NTP Precision Time Synchronization

David L. Mills
University of Delaware
http://www.eecis.udel.edu/~mills
mailto:mills@udel.edu

From *pogo*, Walt Kelly

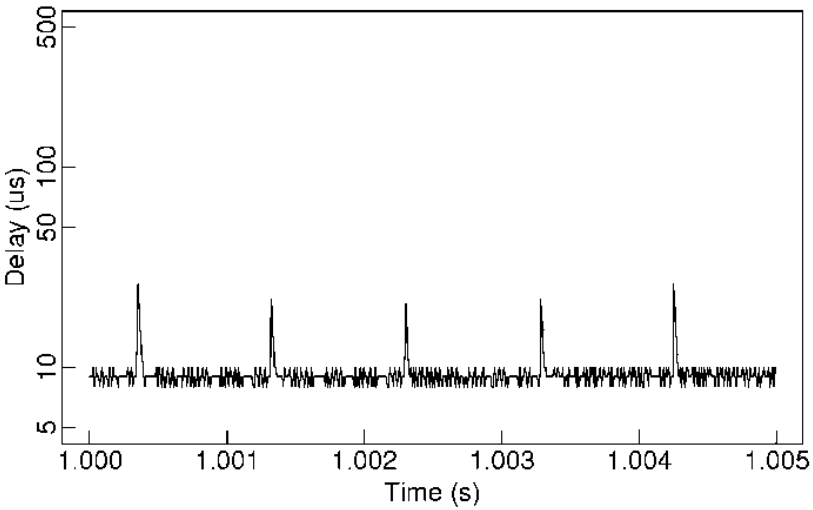
Precision time performance issues

- Improved clock filter algorithm reduces network jitter
- Operating system kernel modifications achieve time resolution of 1 ns and frequency resolution of .001 PPM using NTP and PPS sources.
- With kernel modifications, residual errors are reducec to less than 2 µs RMS with PPS source and less than 20 µs over a 100-Mb LAN.
- New optional interleaved on-wire protocol minimizes errors due to output queueing latencies.
- With this protocol and hardware timestamps in the NIC, residual errors over a LAN can be reduced to the order of PPS signal.
- Using external oscillator or NIC oscillator as clock source, residual errors can be reduced to the order of IEEE 1588 PTP.
- Optional precision timing sources using GPS, LORAN-C and cesium clocks.

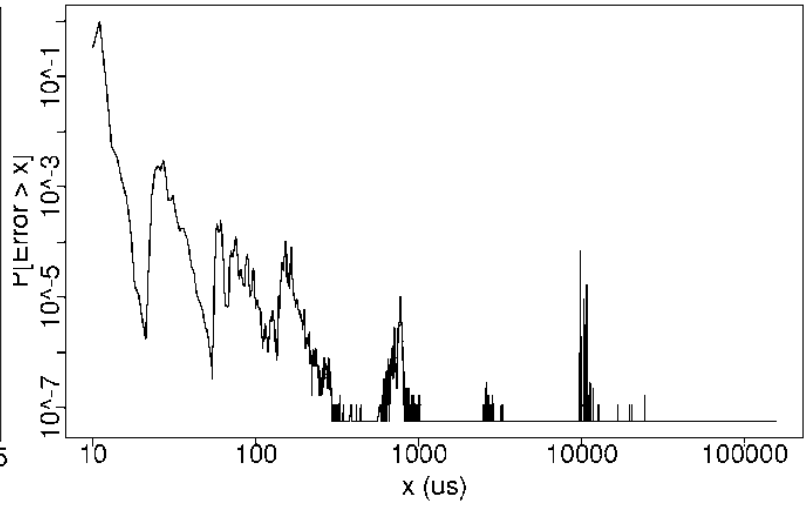
Part 1 – quick fixes

- Assess errors due to kernel latencies
- Reduce sawtooth errors due to software frequency discipline
- Reduce network jitter using the clock filter
- Minimize latencies in the operating system and network

Errors due to kernel latencies



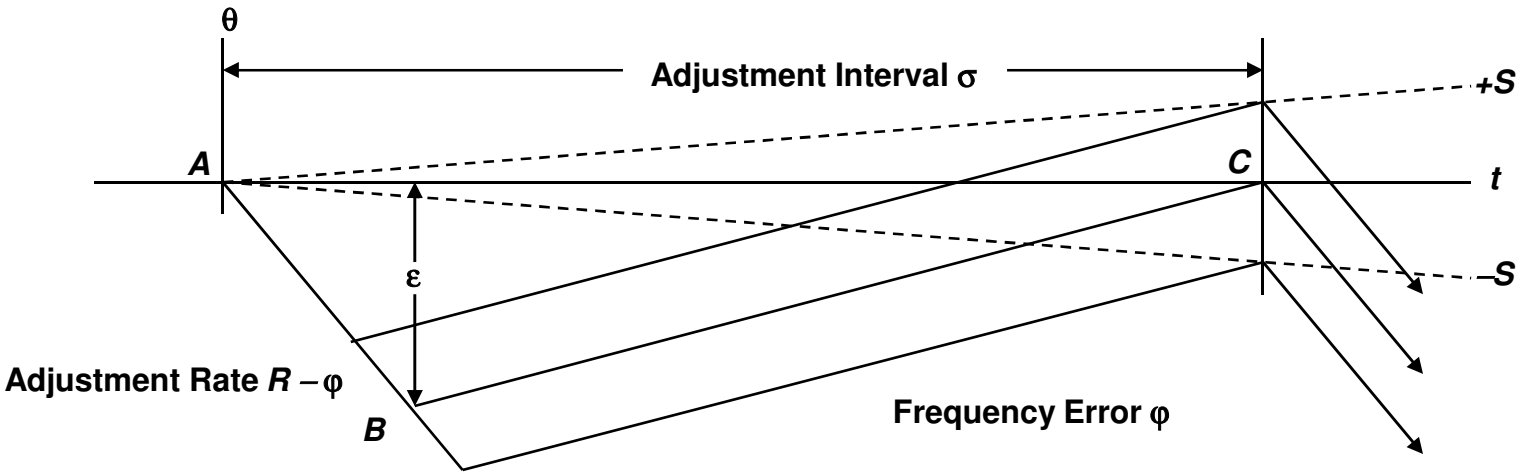
(a) Latency for `getimeofday()` Call



(b) Latency Distribution for (a)

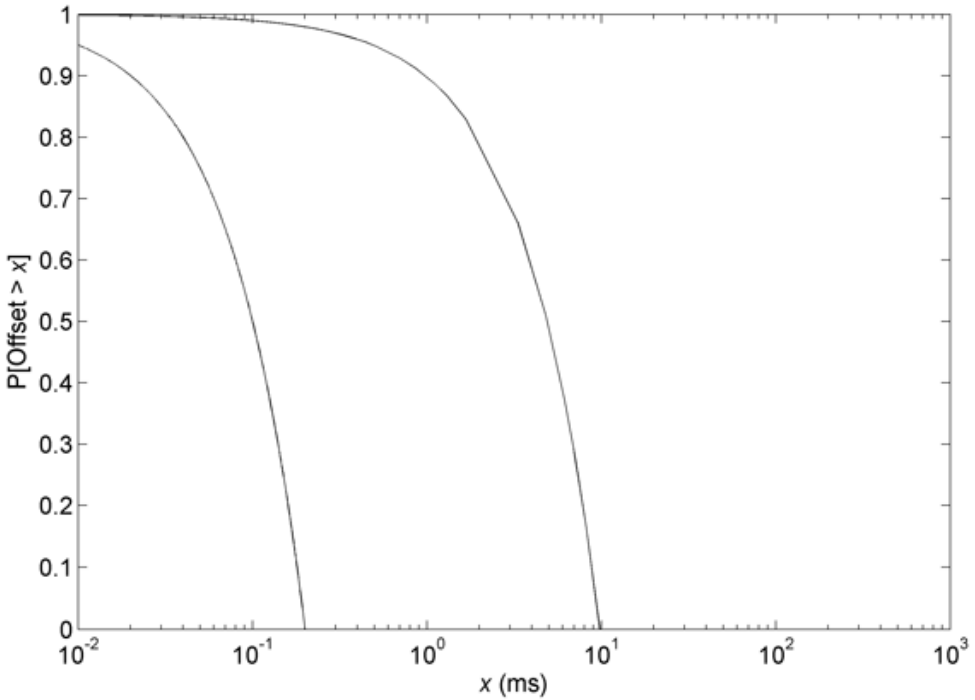
- These graphs were constructed using a Digital Alpha and OSF/1 V3.2 with precision time kernel modifications
- (a) Measured latency for gettimeofday() call
  - spikes are due to timer interrupt routine
- (b) Probability distribution for (a) measured over about ten minutes
  - Note peaks near 1 ms due timer interrupt routine, others may be due to cache reloads, context switches and time slicing
  - Biggest surprise is very long tail to large fractions of a second

Sawtooth errors due to software frequency discipline



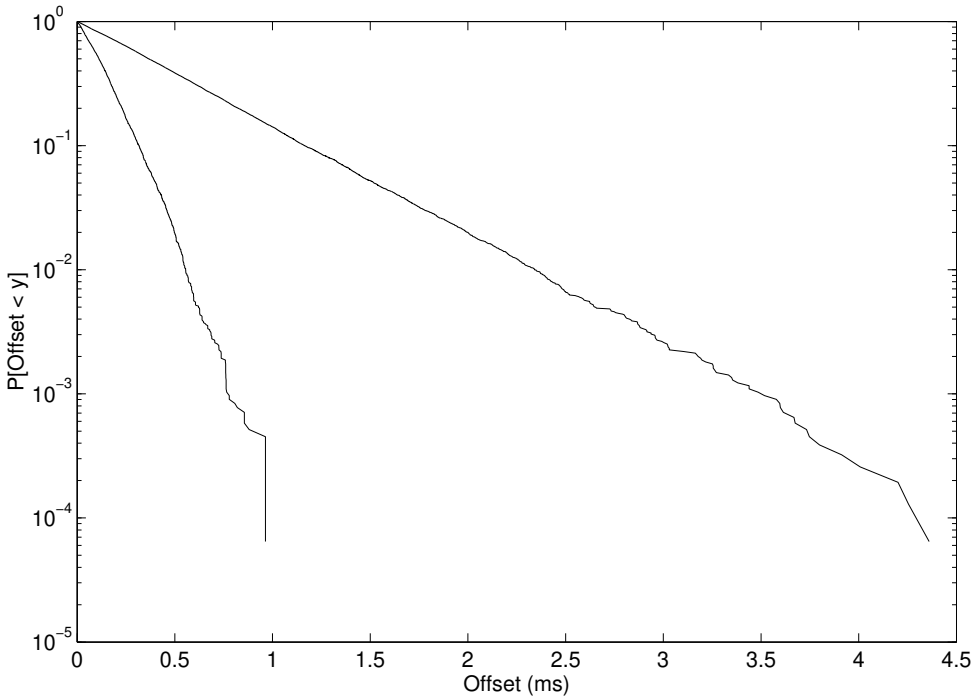
- Unix `adjtime()` slews frequency at net rate $R - \varphi$ PPM beginning at *A*
- Slew continues to *B*, depending on the programmed frequency offset *S*
- Offset continues to *C* with frequency offset due to error φ
- If $\varepsilon \le x$, then $R \ge \varphi + S$ and $\sigma \le \left(\frac{1}{\varphi} + \frac{1}{R - \varphi} \right) x$
- For ε = 100 μs, φ = 200 PPM, *S* = 200 PPM, this requires $R \ge 400$ PPM and $\sigma \le 1$ s
- These are almost completely eliminated using kernel discipline

Cumulative distribution function of network latencies



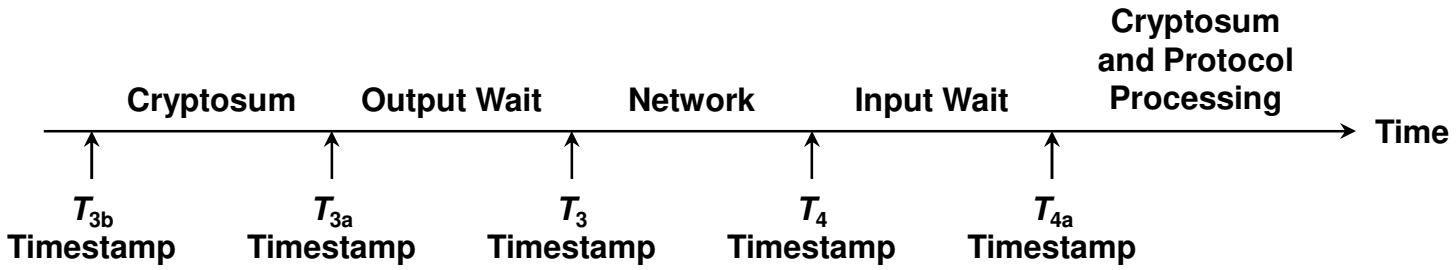
- This cumulative distribution function is from the same day as the time offset slide
  - The rightmost curve represents raw offsets received over the network.
  - The left curve represents the offsets after the clock filter algorithm.

CDF in log-log coordinates – long term



- These data are from other sources
  - The interesting observation is that these lines are almost straight, but with different slope.
  - The awesome fact is they keep going….

Latencies in the operating system and network



- We want T_3 and T_4 timestamps for accurate network timing
  - If output wait is small, T_{3a} is good approximation to T_3
  - T_{3a} can't be included in message after cryptosum is calculated, but can be sent in next message; if not, use T_{3b} as best approximation to T_3
  - T_{4a} is captured at soft-queue interrupt time, so is a fairly good estimator for T_4.
- Largest error is usually cryptosum and output wait
  - With software timestamping, T_3 is captured upon return from the send-packet routine, typically 200 μs after T_{3a}.
  - With interleaved protocol, T_3 is transmitted in the next packet.
  - See http://www.eecis.udel.edu/~mills/onwire.html and related briefing.

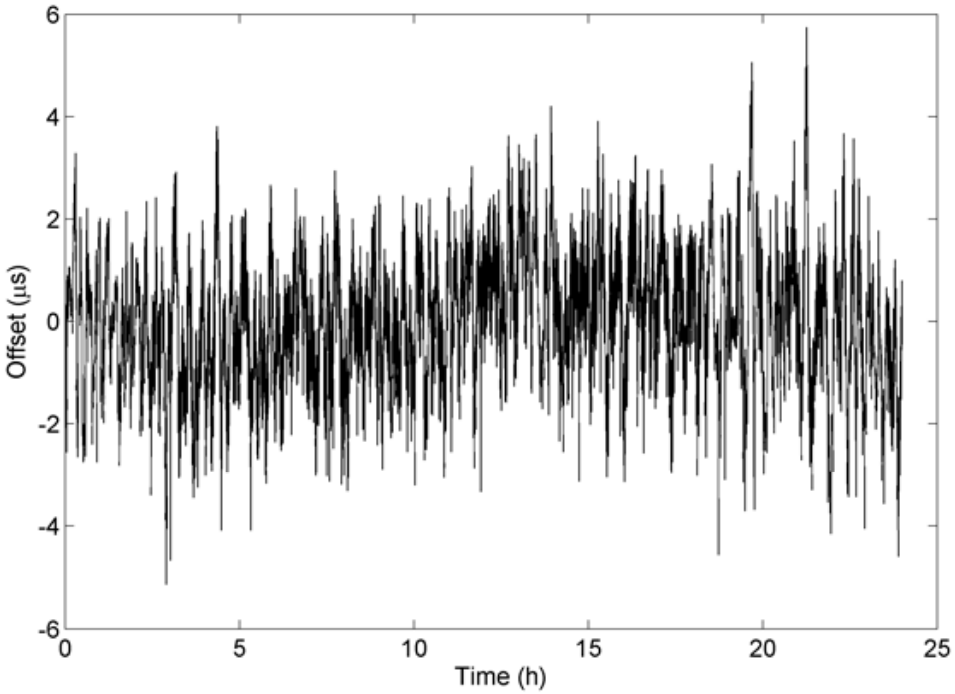
Measured latencies with software interleaved timestamping

- The interleaved protcool captures T_{3b} before the message digest and T_3 after the send-packet routine. The difference varies from 16 µs for a dual-core, 2.8 GHz Pentium 4 running FreeBSD 5.1 to 1100 µs for a Sun Blade 1500 running Solaris 10.
- On two identical Pentium machines in symmetric mode, the measured output delay T_{3b} to T_3 is 16 µs and interleaved delay 2x T_3 to T_{4a} is 90-300 µs . Four switch hops at 100 Mb accounts for 40 µs, which leaves 25-130 µs at each end for input delay. The RMS jitter is 30-50 µs.
- On two identical UltraSPARC machines running Solaris 10 in symmetric mode, the measured output delay T_{3b} to T_3 is 160 µs and interleaved delay 2x T_3 to T_{4a} is 390 µs. Four switch hops accounts for 40 µs, which leaves about 175 µs at each end for input delay. The RMS jitter is 40-60 µs.
- A natural conclusion is that most of the jitter is contributed by the network and input delay.

So, how well does it work?

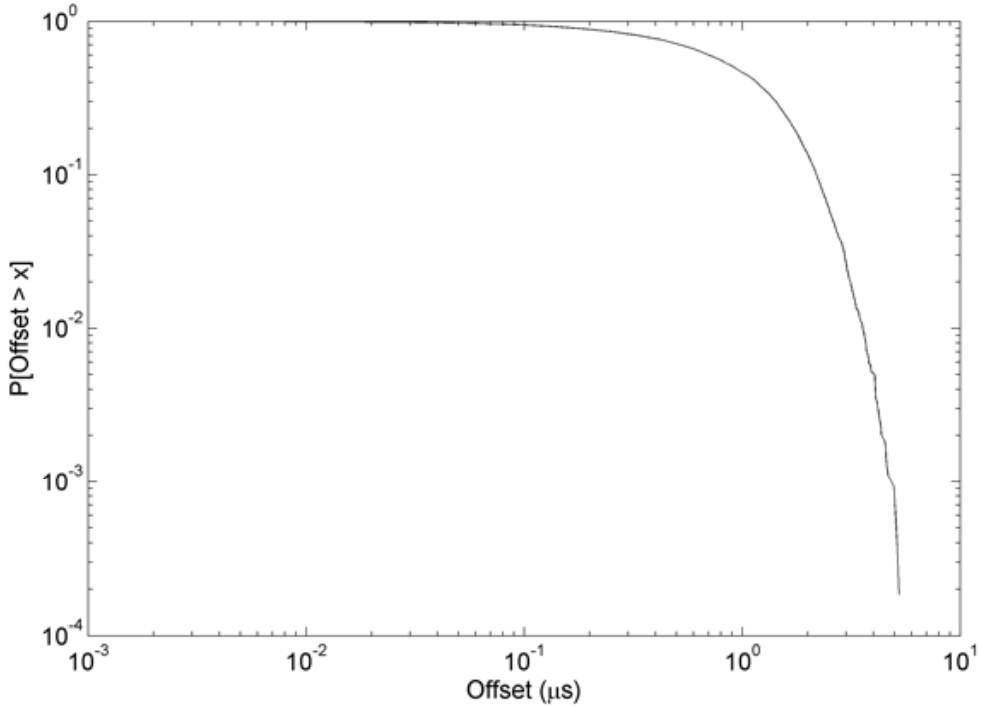
- We measure the max, mean and standard deviation over one day
  - The mean is an estimator of the offset produced by the clock discipline, which is essentially a lowpass filter.
  - The standard deviation is a estimator for jitter produced by the clock filter.
- Following are three scenarios with modern machines and Ethernets
  - The best we can do using the precision time kernel and a PPS signal from a GPS receiver. Expect residual errors in the order of 2 μs dominated by hardware and operating system jitter.
  - The best we can do using a workstation synchronized to a primary server over a fast LAN using optimum poll interval of 15 s. Expect residual errors in the order of 20 μs dominated by network jitter.
  - The best we can do using a workstation synchronized to a primary server over a fast LAN using typical poll interval of 64 s. Expect errors in the order of 200 μs dominated by oscillato rwander.
- Next order of business is the interleaved on-wire protocol and hardware timestamping. The goal is improving network perfomance to PPS level.

Time characteristis with PPS kernel discipline



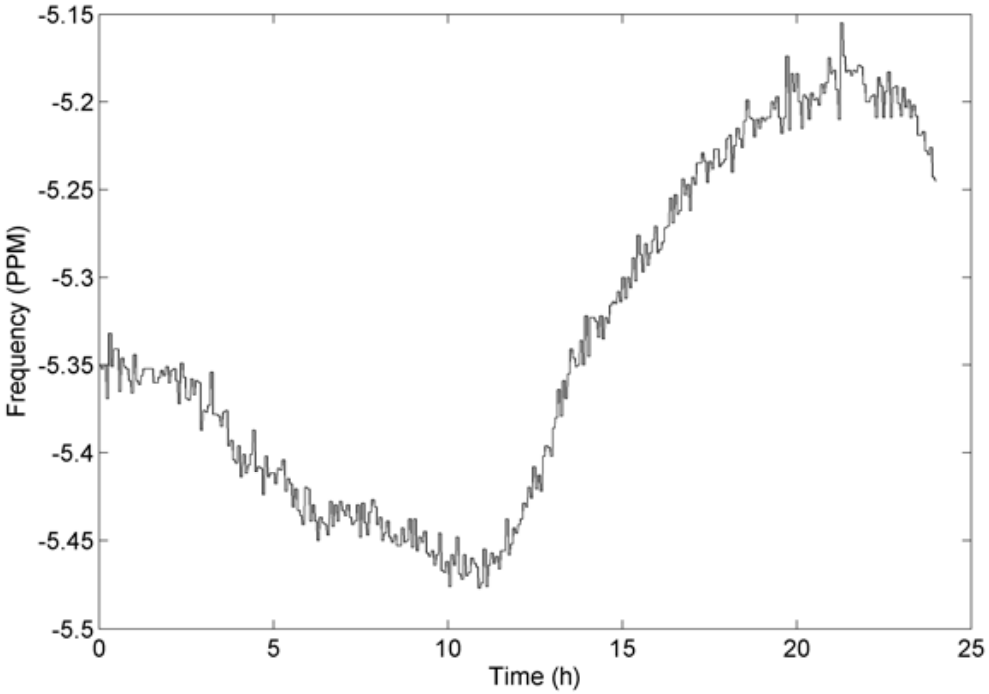
- Machine is Pentium II 300 MHz running FreeBSD 6.1 and synchronized to a GPS receiver via a PPS signal and parallel port
  - Precision nanokernel PPS discipline
  - NTP4 is configured at fixed poll interval 4 (16 s)
  - Behavior appears largely determined by hardware/kernel latencies

Time offset CDF with PPS kernel discipline



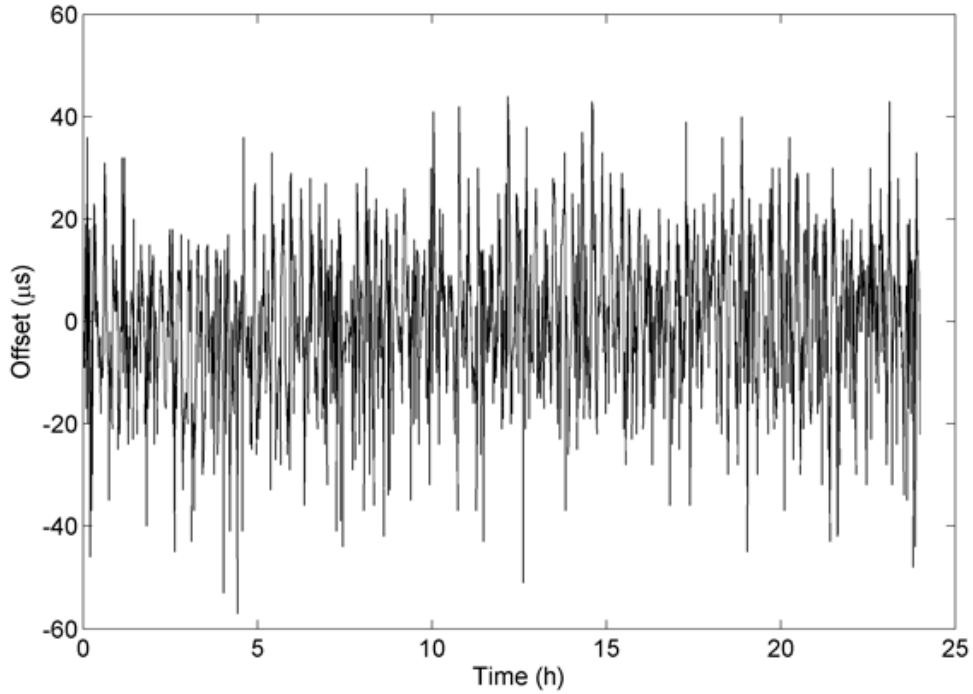
- Same configuration as previous slide
  - Note log-log coordinates
  - Offset statistics: max 5.749 µs, mean -0.039 µs, stdev 1.357 µs

Frequency characteristis with PPS kernel discipline



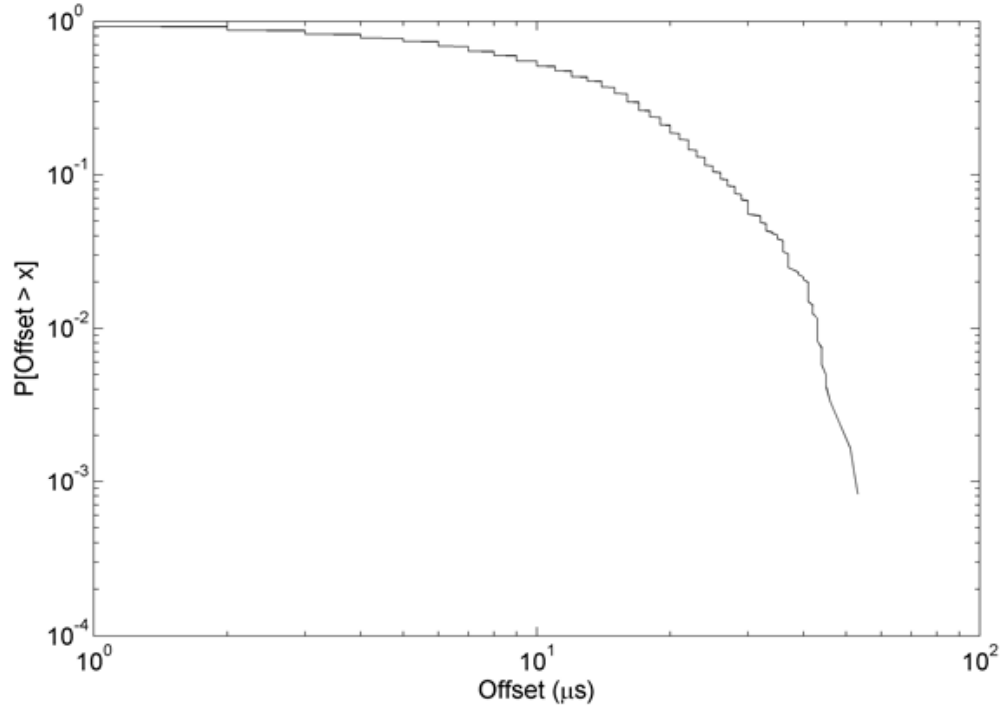
- Same configuration as previous slide
  - Comparison with the time offset characteristics suggest the dominant error contribution is latency jitter rather than frequency discipline.
  - Compare with later data on a typical machine over a fast LAN

Time characteristis with fast LAN and poll 16 s



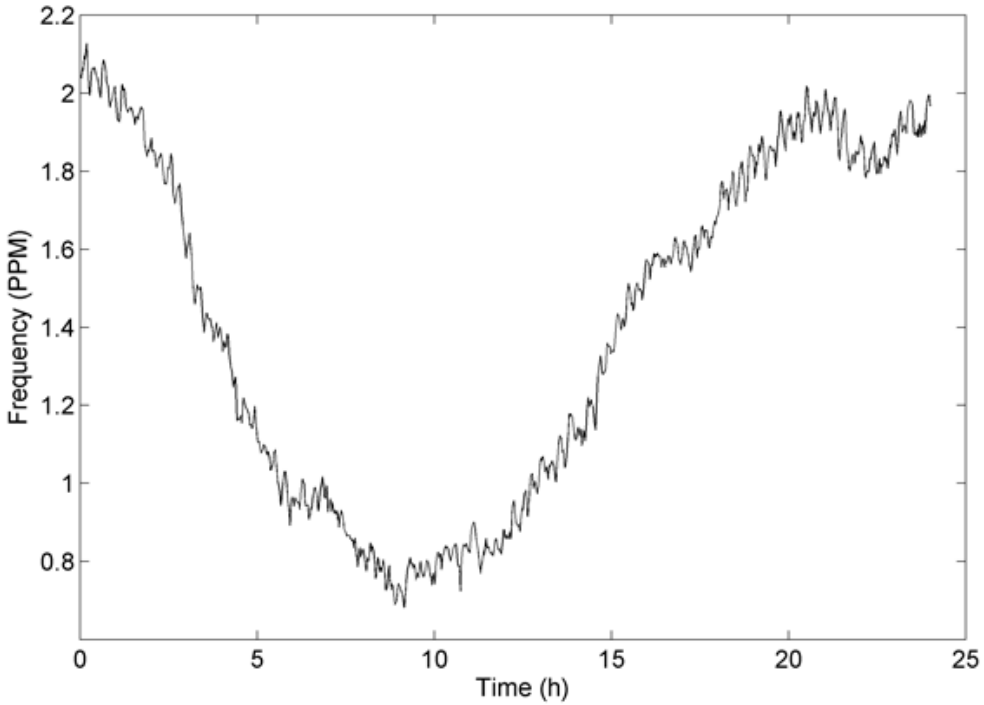
- Machine is UltraSPARC II running Solaris 10 and synchronized to a primary server connected to GPS receiver via a PPS signal
  - NTP4 is configured at fixed poll interval 4 (16 s)
  - Behavior appears largely determined by 100 Mb Ethernet latencies

Time offset CDF with fast LAN and poll 16 s



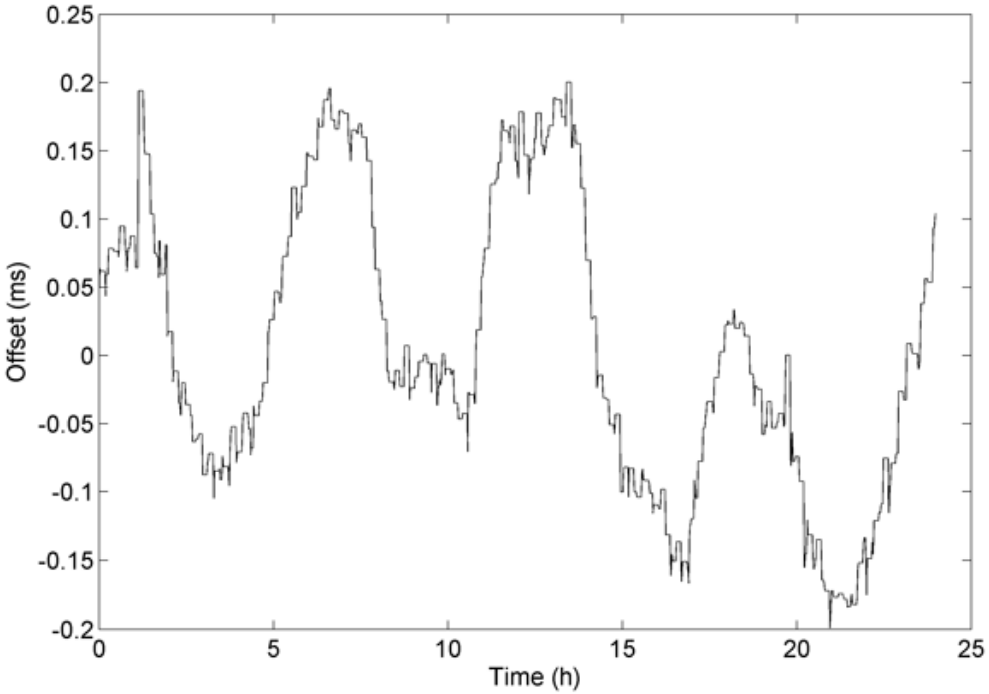
- Same configuration as previous slide
  - Note log-log coordinates
  - Offset statistics: max 57.000 µs mean -0.833 µs stdev 16.078 µs
  - About ten times worse than PPS signal

Frequency characteristis with fast LAN and poll 16 s



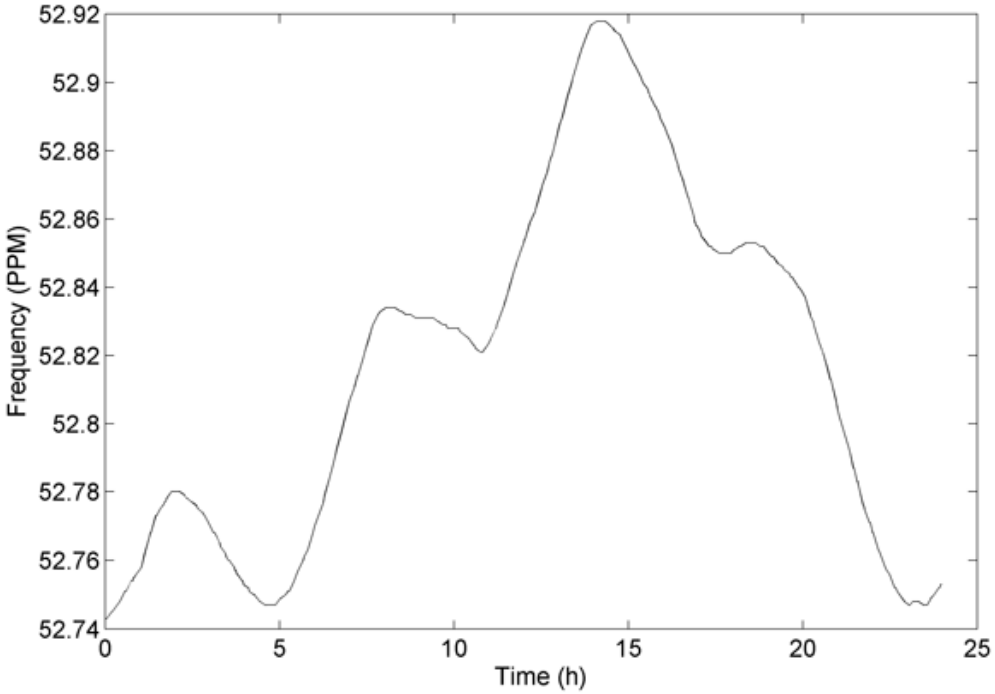
- Same configuration as previous slide
  - Comparison with the time offset characteristics suggest the dominant error contribution is latency jitter rather than frequency discipline.
  - Compare with earlier data with a PPS signal

Time characteristis with fast LAN and poll 64 s



- Machine is Pentium 2.8 GHz running FreeBSD 6.1 and synchronized to a CDMA receiver on a 100 Mb switched Ethernet
  - CDMA receiver claimed accuracy is 10 μs
  - NTP4 is configured at fixed poll interval 6 (64 s)
  - Behavior appears largely determined by oscillator wander

Frequency characteristis with fast LAN and poll 64 s

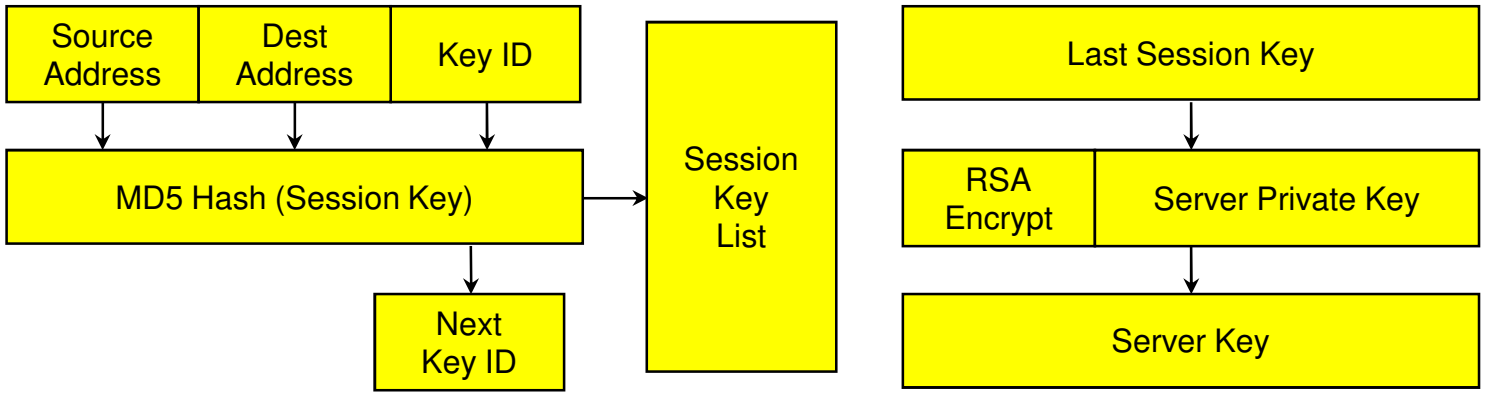


- These data are from the same day as the time offset slide
  - The curve approximates the integral of the time offset data
  - This clearly confirms the errors are primarily due to frequency wander
  - Accuracy improves as the poll interval is reduced, but not below 16 s due increased frequency wander

Not so-quick fixes

- Autokey public key cryptography
  - Avoids errors due to cyrptographic computations
  - See briefing and specification
- Precision time nanokernel
  - Improves time and frequency resolution
  - Avoids sawtooth error
- Improved driver interface
  - Includes median filter
  - Adds PPS driver
- External oscillator/NIC oscillator
  - With interleaved protocol, performance equivalent to IEEE 1588
  - LORAN C receiver and precision clock source

Avoid inline public-key algorithms: the *Autokey* protocol

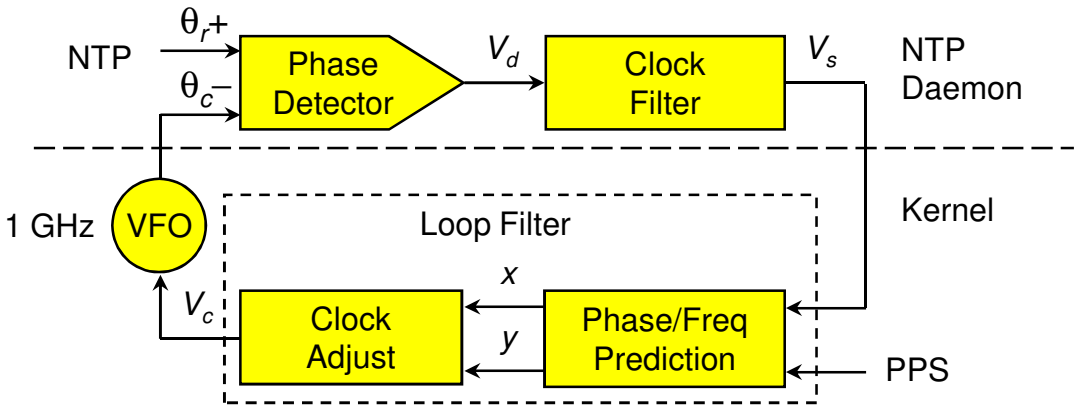


- Server rolls a random 32-bit seed as the initial key ID
- Server generates a session key list using repeated MD5 hashes
- Server encrypts the last key using RSA and its private key to produce the initial server key and provides it and its public key to all clients
- Server uses the session key list in reverse order, so that clients can verify the hash of each key used matches the previous key
- Clients can verify that repeated hashes will eventually match the decrypted initial server key

Kernel modifications for nanosecond resolution

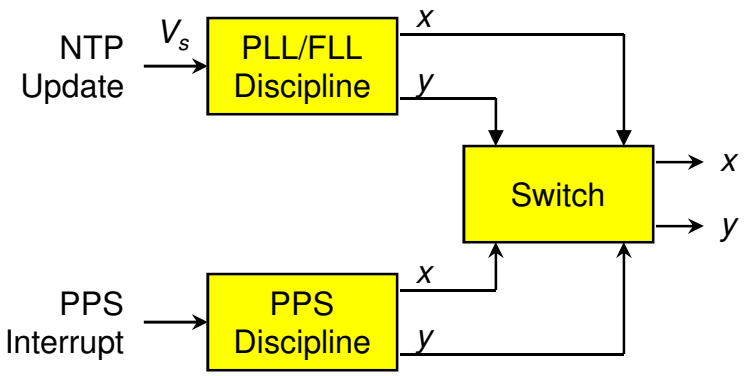
- *Nanokernel* package of routines compiled with the operating system kernel
- Represents time in nanoseconds and fraction, frequency in nanoseconds per second and fraction
- Implements nanosecond system clock variable with either microsecond or nanosecond kernel native time variables
- Uses native 64-bit arithmetic for 64-bit architectures, double-precision 32-bit macro package for 32-bit architectures
- Includes two new system calls `ntp_gettime()` and `ntp_adjtime()`
- Includes new system clock read routine with nanosecond interpolation using process cycle counter (PCC)
- Supports run-time tick specification and mode control
- Guaranteed monotonic for single and multiple CPU systems

NTP clock discipline with nanokernel assist



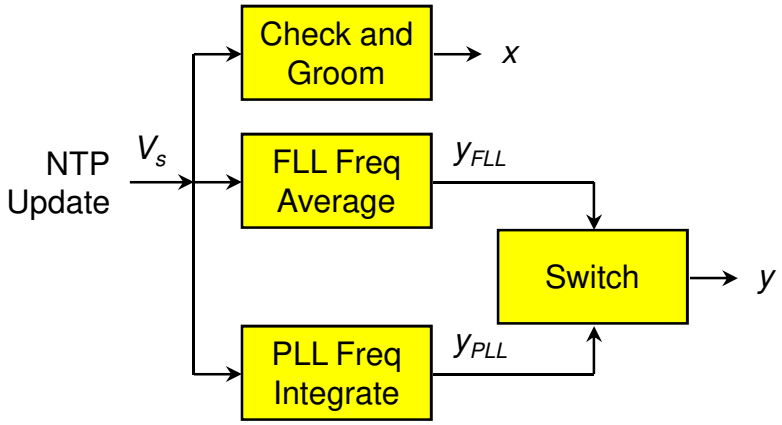
- Type II, adaptive-parameter, hybrid phase/frequency-lock loop disciplines variable frequency oscillator (VFO) phase and frequency
- NTP daemon computes phase error $V_d = \theta_r - \theta_o$ between source and VFO, then grooms samples to produce time update V_s
- Loop filter computes phase *x* and frequency *y* corrections and provides new adjustments V_c at 1-s intervals
- VFO frequency adjusted at each hardware tick interrupt

Nanokernel phase/frequency prediction



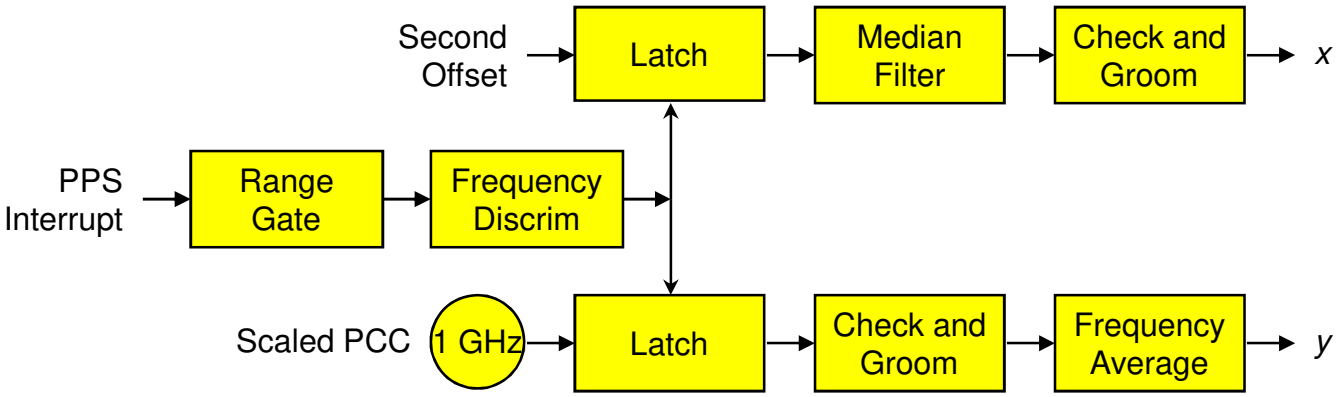
- PLL/FLL discipline predicts phase *x* and frequency *y* at averaging intervals from 1 s to over one day.
- PPS discipline predicts *x* and *y* at averaging intervals from 4 s to 128 s, depending on nominal Allan intercept.
- On overflow of the clock second, new values for time θ and frequency ϕ offset are calculated.
- Phase adjustment $\alpha\theta + \phi$ is added to system clock for $\alpha < 1$ at every tick interrupt, then θ is reduced by $(1 - \alpha)\theta$.

NTP phase and frequency discipline



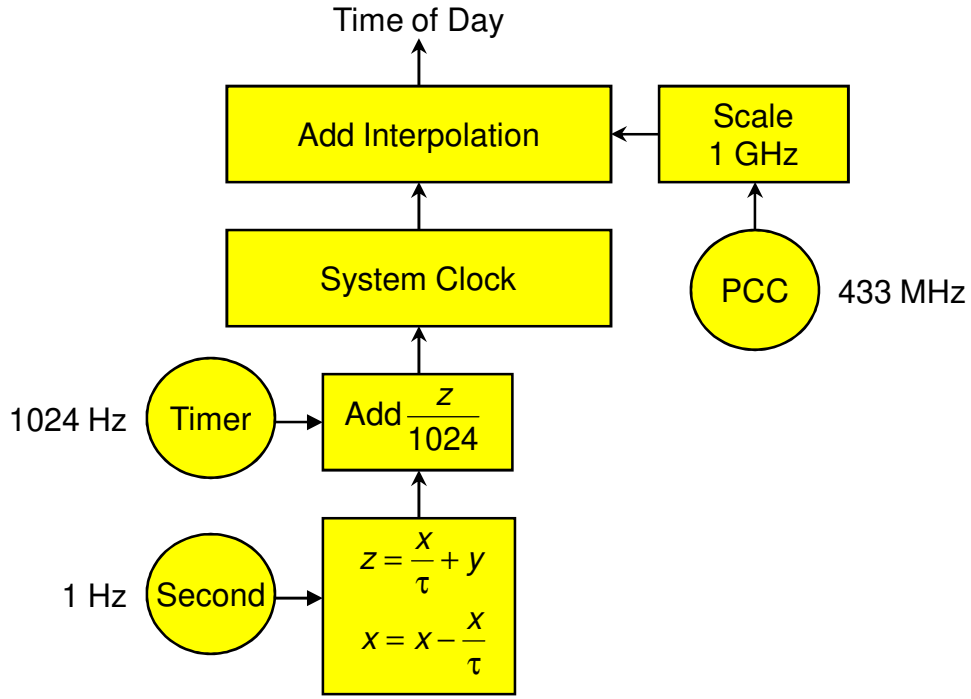
- *x* is the phase correction initially set at the update value.
- y_{FLL} is the frequency prediction computed as the average of past update differences.
- y_{PLL} is the frequency prediction computed as the integral of past update values.
- The switch controlled by the API selects which of y_{FLL} or y_{PLL} are used.

PPS phase and frequency discipline



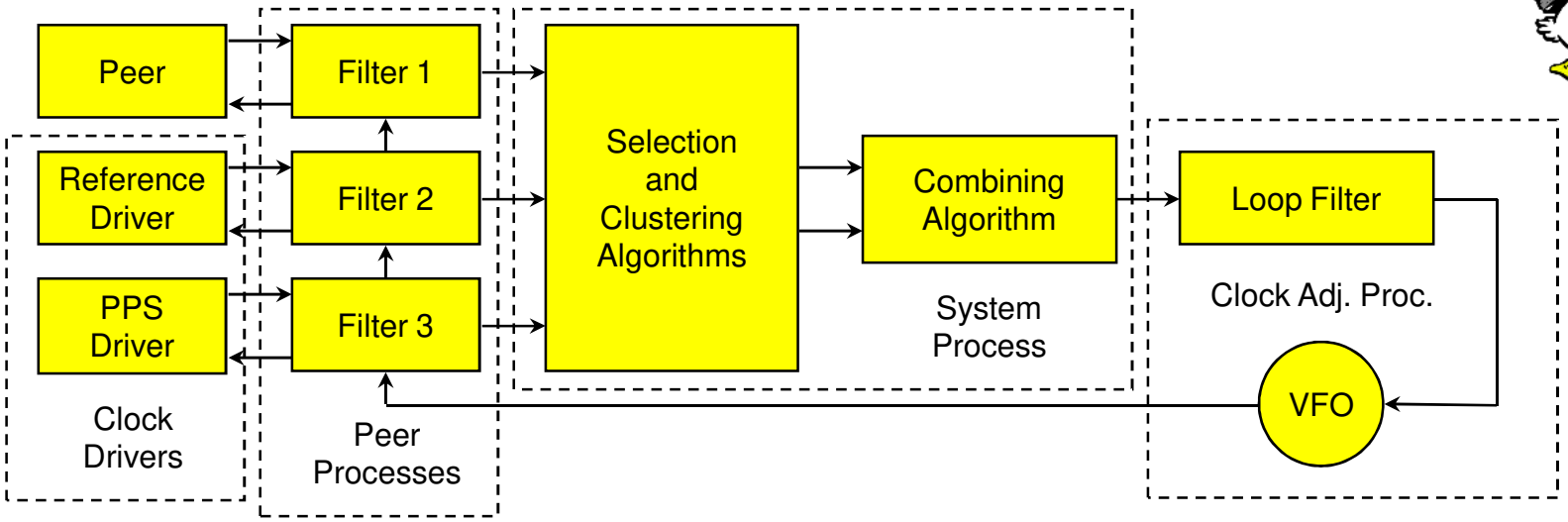
- Phase and frequency disciplined separately - phase from system clock second offset, frequency from processor cycle counter (PCC)
- Frequency discriminator rejects noise and invalid signals
- Median filter rejects sample outlyers and provides error statistic
- Check and groom rejects popcorn spikes and clamps outlyers
- Phase offsets exponentially averaged with variable time constant
- Frequency offsets averaged over variable interval

Nanosecond clock



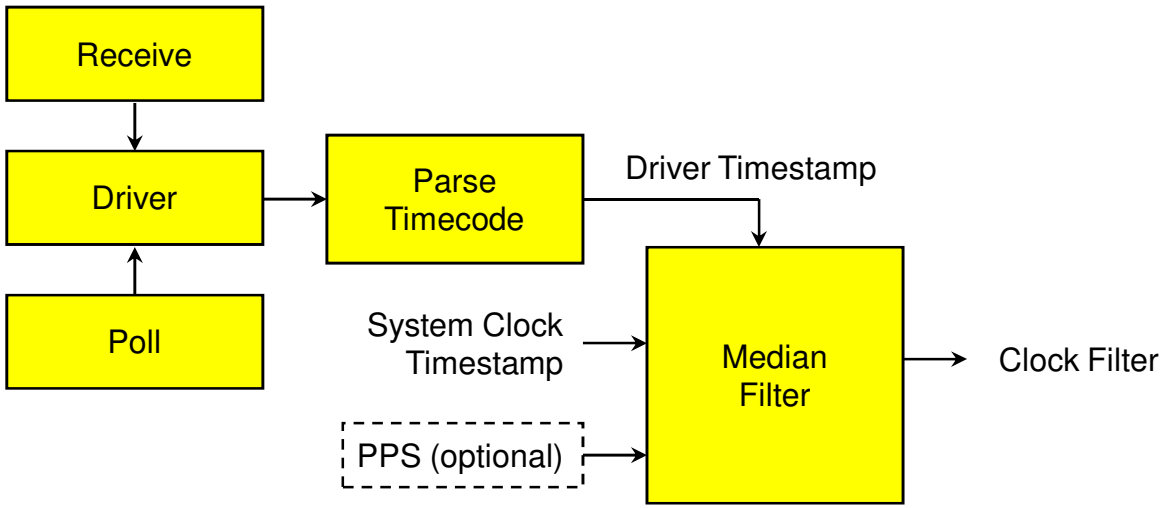
- Phase *x* and frequency *y* are updated by the PLL/FLL or PPS loop.
- At the second overflow increment *z* is calculated and *x* reduced by the time constant.
- The increment is amortized over the second at each tick interrupt.
- Time between ticks is interpolated from the PCC scaled to 1 GHz.

Reference clock drivers



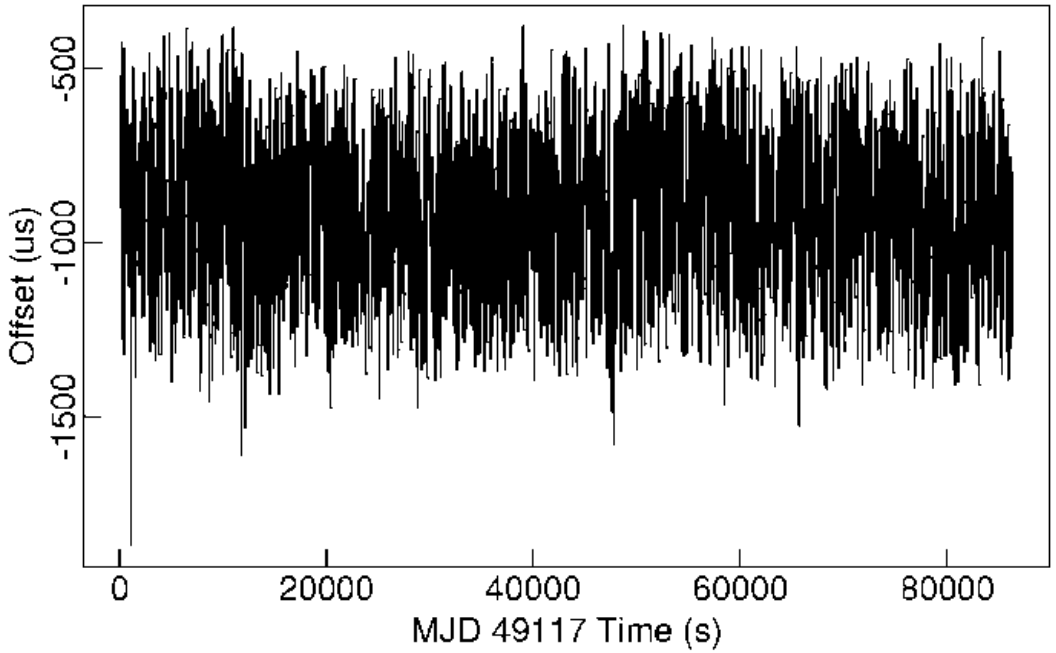
- Reference clock drivers work just like NTP peers.
  - Active drivers produce timecode message in response to poll message.
  - Passive drivers provide timecode registers that can be read by poll routine.
- PPS driver augments prefer peer for precision time.
  - Offset only within the second; seconds numbering must be provided by reference driver or NTP peer.
  - PPS believed only if prefer peer correct and within 128 ms.

Reference clock driver interface



- Driver timecode is read either by timecode message interrupt or poll routine.
  - Timecode and associated data are parsed according to specific format.
  - Offset is computed between driver timestamp and system clock timestamp.
  - Offsets accumulate in median filter shift register until processed and sent to clock filter..
- Optional PPS signal (PPS driver only) provides offset in second.

Minimize effects of serial port hardware and driver jitter

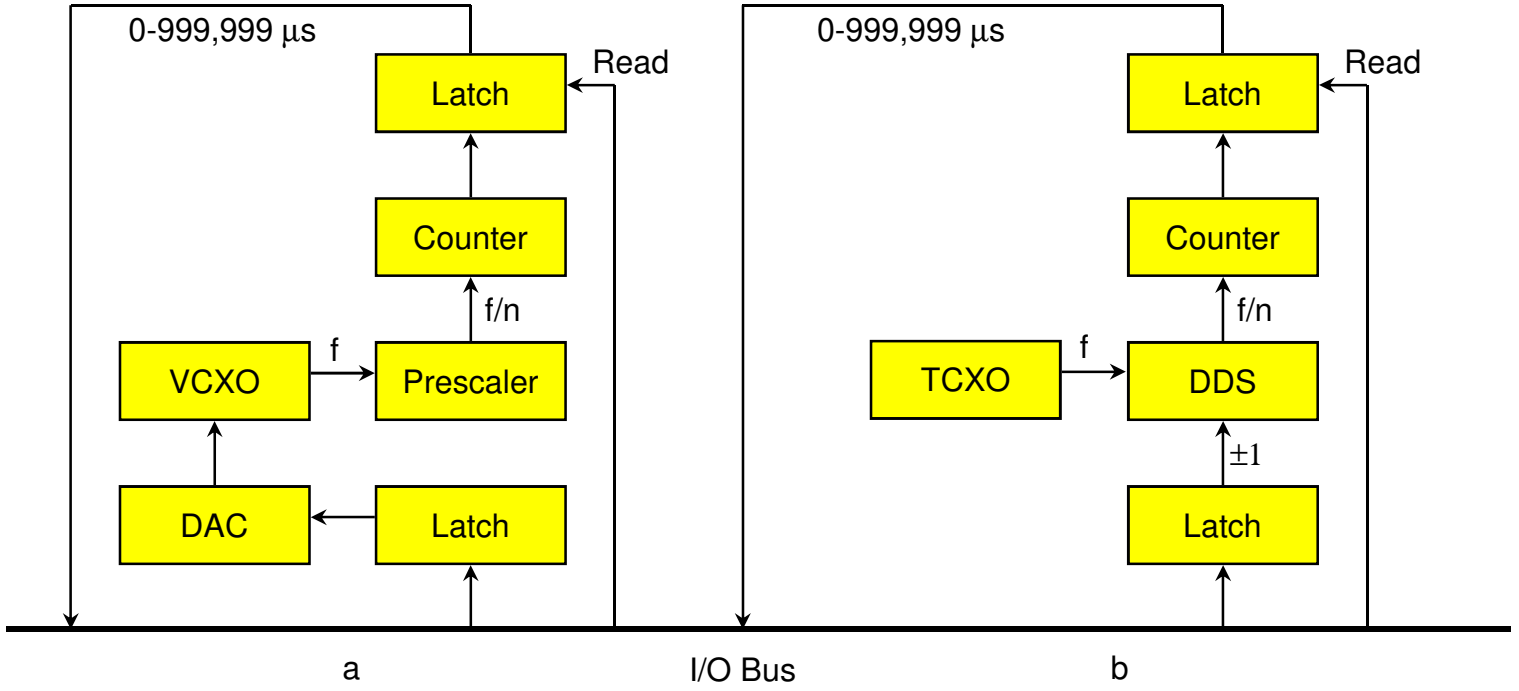


- Graph shows raw jitter of millisecond timecode and 9600-bps serial port
  - Additional latencies from 1.5 ms to 8.3 ms on SPARC IPC due to software driver and operating system; rare latency peaks over 20 ms
  - Latencies can be minimized by capturing timestamps close to the hardware
  - Jitter is reduced using median/trimmed-mean filter of 60 samples
  - Using on-second format and filter, residual jitter is less than 50 µs

Precision time and frequency sources

- KSI/Odetics TPRO IRIG-B SBus interface
  - Provides direct-reading microsecond clock in BCD format
  - Synchronized to GPS receiver using IRIG-B signal
  - Supported both as an NTP driver and as kernel system clock
  - Stabilizes time to 1 μs and frequency to 0.1 PPM
- Precision oven-stabilized system clock
  - SBus memory-mapped interface
  - Provides direct-reading microsecond clock in Unix timeval format
  - Supported as kernel system clock
  - Stabilizes time via radio or NTP and frequency to .005 PPM
- PPS discipline
  - Driver or kernel interface via modem control line
  - Stabilizes frequency to .001 PPM relative to external 1-PPS source
  - Stabilizes time within 1 μs with seconds numbered by NTP

Hardware clock discipline

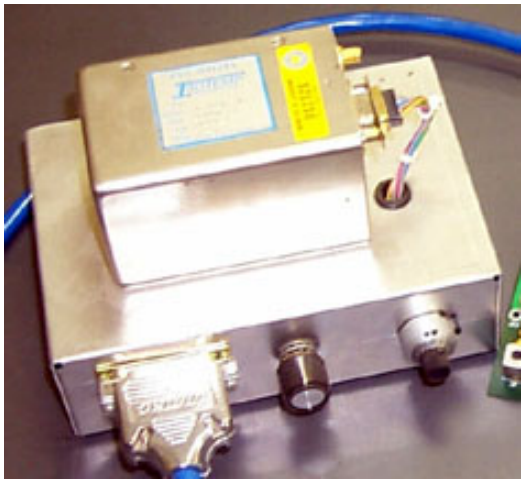


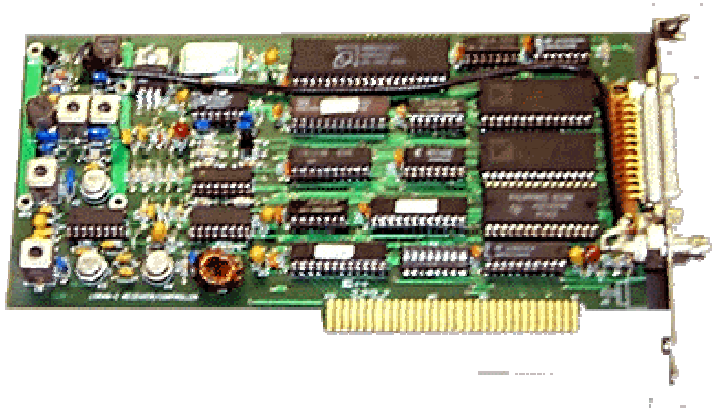
- Analog (a) and digital (b) frequency discipline methods
  - Analog method uses voltage-controlled low-frequency oscillator.
  - Digital method uses direct digital synthesis and high-frquency oscillator.
- Either method could be used in a NIC or bus peripheral

Gadget Box PPS interface

- Used to interface PPS signals from GPS receiver or cesium oscillator
  - Pulse generator and level converter from rising or falling PPS signal edge
  - Simulates serial port character or stimulates modem control lead
- Also used to demodulate timecode broadcast by CHU Canada
  - Narrowband filter, 300-baud modem and level converter
  - The NTP software includes an audio driver that does the same thing

LORAN-C timing receiver





- Inexpensive second-generation bus peripheral for IBM 386-class PC with oven-stabilized external master clock oscillator
  - Includes 100-kHz analog receiver with D/A and A/D converters
  - Functions as precision oscillator with frequency disciplined to selected LORAN-C chain within 200 ns of UTC(LORAN) and 10^{-10} stability
  - PC control program (in portable C) simultaneously tracks up to six stations from the same LORAN-C chain
- Intended to be used with NTP to resolve inherent LORAN-C timing ambiguity

Further information

- NTP home page http://www.ntp.org
  - Current NTP Version 3 and 4 software and documentation
  - FAQ and links to other sources and interesting places
- David L. Mills home page http://www.eecis.udel.edu/~mills
  - Papers, reports and memoranda in PostScript and PDF formats
  - Briefings in HTML, PostScript, PowerPoint and PDF formats
  - Collaboration resources hardware, software and documentation
  - Songs, photo galleries and after-dinner speech scripts
- Udel FTP server: ftp://ftp.udel.edu/pub/ntp
  - Current NTP Version software, documentation and support
  - Collaboration resources and junkbox
- Related projects http://www.eecis.udel.edu/~mills/status.htm
  - Current research project descriptions and briefings