Network Frequency Transfer 0.01
2009 Kasper Pedersen

NFT lets you perform oscillator verification with a PC and an Internet connection.

It is being offered as "Likely unsuitable for any purpose whatsoever, has bugs, and may eat babies."

NFT exploits NTP to do frequency transfer across the Internet. Usage is simple:

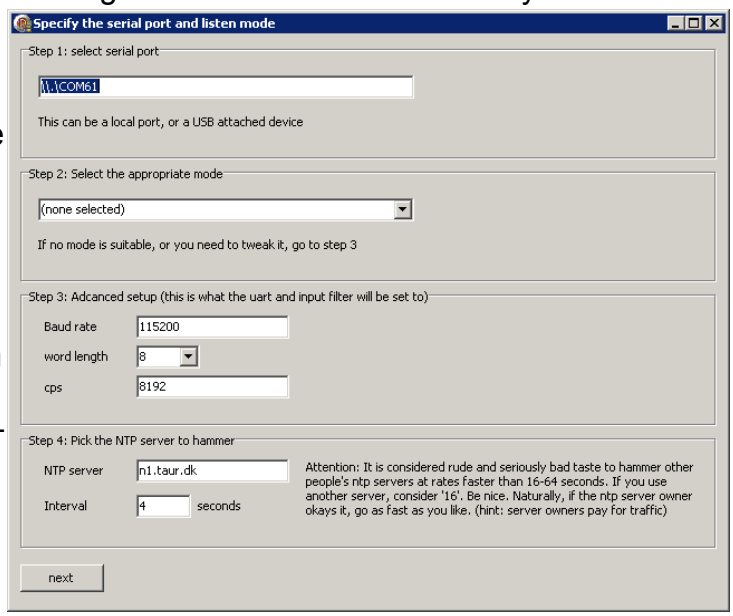
- You generate a ~1kHz (1Hz-10kHz) signal from the oscillator to be calibrated. It must be interpretable by a PC UART as a character at 115200 bps. See hints on the last page.
- NFT on a PC locks a software PLL to the incoming signal. The PC now has a realization of your oscillator.
- NFT sends NTP queries to an NTP server, time stamped against your oscillator. The drift is calculated, and out comes the frequency offset. Provided the NTP server being asked is good.

Settings dialog

Put in the serial port, and choose something that looks close to your scenario. Don't panic if there is no exact match: The advanced settings has uart settings, and the expected number of characters to receive per second. If you have a pulse rate that is not present, input it in the cps field. You can enter non-integer rates.
You do not need a uart in your hardware to generate the pulse. Anything with 8-76µs high time at the PC end and <11kHz repetition rate is acceptable.

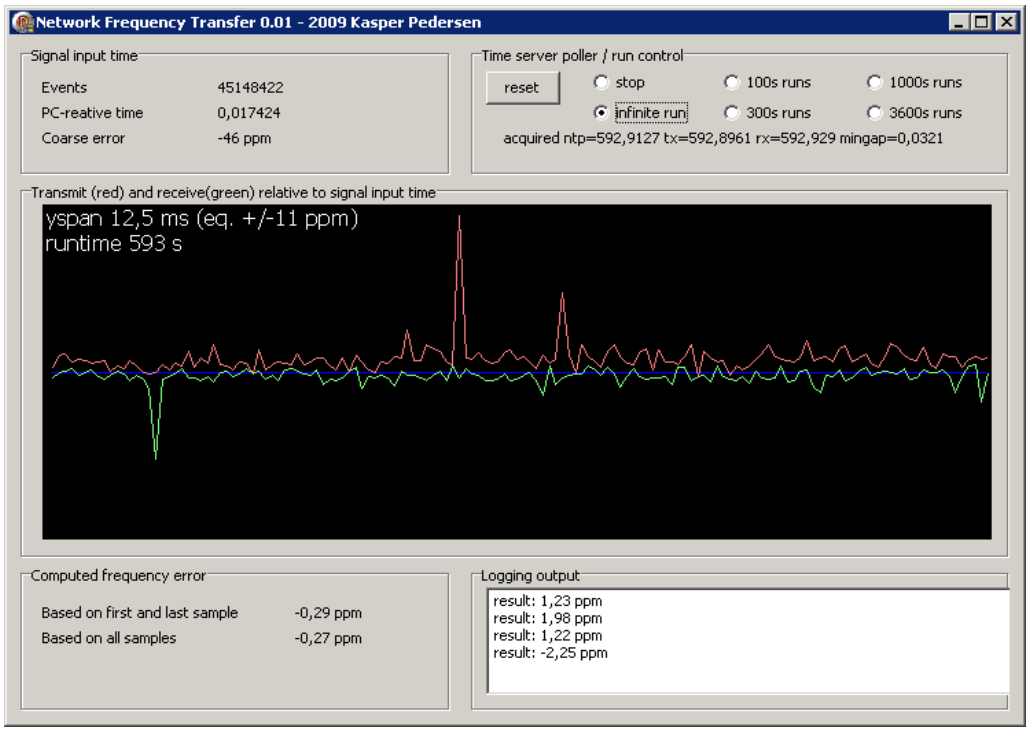
The final choise is the NTP server to ask. The default choice is my privately-run n1.taur.dk. *The server is a stratum-1 server and should NOT be used directly by clients otherwise. It is used by stratum-2 servers, and these are more reliable from the user's perspective (if n1's refclock goes down, n1 becomes unusable. A stratum-2 will provide time from another good stratum-1).*

It is important that the NTP server used has a good local clock. This basically means that anything running on Windows is unusable. Fortunately, running a stratum 1 on Windows is so bloody hard that almost noone tries (and those I have seen are miserable). If the server is bad, you'll be able to see it. Having a good local clock also means a clock that ticks at the correct rate. That is why I put in n1 as the default; Its local clock is steered against PTB in Germany and is better than 1ppm on a bad day. The final point is the poll rate - I allow 4s for the purposes of NFT, others may get angry if you go faster than 16s or 64s.



NFT will honor KoD packets.

The main window



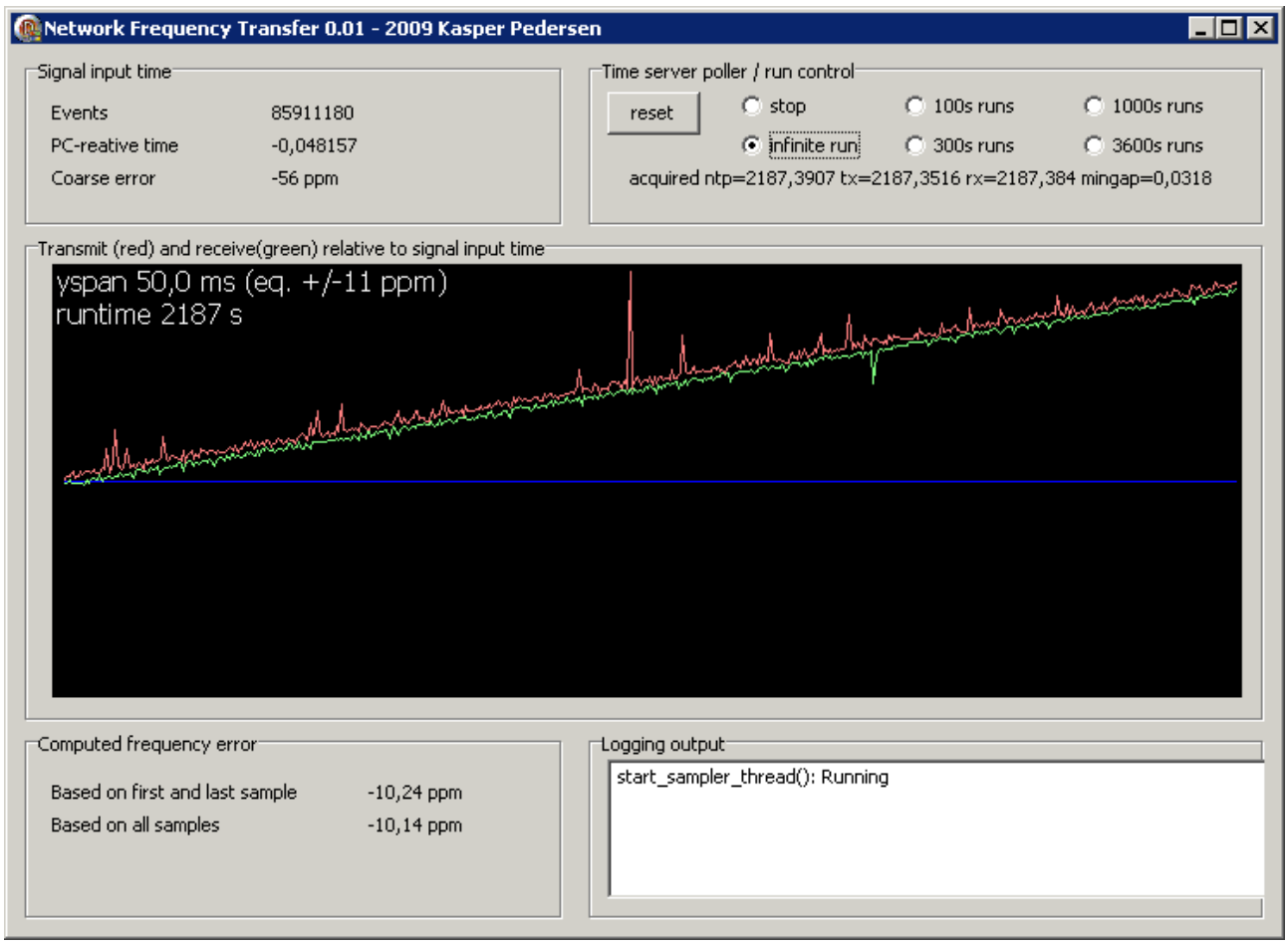
Signal input time:
This is the clock running from the external signal. Events is the number of characters received, PC-relative is the delta between your clock and the PC, and the Coarse Error is the frequency delta between your clock and the PC's. If it ticks at about the right rate, and CE is below +/-1000 ppm, it's running properly. It will take ~10s to settle.

Time server poller/run control:
Tells how long to collect data for. If you pick 100 seconds, the program will spit out a frequency in the logging window every 100 seconds. When you have 3 readings you will know whether you need to increase the integration time, or whether the current precision is sufficient for your purposes. Longer integration times yield better precision. 'Reset' restarts the integration period. Do not start the run until CE above has settled.

Transmit/Receive graph:
The NTP server wieved by against your clock. A climbing graph means your clock is slow (or the NTP server is fast). This allows you to see how noisy the NTP server is.

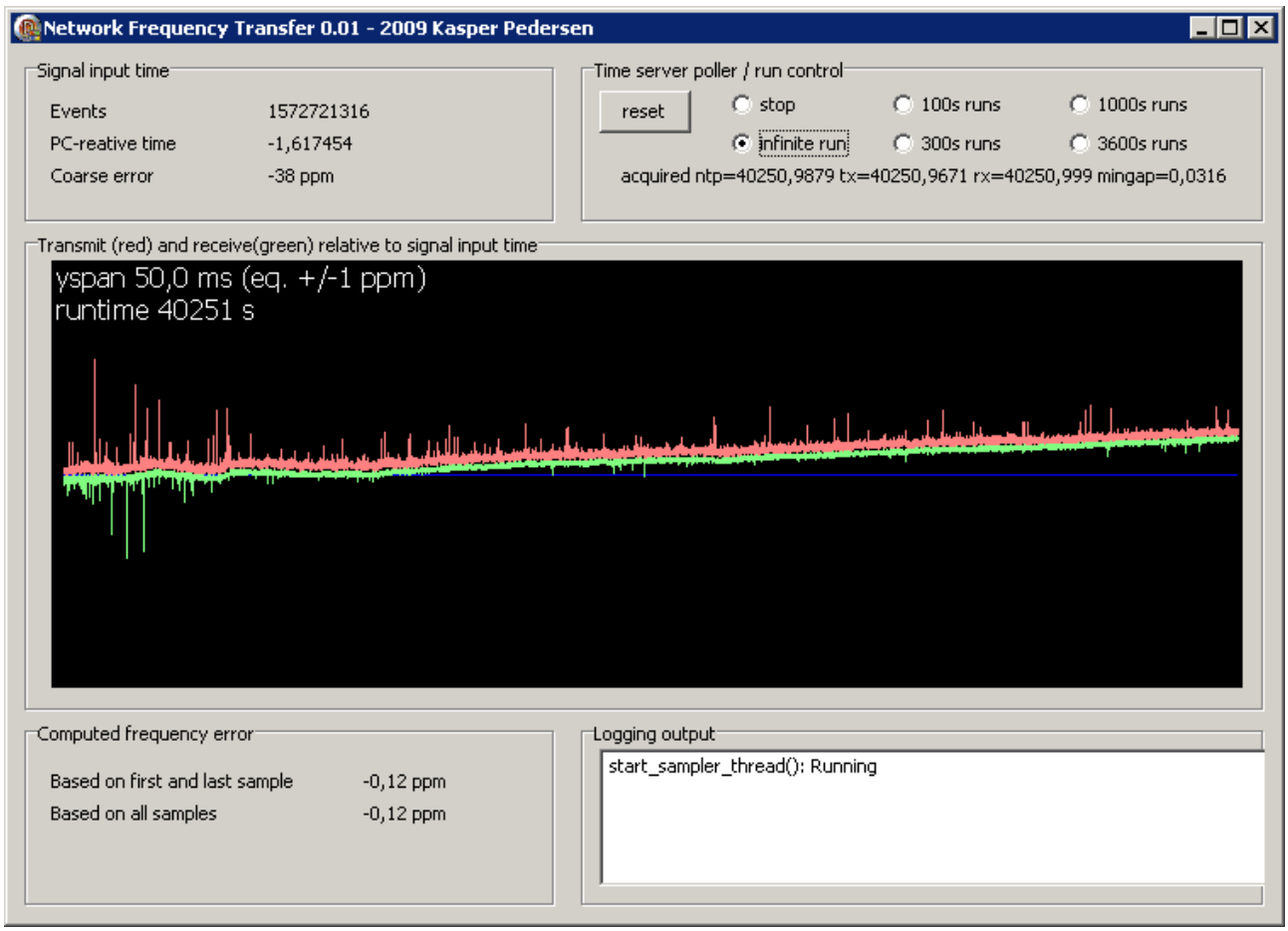
Computed frequency error:
Two frequency errors are computed. The topmost is on the first and last point of the graph. The bottom is a least-squares regression with culling of bad samples. Think of it as a filtered average.

The above screenshot was taken after switching from 100s runs to infinite run. The oscillator feeding the PC is a TCXO that's off by -0.12 ppm.



In this plot, the oscillator being measured is -10.12 ppm (it is running slow).

Since the USB serial port I used can run very high bit rates, this run (and all others in this document) are done using 10MHz/256=39062.5kHz applied to the serial port RXD line, and the port set to 500kbps. NFT has filters to take out the 1ms jitter coming from the USB polling, and they are better at high cps rates.
(the high period in a 39kHz signal is 0.5/39kHz=12µs wide, and a period is 24µs. 12µs at 500kbps is 6 bits, so the signal is seen as 6 bit times low, 6 bit times high a perfectly valid character.)



This is a 11-hour run against n1.taur.dk over the Internet. According to my local counter and GPS frequency standard, the truth is:

That's -0.12 ppm. Not bad over a shared channel with 32ms latency. Yes, the range indicator lights are broken on this counter.

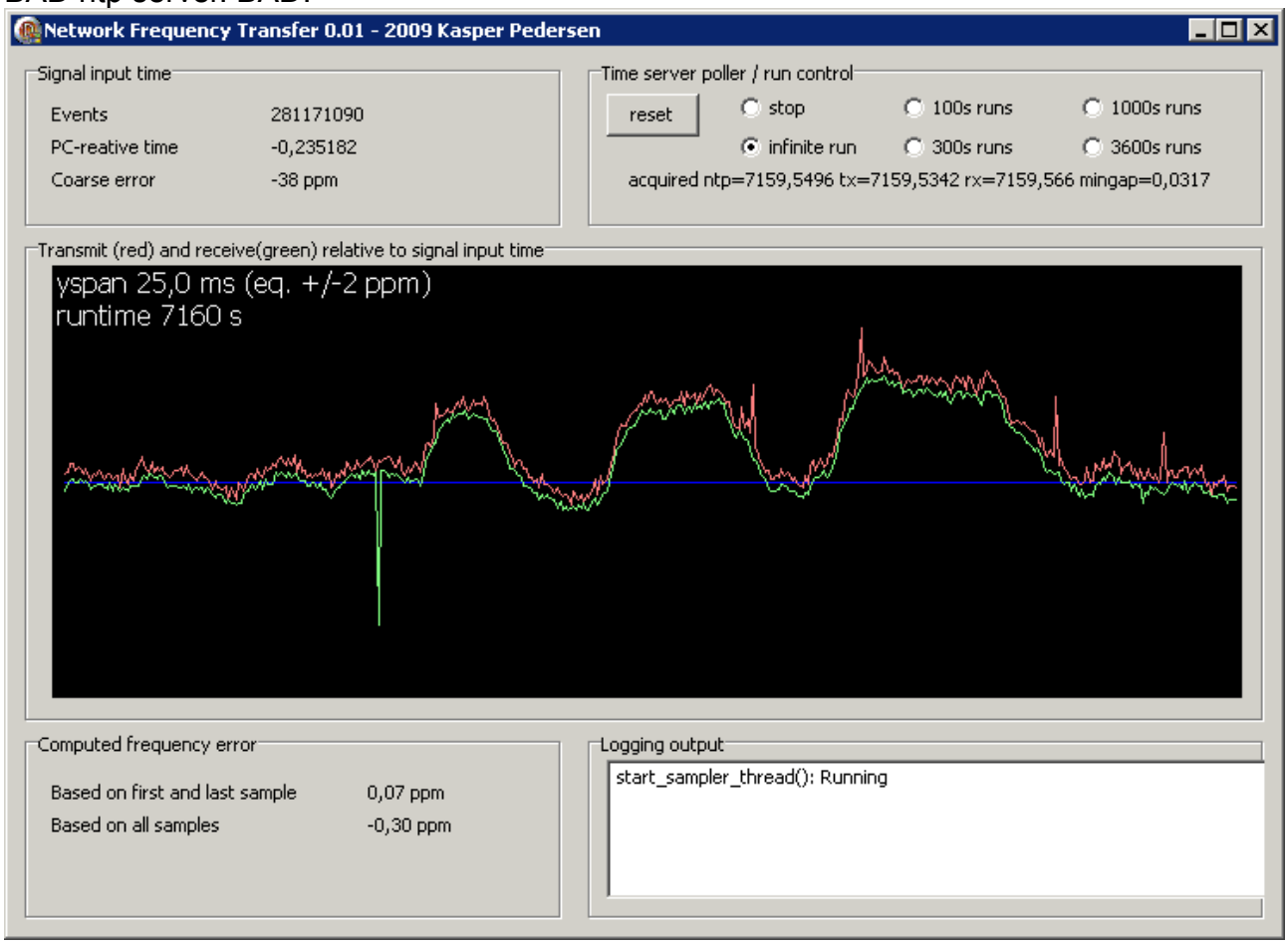
When you want to do really long runs, increasing the polling interval to 64 is a good idea. Otherwise the repainting of the display becomes rather slow.

About choosing the NTP server:
NTP for timekeeing suffers when the latency ('ping time') is large. In the case of NTP for frequency transfer large latency is perfectly fine. The only thing that hurts NFT is varying latency and unstable clocks. The plot below is against ntp1.emnet.dk, emnet's stratum-1 server. I believe they have a GPS receiver feeding it, but they do not use PPS/time mark, thus the local clock wobbles around by as much as 30ms, which means that any short-term observation using this ntp server is likely to give totally bogus numbers.

This server is so bad that it will regularly be declared a falseticker by ntpd.

Even with such a poor ntp server, if you wait long enough, you get usable data: The span on the graph is +/-3ppm, that is, if it was climbing from the center left to the top right, that would be -3ppm. So the reading is still better than 2ppm; It just took 2 hours to get there.

BAD ntp server. BAD!



Generating the timing pulse

To be understood by a 115200N81 uart, the pulse must have a high (>+3V) time of 8-78µs when arriving at the PC RXD pin. If you have a max232 or similar between what you are measuring and the PC, remember that it inverts the signal.

If you generate the pulse using an output pin, it must be 8-78 us long, and less than 11kHz repetition rate. A square wave of 6.4-11 kHz meets this, so 32768/4=8192 Hz would be a good choice.

If you generate the pulse using a uart in a microcontroller, send a 0xFC character. The baud rate does not need to match as well as it does for data; A baud rate between 230400 and 28800 will generate a usable character on the PC end.