.13 of "Intel® 64 and IA-32 Architectures, Software Developer's Manual". Windows has a clear preference about what hardware resource is to be used for timekeeping. When suitable TSC characteristics are obtained, Windows uses the TSC for timekeeping. If the TSC is not suitable, Windows uses the HPET when available, and if that is not available or disabled in BIOS Windows uses the ACPI PM timer ("MSDN: Acquiring high-resolution time stamps.").

It was already shown in section 2.3 of "*Microsecond Resolution Time Services for Windows*" that the Windows system timing cannot be assumed to show a fixed behavior. The evolution of Windows with newly introduced limitations (*... will lose any time adjustments set less than 16.*) and emerging new hardware results in a big variety of schemes for system time adjustments. A few relevant combinations are diagnosed and described here.

4.1. Windows XP and Windows Server 2003: The Classical Case

A call to GetSystemTimeAdjustment reveals a value of 156250 for *lpTimeIncrement* on most platforms running Windows XP or its server variants (Some specific hardware may return other values e.g. 100144). Note: A value of 156250 represents 15.625 ms, an RTC Periodic Interrupt at 64 Hz. This is a very common hardware fingerprint.

Using the function SetSystemTimeAdjustment with *dwTimeAdjustment* = 156250 and *bTimeAdjustmentDisabled* = FALSE shall initiate a system time adjustment. However, according to the gain equation described in 4. no adjustment shall take place, the gain shall be zero, but the adjustment shall be active with *lpTimeAdjustmentDisabled* = FALSE.

Setting *dwTimeAdjustment* to any number different from *lpTimeIncrement* shall result in a system time adjustment. Example: *lpTimeIncrement* = 156250 and *dwTimeAdjustment* = 156257. The system time will advance by 15.6257 ms every 15.6250 ms, the system time will gain 0.0448 ms/s (7/156250). This way the gains are predictable, a small list shows the obtained gains at the neighborhood of 156250 at *dwTimeAdjustment* from 156255 to 156248:

```
156255:  0.032000 ms/s = (156255 - 156250)/156250
156254:  0.025600 ms/s = (156254 - 156250)/156250
156253:  0.019200 ms/s = (156253 - 156250)/156250
156252:  0.012800 ms/s = (156252 - 156250)/156250
156251:  0.006400 ms/s = (156251 - 156250)/156250
156250:  0.000000 ms/s = (156250 - 156250)/156250
156249: -0.006400 ms/s = (156249 - 156250)/156250
156248: -0.012800 ms/s = (156248 - 156250)/156250
```

These numbers are captured on true hardware. The adjustment gain is zero with *dwTimeAdjustment* = 156250. The smallest available adjustment on such a platform is 6.4 µs/s (positive and negative).

A similar scan was carried out on hardware reporting *lpTimeIncrement* = 100144 (*dwTimeAdjustment* = 100146 to 100142):

```
100146:  0.01997124 ms/s  = (100146 - 100144)/100144
100145:  0.00998562 ms/s  = (100145 - 100144)/100144
100144:  0.00000000 ms/s  = (100144 - 100144)/100144
100143: -0.00998562 ms/s  = (100143 - 100144)/100144
100142: -0.01997124 ms/s  = (100142 - 100144)/100144
```

This hardware consistently follows the gain equation provided by the MSDN description. However, the smallest adjustment gain on this hardware is almost 10 µs/s.

Windows XP and Windows Server 2003 do not support a hardware HPET. These Windows versions may use Programmable Interrupt Timers (PIT), Real Time Clocks (RTC), the processors Time Stamp Counter (TSC), and Power Management Timer (PMTIMER) to mimic what is later done by the High Precision Event Timer (HPET). These Windows versions increment the system time at a fixed period every *lpTimeIncrement*. This period does not depend on settings of the timer resolution by means of the timeBeginPeriod() function. This is easiest confirmed by polling system file time transitions over a longer period of time with different settings of timeBeginPeriod(). As a result, the granularity of the system time is typically in the range of 10 ms to 20 ms.

4.2. Windows Vista, Windows 7, Windows 8, 8.1 and Windows 10

Windows VISTA introduced HPET support. It has been the first public Windows version decoupling the system time update and the system time adjustment from the RTC Periodic Interrupt function or the ACPI PM timer in case of existing HPET hardware. This was a big step towards higher timing accuracy. However, it also caused some inconsistency with a remarkable drawback for Windows VISTA and Windows 7 (KB2537623) persisting until now. Windows Vista also introduced the influence of the multimedia timer resolution (set by timeBeginPeriod) to the update period of the system time: The system time is updated at a period of *ActualResolution* returned by the function NtQueryTimerResolution.

The following list of system time gains vs. *dwTimeAdjustment* (156154 to 156330) was taken with Windows Vista on a platform without HPET/TSC support (*lpTimeIncrement* = 156250):

```
156154 to 156169 [16 element(s)] gain -0.5120328 ms/s.
156170 to 156185 [16 element(s)] gain -0.4096262 ms/s.
156186 to 156201 [16 element(s)] gain -0.3072197 ms/s.
156202 to 156217 [16 element(s)] gain -0.2048131 ms/s.
156218 to 156233 [16 element(s)] gain -0.1024066 ms/s.
156234 to 156250 [17 element(s)] gain +0.0000000 ms/s.
156251 to 156266 [16 element(s)] gain +0.1024066 ms/s.
156267 to 156282 [16 element(s)] gain +0.2048131 ms/s.
156283 to 156298 [16 element(s)] gain +0.3072197 ms/s.
156299 to 156314 [16 element(s)] gain +0.4096262 ms/s.
156315 to 156330 [16 element(s)] gain +0.5120328 ms/s.
```

This list discloses some information contained in *"... will lose any time adjustments set less than 16..."*. It seems that it is not losing time adjustments with values less than 16, but SetSystemTimeAdjustment ignores the lower 4 bits of *dwTimeAdjustment*. The

obtained gain is the same for all *dwTimeAdjustment* values in one group. The group size is 16. Only the group ranging from 156234 to 156250 has 17 members. It is yet unclear why the scheme shows this exception. However, the gain equation used for the gain calculation obviously does not apply here. Therefore, MSDN: "For each *lpTimeIncrement* period of time that actually passes, *lpTimeAdjustment* will be added to the time of day" becomes incorrect for this configuration. Exception: Gain is zero at *dwTimeAdjustment* = *lpTimeIncrement*.

The next list is taken with Windows Vista on a platform with HPET/TSC support (*dwTimeAdjustment:* 155908 to 156079, *lpTimeIncrement* = 156001):

```
155908 to 155922 [15 element(s)] gain -0.5000000 ms/s.
155923 to 155938 [16 element(s)] gain -0.4000000 ms/s.
155939 to 155954 [16 element(s)] gain -0.3000000 ms/s.
155955 to 155969 [15 element(s)] gain -0.2000000 ms/s.
155970 to 155985 [16 element(s)] gain -0.1000000 ms/s.
155986 to 156001 [16 element(s)] gain +0.0000000 ms/s.
156002 to 156016 [15 element(s)] gain +0.1000000 ms/s.
156017 to 156032 [16 element(s)] gain +0.2000000 ms/s.
156033 to 156047 [15 element(s)] gain +0.3000000 ms/s.
156048 to 156063 [16 element(s)] gain +0.4000000 ms/s.
156064 to 156079 [16 element(s)] gain +0.5000000 ms/s.
```

The periodic interrupt increments the system time and performs the system time adjustment. However, the "lost 4 bits" idea becomes questionable one more time. Groups have either 15 or 16 elements.

The minimum selectable adjustment gain appears to be coarse on Windows VISTA. TSC, HPET, or PM timer configurations show a minimum gain of approx. +/- 0.1 ms/s.

Windows 7 and Windows Server 2008 R2 introduced Timer Coalescing (more detailed: TimerCoal.docx) to *"...improve the efficiency of periodic software activity by expiring multiple distinct software timers at the same time..."*. This portion of software shifts interrupts into groups of interrupts. A requested interrupt is accompanied by a tolerance to tell the OS by how much it is allowed to shift the interrupt in time. This may affect the update of the system time and has to be diagnosed carefully. Windows 7 does not update the system time by fixed increments.

Capturing the adjustment gain on a Windows 7 platform with constant TSC support results in the following list (*dwTimeAdjustment:* 155908 to 156079 *lpTimeIncrement* = 156001):

```
155908 to 155922 [15 element(s)] gain -0.5571681 ms/s.
155923 to 155938 [16 element(s)] gain -0.4571738 ms/s.
155939 to 155954 [16 element(s)] gain -0.3571796 ms/s.
155955 to 155969 [15 element(s)] gain -0.2571853 ms/s.
155970 to 155985 [16 element(s)] gain -0.1571910 ms/s.
155986 to 156001 [16 element(s)] gain -0.0571967 ms/s.
156002 to 156016 [15 element(s)] gain +0.0427976 ms/s.
156017 to 156032 [16 element(s)] gain +0.1427918 ms/s.
156033 to 156047 [15 element(s)] gain +0.2427861 ms/s.
156048 to 156063 [16 element(s)] gain +0.3427804 ms/s.
156064 to 156079 [16 element(s)] gain +0.4427747 ms/s.
```

Three important results can be drawn from the list above:

- The gain at *dwTimeAdjustment* = *lpTimeIncrement* is not 0 ms/s.
- The gain distribution is asymmetric. The gain steps are in the order of 0.1 ms/s, but the smallest positive gain differs from the smallest negative gain.

- The smallest available adjustment is 42 µs/s in the positive direction and -57 µs/s in the negative direction. This does not appear to be a good resolution when compared to Windows XP and Windows Server 2003.

This behavior raises the question of whether a specific gain for a specific value of *dwTimeAdjustment* remains constant over time. Careful evaluation of this matter has not confirmed any variation of the gain (added advancement of the system time) when a constant value of *dwTimeAdjustment* is applied. Therefore, it remains difficult to predict the adjustment gain for values of *dwTimeAdjustment* for systems affected by this scheme (Windows Vista and Windows 7 with HPET/TSC support). "For each *lpTimeIncrement* period of time that actually passes, *dwTimeAdjustment* will be added to the time of day." In this regard, [MSDN]'s claim turns out to be wrong on Windows 7 too. Note: This specific asymmetry occurs with the systems interrupt period set the minimum by means of e.g. timeBeginPeriod(wPeriodMin).

All software packages using *SetSystemTimeAdjustment* are in serious danger of relying on predictable gains. It should also be noted that there is no *dwTimeAdjustment* setting for a gain of 0.0 ms/s. It was shown in section 4.1 that earlier versions of Windows had a much more predictable scheme. The scheme observed on Windows VISTA and Windows 7 requires the software to calibrate itself to the appropriate gain for values of *dwTimeAdjustment* because it cannot be easily evaluated by the given values of *lpTimeIncrement* and *lpTimeAdjustment*.

The system time synchronization routines of these newer Windows versions do not seem to take these facts into account. A typical synchronization to an internet time server uses all bits for setting the values of *dwTimeAdjustment*. This can be easily monitored through frequent use of *GetSystemTimeAdjustment*. Furthermore, these tools expect the lower 4 bits to be taken into account by the system. Windows calculates a correction scheme ahead of the actual adjustment based on the offset to the network time. Unfortunately, the gains are not set as expected and the predicted scheme messes up the adjustment/synchronization, which results in the synchronization being completely off. This is accompanied by the fact that there is no monitoring of the internet time provider while the system time adjustment progresses. Such an adjustment can run for hours and a big deviation may appear with wrong gain estimates resulting from the synchronization algorithm. Finally, at some point the deviation will be several seconds and the next synchronization will only set the local time to the network time without applying the function *SetSystemTimeAdjustment*.

Windows 8 has finally fixed this mishap. This list has been captured on a Windows 8 system with constant TSC support:

```
155995 to 155995 [1 element(s)] gain -0.0377952 ms/s.
155996 to 155996 [1 element(s)] gain -0.0316960 ms/s.
155997 to 155997 [1 element(s)] gain -0.0255968 ms/s.
155998 to 155998 [1 element(s)] gain -0.0185976 ms/s.
155999 to 155999 [1 element(s)] gain -0.0127984 ms/s.
156000 to 156000 [1 element(s)] gain -0.0062992 ms/s.
156001 to 156001 [1 element(s)] gain +0.0003000 ms/s.
156002 to 156002 [1 element(s)] gain +0.0073991 ms/s.
156003 to 156003 [1 element(s)] gain +0.0130983 ms/s.
156004 to 156004 [1 element(s)] gain +0.0200975 ms/s.
156005 to 156005 [1 element(s)] gain +0.0257967 ms/s.
156006 to 156006 [1 element(s)] gain +0.0327959 ms/s.
156007 to 156007 [1 element(s)] gain +0.0389951 ms/s.
```

The missing resolution for the value of *dwTimeAdjustment* is gone, each value has its own gain and the gain is close to the predicted gain (Example: 156003: (156003 - 156001)/156001 = 0.0128 ms/s). The deviations of gains shown in this list are a result of the changes in Windows 8 timekeeping. Windows 8 does not increment the system time by constant increments, it rather applies a variety of increments to achieve a desired *mean increment*. As a consequence, the above measurement would have to be taken over many more periods to show results with less deviation. However, it is very obvious that the described adjustment scheme if fulfilled with Windows 8.

As of Windows 8.1, timekeeping has again undergone some modification. The same hardware now reports 156250 for *lpTimeIncrement*. The list of gains appears as follows:

```
156244 to 156244 [1 element(s)] gain -0.0382037 ms/s.
156245 to 156245 [1 element(s)] gain -0.0316040 ms/s.
156246 to 156246 [1 element(s)] gain -0.0259043 ms/s.
156247 to 156247 [1 element(s)] gain -0.0184048 ms/s.
156248 to 156248 [1 element(s)] gain -0.0124051 ms/s.
156249 to 156249 [1 element(s)] gain -0.0066054 ms/s.
156250 to 156250 [1 element(s)] gain +0.0004942 ms/s.
156251 to 156251 [1 element(s)] gain +0.0060939 ms/s.
156252 to 156252 [1 element(s)] gain +0.0135934 ms/s.
156253 to 156253 [1 element(s)] gain +0.0188931 ms/s.
156254 to 156254 [1 element(s)] gain +0.0253928 ms/s.
156255 to 156255 [1 element(s)] gain +0.0324924 ms/s.
156256 to 156256 [1 element(s)] gain +0.0385920 ms/s.
```

Windows 8.1 has finally returned to the original Windows heartbeat of 64 Hz (1/15.625 ms). Each value of *dwTimeAdjustment* produces an individual gain and the result follows the documentation.

When operating on a platform with invariant TSC, the scheme looks like this:

```
156244 to 156244 [1 element(s)] gain -0.0384000 ms/s.
156245 to 156245 [1 element(s)] gain -0.0320000 ms/s.
156246 to 156246 [1 element(s)] gain -0.0256000 ms/s.
156247 to 156247 [1 element(s)] gain -0.0192000 ms/s.
156248 to 156248 [1 element(s)] gain -0.0128000 ms/s.
156249 to 156249 [1 element(s)] gain -0.0064000 ms/s.
156250 to 156250 [1 element(s)] gain +0.0000000 ms/s.
156251 to 156251 [1 element(s)] gain +0.0064000 ms/s.
156252 to 156252 [1 element(s)] gain +0.0128000 ms/s.
156253 to 156253 [1 element(s)] gain +0.0192000 ms/s.
156254 to 156254 [1 element(s)] gain +0.0256000 ms/s.
156255 to 156255 [1 element(s)] gain +0.0320000 ms/s.
156256 to 156256 [1 element(s)] gain +0.0384000 ms/s.
```

This looks very much like the classical Windows XP adjustment gain scheme. It matches the formula *gain = (lpTimeAdjustment - lpTimeIncrement)/ lpTimeIncrement.*

The system time adjustment will take care that the system time will progress by *TimeAdjustment* during *TimeIncrement*. This effectively happened with Windows XP. Since Windows 8 (on specific hardware also since Windows 7) this process may also appear as a progress in smaller steps, depending on the setting of the timer resolution. When the timer resolution is set to maximum resolution (see section 2.1. of *Microsecond Resolution Time Services for Windows)*, the obtained increments are in the same order of magnitude as the timer resolution. However, Windows 8 and Windows 8.1 maintain the average progress of *TimeAdjustment* during *TimeIncrement*.

This scheme also applies for Windows 10.

Additional information:

MSDN: "*The W32Time service cannot reliably maintain sync time to the range of 1 to 2 seconds. Such tolerances are outside the design specification of the W32Time service*." [Support boundary to configure the Windows Time service for high accuracy environments]

MSDN: "*If the time difference between the local clock and the selected accurate time sample (also called the time skew) is too large to correct by adjusting the local clock rate, the time service sets the local clock to the correct time.*" [How the Windows Time Service Works]

4.3. Monitoring an NTP time provider

A much more detailed view of the system time adjustment can be obtained when the local time is compared to a precise remote time while the system time adjustment is active. The accuracy of w32tm.exe is simply too poor to extract meaningful results. Also, the accuracy of time.windows.com is unsatisfactory.

In order to facilitate a closer look at the problems described above, an NTP (Network Time Protocol) client was added to the time services and the user interface was extended by an NTP Offset tab. This allows to see how the local time progresses against a reference time.

The calibrated performance counter frequency receives an additional correction when a system time adjustment is active. The system time adjustment forces the local time to advance slower or faster, thus the performance counter frequency has to be corrected in a way that takes the modified duration of the "second" during the adjustment into account (see section 2.1.3. of Microsecond Resolution Time Services for Windows). Consequently, an applied system time adjustment becomes visible in the "Calibrated Performance Counter Frequency Offset" tab. As of version 1.2, the calibrated performance counter frequency offset is given in ppm. It is referenced to the value given by QueryPerformanceFrequency() and scaled to show deviation in parts per million. This corresponds to µs/s. This way applied system time adjustment gains will directly show in the plot with real numbers.

The user interface now also provides two checkboxes. When the NTP checkbox is checked, NTP monitoring is activated. The "Autoadjust" checkbox enables permanent synchronization of the local time to a network time:

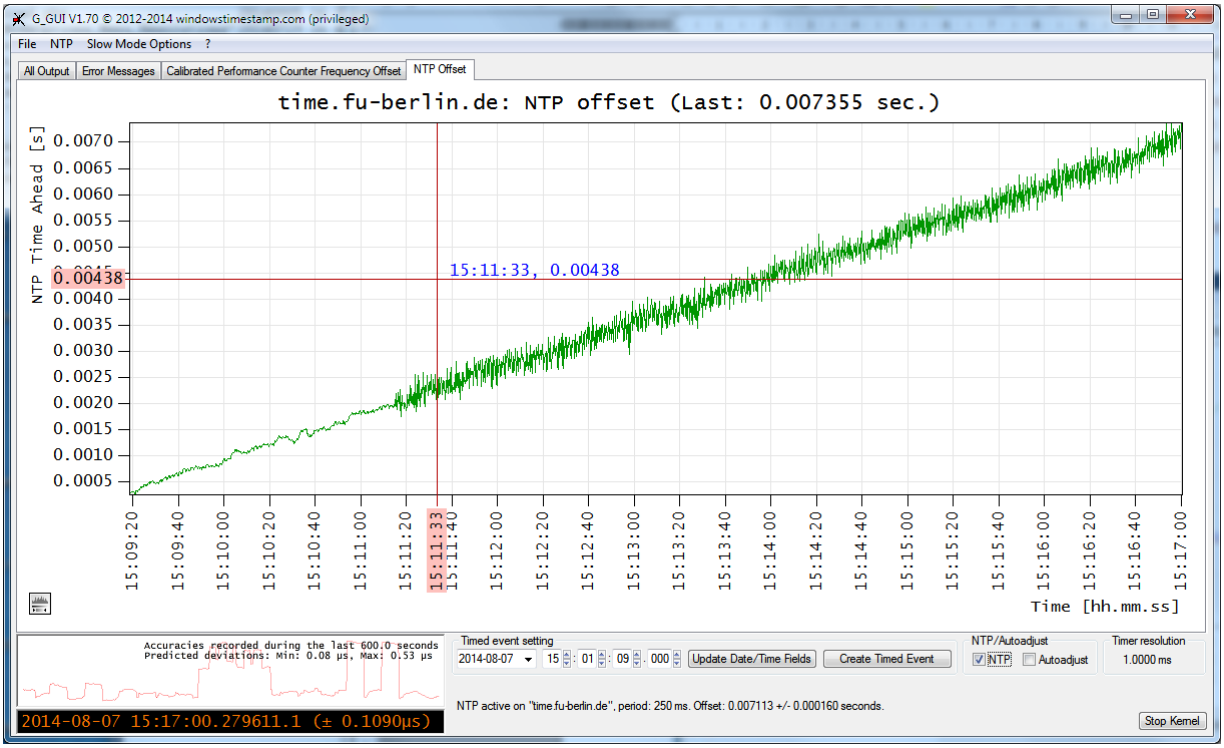

Fig. 4.3.1: GUI V1.70 with NTP Offset tab, NTP and Autoadjust checkboxes, and NTP/Autoadjust status lines.

The NTP status and the current offset to the network time are reported at the bottom in the NTP status line. Another status line contains information about the automatic adjustment (see section 4.4. for more information on automatic adjustment).

The following two plots were captured when the a system time adjustment was triggered by Windows XP:

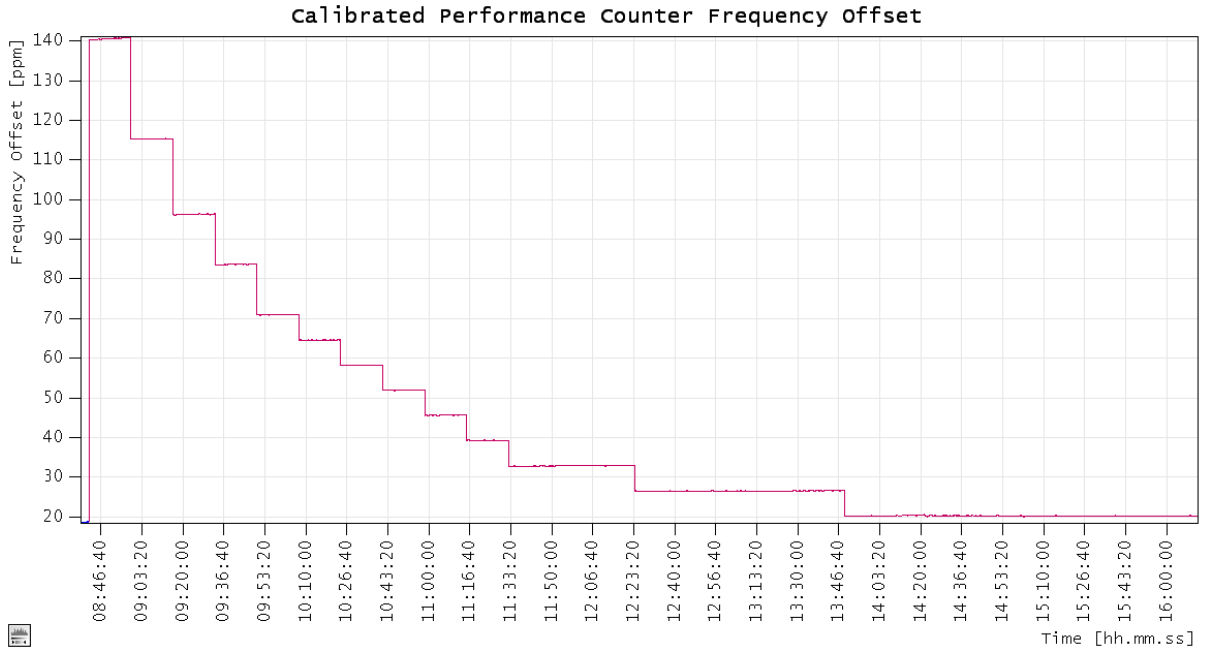

Fig. 4.3.2: System time adjustment mapped to performance counter frequency (Windows XP).

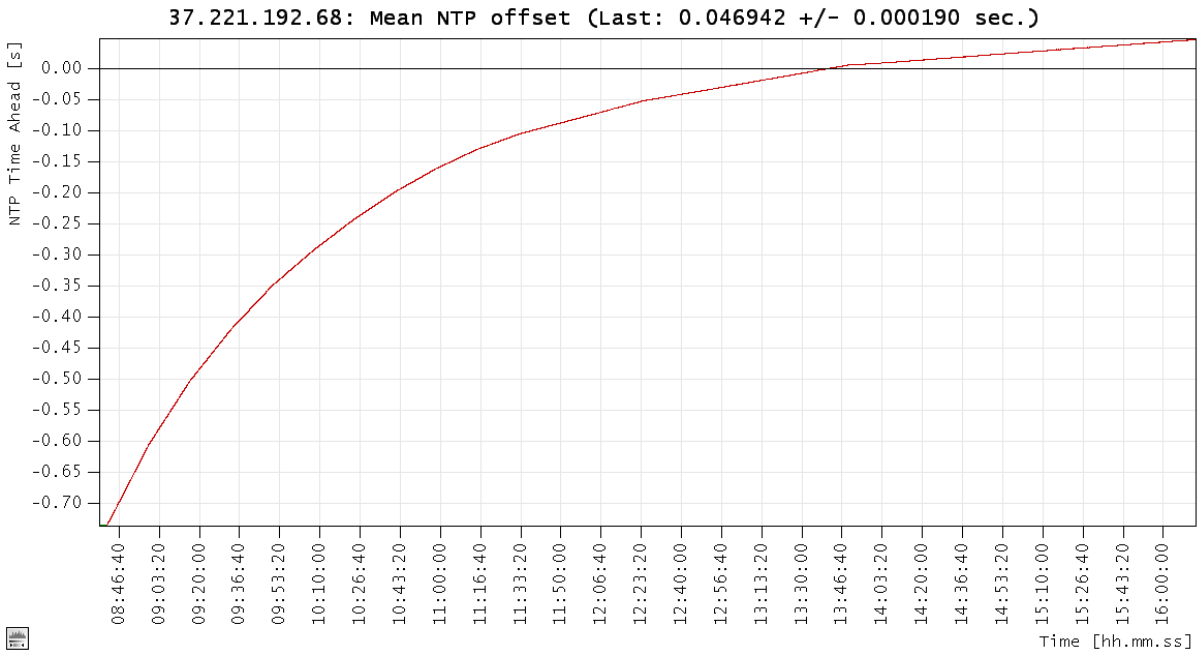

Fig. 4.3.3: NTP Offset during the adjustment (Windows XP).

Fig. 4.3.2 shows that the performance counter frequency offset jumps to about 140 ppm. This corresponds to an initial adjustment gain of 120 µs/s because the initial offset was already 20 ppm. The gain was reduced in steps over a long period of time (the total adjustment lasted from 8:46 to around 16:00). In the first part, the gain was reduced after about the same time until about 11:33. At that point, the granularity of *dwTimeAdjustment* prohibited smaller steps and the time between the modifications of *dwTimeAdjustment* was extended. This way, the target could be approached with a decreasing adjustment speed. The last step from about 13:50 represents the *dwTimeAdjustment* = 156250. The system time adjustment was still enabled, however the gain was 0.0 ms/s. At this point, the system drifted with its own drift rate.

Typical drifts of local time are in the area of a few µs/s. However, the smallest gain obtainable on Windows XP is 1/156250 = 6.4 µs. In practice, the drift may be higher than the smallest gain setting. This way, a final adjustment step may not move in the desired direction. This can be seen in Fig. 4.3.3. As mentioned, the whole scheme of how and when the various gain settings are applied is worked out ahead of the actual adjustment; however, the local drift can add a considerable offset when the adjustment takes many hours.

As described in 4.2., a lot can fail during an adjustment on newer Windows versions. The following plot was recorded during an adjustment on Windows 7:

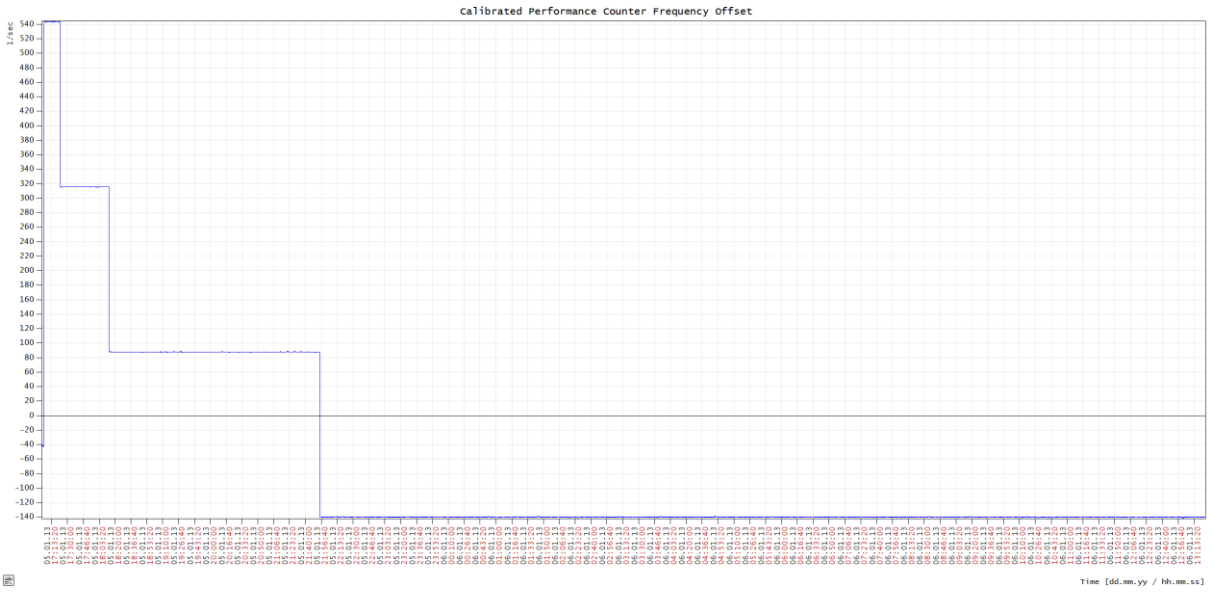

Fig. 4.3.4: Calibrated performance counter frequency during a system time adjustment (Windows 7).

The initial offset is about -40 ppm. The jump to 540 ppm indicates an initial gain of about 580 ppm or µs/s. Due to poor resolution (granularity of gain), the sign of the adjustment gain changes after just 2 steps and remains there for a long time (at least for another day). This is a typical example of a failing system time adjustment on a Windows 7 system. The offset time is basically the sum of the adjustments and is completely messed up (large negative offset) during this attempt.

Windows 8 has fixed the limited resolution of *dwTimeAdjustment* and shows adjustments comparable to Windows XP. The following two plots show a system time adjustment initiated by the Windows 8 internet time GUI:

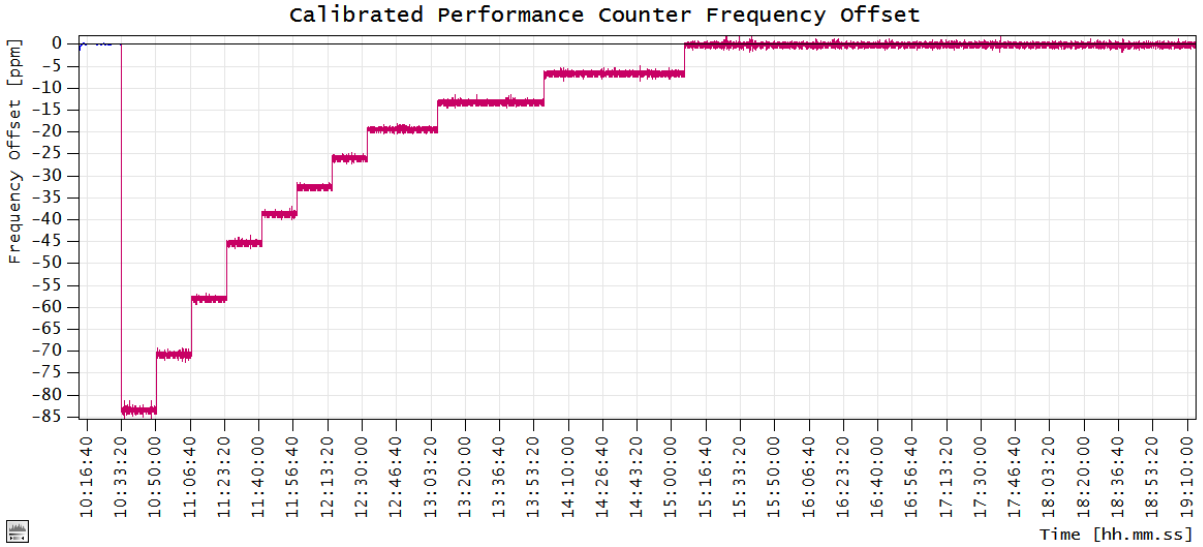

Fig. 4.3.5: Calibrated performance counter frequency during a system time adjustment (Windows 8).

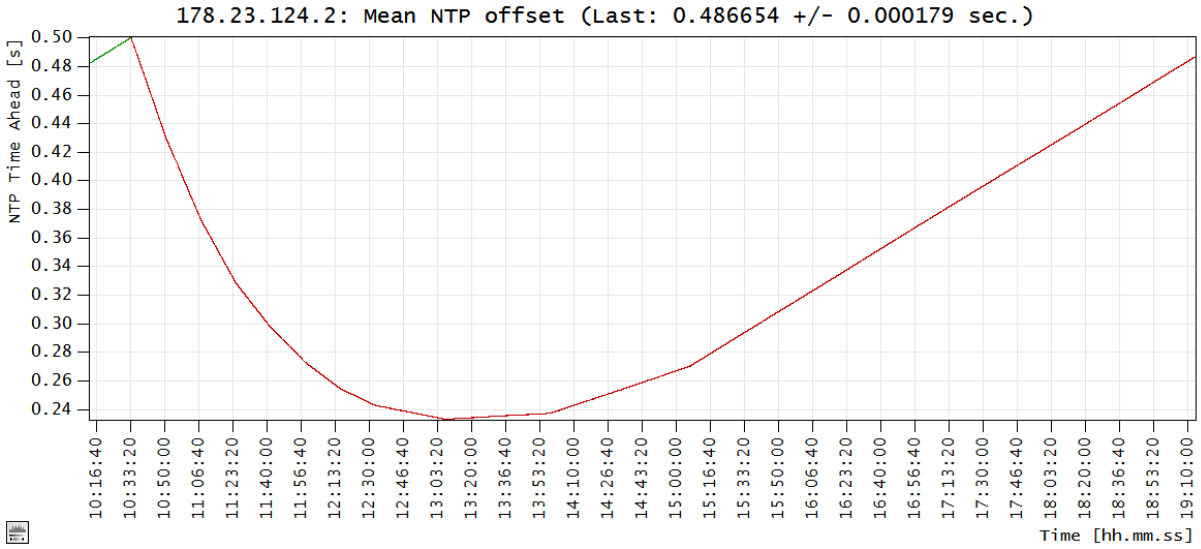

Fig. 4.3.6: NTP offset during a system time adjustment (Windows 8).

NTP monitoring was enabled at 10:15:15. From this point in time no adjustment was active, the system drifted at about 14.4 µs/s until 10:33:13 when the NTP offset reached 0.5 s (500 ms) and the system time adjustment was enabled. The procedure was performed by Windows in 11 steps, starting with *dwTimeAdjustment* = 156014:

```
156014 from 10:33:13 to 10:50:17, duration: 1024 s, gain = +83.333 µs/s,
       gained +85.333 ms, remaining offset: +414.7 ms

156012 from 10:50:17 to 11:07:21, duration: 1024 s, gain = +70.512 µs/s,
       gained +72.204 ms, remaining offset: +342.5 ms

156010 from 11:07:21 to 11:24:25, duration: 1024 s, gain = +57.692 µs/s,
       gained +59.077 ms, remaining offset: +283.4 ms

156008 from 11:24:25 to 11:41:30, duration: 1025 s, gain = +44.872 µs/s,
       gained +45.994 ms, remaining offset: +237.4 ms

156007 from 11:41:30 to 11:58:34, duration: 1024 s, gain = +38.461 µs/s,
       gained +39.384 ms, remaining offset: +198.0 ms

156006 from 11:58:34 to 12:15:37, duration: 1023 s, gain = +32.051 µs/s,
       gained +32.788 ms, remaining offset: +165.2 ms

156005 from 12:15:37 to 12:32:41, duration: 1024 s, gain = +25.641 µs/s,
       gained +26.256 ms, remaining offset: +138.9 ms

156004 from 12:32:41 to 13:06:49, duration: 2048 s, gain = +19.231 µs/s,
       gained +39.385 ms, remaining offset:  +99.5 ms

156003 from 13:06:49 to 13:58:02, duration: 3072 s, gain = +12.820 µs/s,
       gained +39.383 ms, remaining offset:  +60.1 ms

156002 from 13:58:02 to 15:06:17, duration: 4095 s, gain =  +6.410 µs/s,
       gained +26.249 ms, remaining offset:  +33.8 ms

156001 from 15:06:17 to ?
```

The list shows the progress of the adjustment for each setting of *dwTimeAdjustment* followed by the period of time during which *dwTimeAdjustment* was active. The gain was calculated using the expression given in 4. Consequently, the adjustment contribution and the remaining offset was calculated. The adjustment scheme looks identical to the scheme observed on Windows XP. Presumably no changes have been made to the systems adjustment tool. However some more details can be extracted from the list above:

- The scheme initially varies the step width of *dwTimeAdjustment* to obtain a degressive progress of the adjustment (156014 ... 012 ... 010 ... 008).
- It extends the duration to achieve a similar effect. However, the duration is unnecessarily fixed to multiples of 1024 seconds (156004: 2048 s, 156003 3072 s ...).
- Reaching *dwTimeAdjustment* = 156003 causes the desired gain of 12.82 µs/s to be below the systems drift. From this point onwards, the adjustment gain is not capable to compensate for the systems drift. This also becomes very obvious in the NTP offset plot, from about 13:06 the offset starts to increase again.
- Adjustment is effectively disabled at 15:06:17 by setting *dwTimeAdjustment* to 156001, which causes the (mean-) gain to be zero. But *lpTimeAdjustmentDisabled* remains FALSE for an unknown reason. Even many hours later (past 19:10:00) *lpTimeAdjustmentDisabled* was kept FALSE by Windows.

The observed offset at the end of the active adjustment was approx. 270 ms. The total adjustment time was 16386 s (10:33:11 to 15:06:17, 16 x 1024 s). The systems drift was 14.4 µs/s. At a drift rate of 14.4 µs/s the system drifted by 235.36 ms over the 16386 seconds. The difference to the observed offset of 270 ms is 34.64 ms. This corresponds to the remaining offset derived from the adjustment progress table.

This evidently shows that Windows calculates an adjustment scheme based on a one-time offset measurement ahead of the actual adjustment. Unfortunately the scheme captured here does allow for a remarkable remaining offset. The drift is not taken into account at any time. This way an adjustment, like the adjustment shown here, may take several hours to adjust the offset into the few milliseconds regime and just about the same time to be where the offset was prior to the attempt to adjust.

Larger offsets are not adjusted using such a scheme. An offset of say 10 seconds is simply corrected by setting the system time in one shot. This produces a jump in time which may be confusing to software, particularly when the jump in time is backwards.

4.4. Synchronizing to an NTP time provider

Windows broadcasts a WM_TIMECHANGE message to all top level windows when a system time change occurs. This can be used to detect changes of system time but it requires a window. However, there is no notification when the system time is adjusted. As a result, the system time changes gradually without any notification other than the actual changes in the flow of time. The only way to check this is through a frequent call to *GetSystemTimeAdjustment.* This is an obvious drawback. The state of such asynchronous behavior can only be closely estimated by calling *GetSystemTimeAdjustment* frequently.

Time control with high accuracy, as proposed by the Windows Timestamp Project, cannot accept the uncertainties and inaccuracies described here. The proposed solution is continuous synchronization of the system time to a network time using NTP. This automatic adjustment can be enabled by checking the "Autoadjust" checkbox of the GUI (Fig. 4.3.1). Synchronizations of the local time may still occur asynchronously when scheduled by the operating system; however, the service described here is capable of detecting and canceling them. Nevertheless, disabling the automatic synchronization provided by Windows (see the Windows GUI in section 1) is recommended in order to obtain the greatest accuracy.

The following graph shows a Windows 8 system:

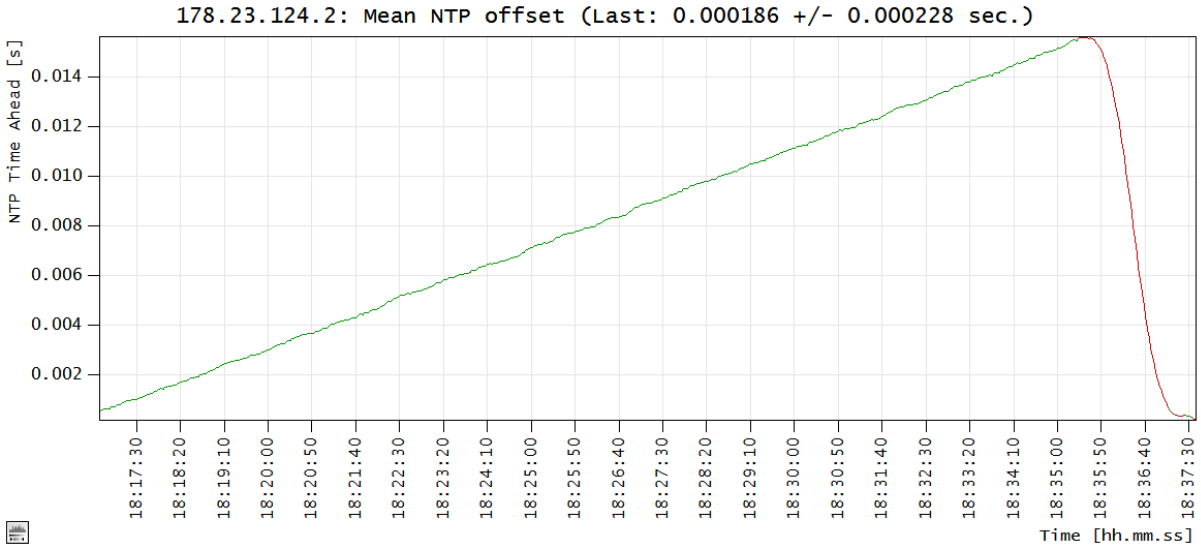

Fig. 4.4.1: Drift and autoadjust on a Windows 8 system.

NTP monitoring was started at around 18:16 and the local time drifted at a rate of about -14.2 µs/s. The NTP offset increased from around 0.0005 s to around 0.015 s within the next 19 min (green plot line). At about 18:35, the autoadjust was enabled and the local time was synchronized to the network time.

The effect of the system time adjustment on the performance counter frequency has been described in section 4.3. The plot of the calibrated performance counter offset for the adjustment shown in Fig. 4.4.1 is given below:

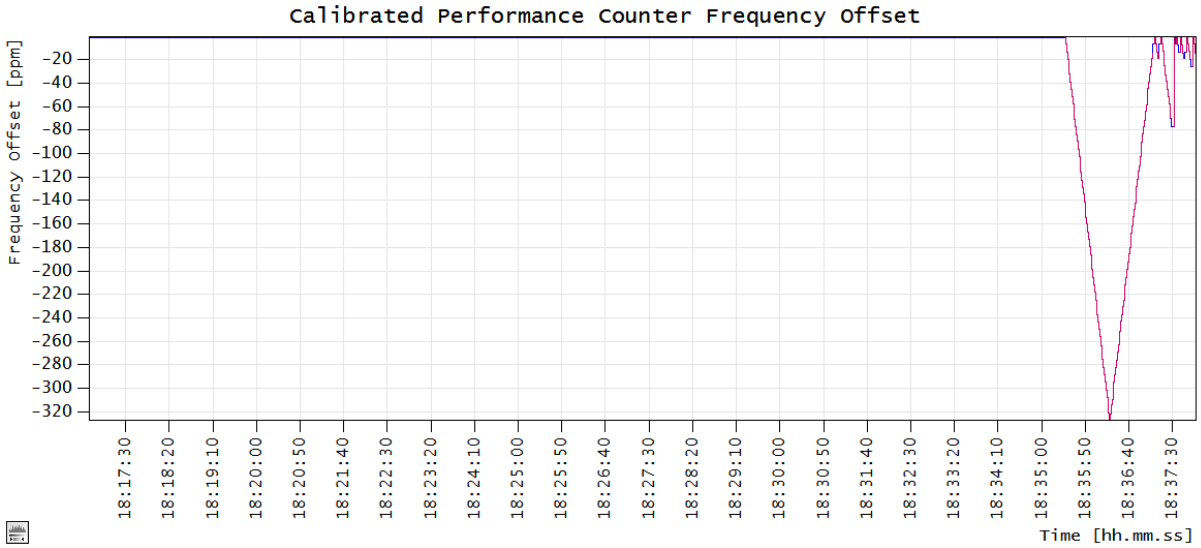

Fig. 4.4.2: Adjustment steps on a Windows 8 system.

Fig. 4.4.1 shows that the network time is running faster and the local time loses about 14 µs/s. Positive gains are required to catch up with the network time. The time service started by applying the smallest positive gain with *dwTimeAdjustment* = 156002. This resulted in a gain of 0.00641 ms/s. Afterwards, the value of *dwTimeAdjustment* was incremented periodically. At a value of 156051, the gain increased to 0.3205 ms/s. The *dwTimeAdjustment* was decremented periodically after half of the desired offset was

adjusted. A positive gain causes the system time to progress faster; the calibrated performance counter frequency consequently gets lowered with positive gains. As already mentioned, the calibrated performance counter offset is normalized to the performance counter frequency given by the system to show ppm. As a result the plot effectively shows negated gain values (e.g. a gain of +18.2 µs/s will show as -18.2 ppm).

The continuous adjustment results in a mean offset of the network time to local time in the range of a few 100 microseconds. However, this may be affected by network bandwidth and/or NTP server quality. The network time server pool used here is pool.ntp.org (it is highly recommended to read the information provided by this site). The accuracy of servers provided by this source typically outperforms the accuracy of time.windows.com. The available bandwidth is essential for very high accuracy. Heavy traffic on the network connection may temporarily drop the level of accuracy to within a few milliseconds.

The next graph shows a continuous adjustment interrupted by a three minute drift phase in between to highlight the narrow band in which the NTP offset is held during the adjustment:

Fig. 4.4.3: Windows 7: Continuous adjustment interrupted by a three minutes drift phase.

This figure was taken as a screenshot of the GUI to show the estimated local drift. This local drift can be estimated from the mean of the applied gains after a few minutes of continuous operation of "autoadjust". Its value appears in the "all output" tab and at the end of the NTP status line when available.

The quality of adjustment becomes visible when the network time offset drifts. In just three minutes, the offset drifted to about 2.7 ms. If high accuracy is required, it is not only necessary to synchronize the local time to a network time periodically; it is essential to synchronize it continuously.

Note: Version 1.70 introduced the precision mode. The NTP capture leaves this mode when the offset exceeds 2 ms and re-enters it when the offset is below 1.5 ms. This behavior was already visible in fig. 4.3.1 and becomes also visible here.

4.5. Conclusions

Windows synchronization to a network time reference has proved to not be very accurate. In particular, Windows versions VISTA and 7 seem to have lost some of the capabilities for some unknown reason. Unfortunately, there is not much information on this issue and the little information available basically says that Windows time synchronization should not be expected to be more accurate than a few seconds and that there may be a mishap in the behavior of *SetSystemTimeAdjustment* with respect to the meaning of the value of *dwTimeAdjustment*. Only Windows 8 has now overcome these drawbacks and its system time adjustment performs like it did on Windows XP.

Unfortunately, there are still many NTP synchronization packages around which operate under the assumption of the current MSDN description that "For each *lpTimeIncrement* period of time that actually passes, *lpTimeAdjustment* will be added to the time of day". Evidently, this assumption is not true for Windows VISTA and Windows 7. These versions need software that is capable of dealing with the artifacts described here to set the system time correctly to obtain good accuracy.

Offsets of system time may drift seconds per day. Even on systems with a low drift rate the drift can easily reach half a second per day. This can only be overcome by a correction of the systems knowledge of its clock frequency. Newer Windows versions calibrate the performance counter frequency (result of QueryPerformanceFrequency) at boot time when operating with TSC and/or HPET. This was initially done by Windows 7 and has improved with Windows 8. But there does not seem to be an on the fly correction of this value while a network time synchronization occurs. This is basically the reason for the noticeable drift and the need for a continuous adjustment. Windows 8.1 has not shown any improvements with respect to the "build in" system time adjustment.

...

Note: Don't miss more details described on the "News" page. A pdf version of the **News History** can be downloaded here.