Time Synchronization Accuracy With NTP
Table of Contents

Time Synchronization Accuracy With NTP ................................................................. 1
Accuracy Determining Factors At The Client Side .................................................. 1
Accuracy Determining Factors On The Network ..................................................... 1
Time Server Tasks vs. Time Client Tasks ............................................................. 2
Example Time Synchronization Results For Linux .................................................. 3
Example Time Synchronization Results For Windows ............................................. 5
  Limitations Under Older Windows Versions ....................................................... 5
  Enhancements With Windows 8 and Newer ......................................................... 9
Conclusion ............................................................................................................ 11
Time Synchronization Accuracy With NTP

Accuracy Determining Factors At The Client Side

Of course the time accuracy you can yield at the client side depends on the accuracy of the available reference time source, but it also depends strongly on a number of preconditions at the client side, e.g. the operating system type (Windows, Linux, ...) and particular version of that OS, the CPU type (good TSC support, or not), the chipset on the mainboard, and the quality of the oscillator on the mainboard which determines the general stability of the system time. In addition there are hardware limitations introduced by the concept of the system busses, e.g. PCI or USB, interrupt latencies, etc.

Linux, FreeBSD, etc. provide a good programming interface to read the current system time accurately, and to apply adjustments to the system time properly. Windows is much worse than this.

There are Intel x86-compatible CPU types providing reliable TSC counters which can be used to measure and thus at least partially compensate some latencies. On other CPU types the TSCs of different CPU cores may not be not synchronized to each other, or the TSC clock frequency also changes when the CPU clock changes due to power saving. Non-Intel CPUs may not even provide a high resolution counter similar to the TSC, so with these CPU types it's much harder to achieve accurate time synchronization.

The frequency of the oscillator on the particular mainboard has an individual mean offset from its nominal frequency, and in addition the frequency varies more or less with changes of the ambient temperature inside or outside the PC housing. This can be due to changes in the CPU load causing more or less power consumption and associated heat inside the PC housing, an air condition kicking in and out in a server room, or just the daily temperature variations between day and night.

Since the time drift depends on the frequency variations, the control loop of a good time synchronization software tries to determine and compensate the frequency of the oscillator, and steers the oscillator so that a time offset doesn't even evolve. However, how good this works depends on the factors mentioned above.

Accuracy Determining Factors On The Network

Whenever a network packet is sent from one application running on one node to another application running on a different node the network packet has to traverse a path which include quite a number of unknown delays or latencies:
All the yellow lines indicate unknown processing delays. If the packet has to go through a switch, but the output port is just busy with a different packet (red lines) then the packet is delayed. Also different link speeds on the 2 involved switch ports cause an asymmetry of the packet delay which results in a time offset. Only the cable delays (green lines) are really constant.

**Hardware time stamping** of network packets can be used to measure and thus compensate the variable packet delays. However, this is mainly used by the PTP protocol and requires hardware support for time stamping specific packets, and it has to be done on every network port blue flashes). Unfortunately there are only switches available which support this for PTP, but there are no switches which support this for NTP. On the other hand, even if hardware time stamping of NTP packets was supported the current version of the NTP protocol doesn't even provide a proper way to forward the measured delays.

The reference implementation of the NTP protocol, ntpd, can yield pretty good accuracy anyway with standard hardware (no time stamping support) due to its built-in filter algorithms.

However, there are still possible time offset errors due to network asymmetries. See:

- [Time Synchronization Errors Caused By Network Asymmetries](#)

**Time Server Tasks vs. Time Client Tasks**

Assuming a time source provides accurate time, the accuray you can yield when synchronizing your system time on a specific client hardware depends mostly on the implementation of the software on the system to be synchronized. The only things an NTP server has to do when it receives an NTP request packet from a client is to:

- Add a time stamp to the packet when the request packet comes in
- Eventually check if some restrictions apply, and prepare the packet as reply to the client
- Add another time stamp to the reply packet immediately before the packet is sent
On the other hand, the task of an NTP client is to

- Prepare a request packet and add a time stamp of its own system time immediately before the packet is sent to the server
- Wait for the reply packet from the server, and get another time stamp of its own system time as soon as the packet arrives
- Evaluate the 4 time stamps associated with the reply packet to estimate the packet delay and determine its own time offset from the server
- Adjust its own system time so that the time offset is minimized

So obviously it’s the task of the client software to determine and compensate the packet delays on the network, eventually filter out outliers if the a request and/or reply packet has been delayed more than usually, and adjust its own system time more or less accurately. The server can’t do this, it can just provide accurate time.

A good client implementation like ntpd evaluates the time stamps from a series of subsequent packet exchanges, tries to filter out network delays and jitter, and takes much care to adjust the system time as smoothly as possible, unnoticeably for applications.

A simple client implementation could just pick up the time from the server reply and just set the system time whenever it has queried the time from a server. This is what some SNTP (Simple NTP) implementations are doing, and this obviously only yields less accuracy than a more sophisticated implementation.

Normally the time synchronization software shows the time offset it has determined. The precision and accuracy of this time offset when the system time control loop has settled depends on many conditions: the accuracy and precision provided by the time source (NTP server, GPS receiver, or whatever), on the way the time source is accessible (via the network, via USB, via PCI, via 1 PPS signal, …), on the type and version of the client operating system (Linux, Windows, …) and the version of the OS, as well as the client hardware (CPU, chipset, physical or virtual machine), and in case of a VM on the type, version, and configuration of the virtualization software (VMware, XEN, Hyper-V, …).

For any of the combinations above the computed time offset may be correct, or be more or less off due to some systematic errors. E.g., an asymmetric network connection (e.g. an ADSL line) can cause a real time offset of a few milliseconds which the client software can’t determine, and thus such systematic offsets can’t be compensated. In case of a 1 PPS hardware signal a time offset error up to a few 10s of microseconds can be caused by interrupt latency, depending on the system characteristics.

### Example Time Synchronization Results For Linux

The example output below is from an `ntpq -p` command on a Linux machine synchronized with a Meinberg GPS180PEX PCI card as time reference listed as SHM(0):

<table>
<thead>
<tr>
<th>remote</th>
<th>refid</th>
<th>st t when</th>
<th>poll</th>
<th>reach</th>
<th>delay</th>
<th>offset</th>
<th>jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>*SHM(0)</td>
<td>.shm0.</td>
<td>0 l 3 16 377</td>
<td>0.000</td>
<td>-0.000</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Time Synchronization Results For Linux
<table>
<thead>
<tr>
<th>Source</th>
<th>Device Type</th>
<th>Uid</th>
<th>Pid</th>
<th>Process</th>
<th>Time Base</th>
<th>Time Deviation</th>
<th>Time Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt-martin.py.me</td>
<td>MRS</td>
<td>1</td>
<td>15</td>
<td>64</td>
<td>377</td>
<td>0.067</td>
<td>-0.004</td>
</tr>
<tr>
<td>ntp-master-1.py</td>
<td>PPS</td>
<td>1</td>
<td>7</td>
<td>64</td>
<td>377</td>
<td>0.201</td>
<td>-0.010</td>
</tr>
<tr>
<td>ptbtime1.ptb.de</td>
<td>PTB</td>
<td>1</td>
<td>109</td>
<td>128</td>
<td>377</td>
<td>12.249</td>
<td>0.023</td>
</tr>
<tr>
<td>ptbtime2.ptb.de</td>
<td>PTB</td>
<td>1</td>
<td>35</td>
<td>128</td>
<td>377</td>
<td>11.738</td>
<td>0.247</td>
</tr>
<tr>
<td>ptbtime3.ptb.de</td>
<td>PTB</td>
<td>1</td>
<td>16</td>
<td>128</td>
<td>377</td>
<td>11.889</td>
<td>0.266</td>
</tr>
<tr>
<td>host-24-56-178-</td>
<td>ACTS</td>
<td>1</td>
<td>148</td>
<td>128</td>
<td>376</td>
<td>146.136</td>
<td>-0.959</td>
</tr>
</tbody>
</table>

All sources except SHM(0) have been configured with the `noselect` keyword, so `ntpd` only polls them and computes the estimated time offset, but doesn't use them for synchronization.

`lt-martin` and `ntp-master-1` are GPS controlled Meinberg LANTIME devices on the same subnet, where `lt-martin` is closer (less switches in the path) and thus shows less jitter but a little more accuracy than `ntp-master-1`.

The 3 PTB servers and the host with refid ACTS are only reachable across the WAN/internet, but the determined time offset is still below 1 ms even though the packet delay to the latter is 146 ms. These are quite good results for a setup without special switches and routers. However, this level of accuracy can't be guaranteed since it depends on a number of factors as explained above.

Here is a graph of the `loopstats` file recorded over a few hours:

![Graph of loopstats file](image)

Though the graph above looks very nice, it has been generated of only a small section of a `loopstats` file which has been recorded over about 4 days. The full graph is shown below, and in the frequency offset curve you can see the cyclic daily frequency changes due to cyclic daily temperature changes, resulting in a varying time offset which `ntpd` tries to compensate:
This shows how the stability / temperature sensitivity of the client affects the accuracy of the system time.

**Example Time Synchronization Results For Windows**

Time synchronization under Windows in general can be pretty tricky, mostly due to limitations in individual Windows versions. An overview of time synchronization support in Windows can be found in the article [NTP And Windows History](#).

**Limitations Under Older Windows Versions**

Specifically with `ntpd` the control loop seems to become less stable if the polling interval increases above 7 (128 s). This can be seen in the graph below. Whenever the poll number increases the time accuracy increases, too:
On the other hand, if the polling interval is clamped to 6 (64 s), time synchronization becomes much smoother:
**Windows Vista and Windows 7 contain a bug** where small time corrections are not accepted by the Windows system timekeeping:

- `SetSystemTimeAdjustment` May Loose Adjustments Less than 16
  
  [https://support.microsoft.com/kb/2537623](https://support.microsoft.com/kb/2537623)

ntpd-4.2.7p349 (a development version) and later versions including the subsequent release version ntpd-4.2.8* contain a workaround for this problem.

The next graph shows results from a Windows 7 machine with unlimited polling interval:
Again, you can still see this is not very stable for higher polling intervals. On the other hand, the workaround for the 16 ms bug in Windows can work only properly if the polling interval is at least 6 (64 s), so it doesn't make much sense to reduce the polling interval below 6. As the next graph shows the time discipline under Windows 7 becomes only smooth after the polling interval has ramped up to 6:
So the suggested configuration of upstream NTP servers for `ntpd` under Windows includes lines like this:

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

where `aa.bb.cc.dd` has to be replaced by the real IP address or hostname of the external NTP server.

All loopstats displayed above have been recorded using a Linux machine with GPS-controlled `ntpd` on the local subnet.

**Enhancements With Windows 8 and Newer**

Starting with Windows 8 a new Windows API has been introduced which allows applications to read high resolution system time stamps, and thus yield better precision.

The graph below shows the loopstats recorded by `ntpd 4.2.8p8` under Windows 10 over about 6 days, with the polling interval fixed to 6:
Unfortunately the machine has been rebooted during the experiment, but we can anyway see the daily frequency changes of the oscillator on the mainboard due to temperature changes, which have to be compensated by ntpd.

On the other hand, the same experiment on a different PC, run at the same time, in the same room, also with ntpd 4.2.8p8 under Windows 10, doesn't show these daily temperature cycles:
This shows once more the importance of the stability of the oscillator assembled on the PC's mainboard.

Again, all loopstats displayed above have been recorded using a Linux machine with GPS-controlled ntpd on the local subnet.

**Conclusion**

Timekeeping in general can be pretty tricky, and the more accuracy and precision you need, the more effort has to be taken to achieve it, and the more limiting conditions will come into effect. Network characteristics can degrade the resulting accuracy, and also virtualization is one more condition affecting the time synchronization performance.

See:

- Time Synchronization Errors Caused By Network Asymmetries
- Time Synchronization In Virtual Machines