TSCLOCK: A LOW COST, ROBUST, ACCURATE SOFTWARE CLOCK FOR NETWORKED COMPUTERS

Darryl Veitch
d vejitch@ee.unimelb.edu.au
Collaboration with Julien Ridoux

CUBIN, Department of Electrical & Electronic Engineering
The University of Melbourne

Google Tech Talk, Googleplex CA, July 23, 2007
WE NEED CLOCKS

THE OBVIOUS

- Clocks used everywhere in networking and computing
- Network measurement (active and passive)
- Real-time services

THE FUTURE

- Tighter server integration
- Coordination of simultaneous multiple connections
- Latency a fundamental constraint in distributed services/computing
  Example: Virtual world realism in distributed games
WE NEED CLOCKS

THE OBVIOUS

- Clocks used everywhere in networking and computing
- Network measurement (active and passive)
- Real-time services

THE FUTURE

- Tighter server integration
- Coordination of simultaneous multiple connections
- Latency a fundamental constraint in distributed services/computing
  Example: Virtual world realism in distributed games
AND WANT GOOD CLOCKS

RELIABILITY/ROBUSTNESS

- Can’t build tight systems unless have bounds
- Status quo (ntpd) can be ‘good’ (1ms), bad (20ms), outrageous (100’s ms)
- Measurement community pushed to GPS synchronized capture cards

ACCURACY

- Greater accuracy → wider range of applications
- Status quo (ntpd) limited to 1ms, if all is well..
- Constant error versus ‘jitter’, absolute time versus time differences

AFFORDABILITY

- Use existing hardware (oscillator(s) in PCs)
- Network based synchronization cheap and convenient
- But GPS also cheap, right?
AND WANT GOOD CLOCKS

RELIABILITY/ROBUSTNESS

- Can’t build tight systems unless have bounds
- Status quo (ntpd) can be ‘good’ (1ms), bad (20ms), outrageous (100’s ms)
- Measurement community pushed to GPS synchronized capture cards

ACCURACY

- Greater accuracy → wider range of applications
- Status quo (ntpd) limited to 1ms, if all is well..
- Constant error versus ‘jitter’, absolute time versus time differences

AFFORDABILITY

- Use existing hardware (oscillator(s) in PCs)
- Network based synchronization cheap and convenient
- But GPS also cheap, right?
AND WANT GOOD CLOCKS

RELIABILITY/ROBUSTNESS

• Can’t build tight systems unless have bounds
• Status quo (ntpd) can be ‘good’ (1ms), bad (20ms), outrageous (100’s ms)
• Measurement community pushed to GPS synchronized capture cards

ACCURACY

• Greater accuracy → wider range of applications
• Status quo (ntpd) limited to 1ms, if all is well..
• Constant error versus ‘jitter’, absolute time versus time differences

AFFORDABILITY

• Use existing hardware (oscillator(s) in PCs)
• Network based synchronization cheap and convenient
• But GPS also cheap, right?
AND WANT GOOD CLOCKS

RELIABILITY/ROBUSTNESS

- Can’t build tight systems unless have bounds
- Status quo (ntpd) can be ‘good’ (1ms), bad (20ms), outrageous (100’s ms)
- Measurement community pushed to GPS synchronized capture cards

ACCURACY

- Greater accuracy → wider range of applications
- Status quo (ntpd) limited to 1ms, if all is well..
- Constant error versus ‘jitter’, absolute time versus time differences

AFFORDABILITY

- Use existing hardware (oscillator(s) in PCs)
- Network based synchronization cheap and convenient
- But GPS also cheap, WRONG!
  Can cost 12 months and $10,000 to instrument a machine room
THE TSC CLOCK

DESIGN

- Re-engineered from scratch
- Built on
  - time-scale aware abstraction of oscillator performance
  - separate and decoupled treatment of rate and absolute time
  - RTT based delay filtering
  - feedforward not feedback
- Use oscillator driving CPU, accessible via TSC register
  (commonly available, high resolution, hardware updating, fast read)

PROVIDES

- Very high robustness
- Accuracy an order of magnitude higher than ntpd (or more)
- Separate Absolute and Difference clocks
THE TSC Clock

Design

- Re-engineered from scratch
- Built on
  - time-scale aware abstraction of oscillator performance
  - separate and decoupled treatment of rate and absolute time
  - RTT based delay filtering
  - feedforward not feedback
- Use oscillator driving CPU, accessible via TSC register
  (commonly available, high resolution, hardware updating, fast read)

Provides

- Very high robustness
- Accuracy an order of magnitude higher than ntpd (or more)
- Separate Absolute and Difference clocks
THE TSC CLO CK

DESIGN

- Re-engineered from scratch  **NOT just the fact that TSC is used!**
- Built on
  - time-scale aware abstraction of oscillator performance
  - separate and decoupled treatment of rate and absolute time
  - RTT based delay filtering
  - feedforward not feedback
- Use oscillator driving CPU, accessible via TSC register
  (commonly available, high resolution, hardware updating, fast read)

PROVIDES

- Very high robustness
- Accuracy an order of magnitude higher than **ntpd** (or more)
- Separate Absolute and **Difference** clocks
THE TSCCLOCK

**DESIGN**

- Re-engineered from scratch  **NOT just the fact that TSC is used!**
- Built on
  - time-scale aware abstraction of oscillator performance
  - separate and decoupled treatment of rate and absolute time
  - RTT based delay filtering
  - feedforward not feedback
- Use oscillator driving CPU, accessible via TSC register
  (commonly available, high resolution, hardware updating, fast read)

**PROVIDES**

- Very high robustness
- Accuracy an order of magnitude higher than `ntpd` (or more)
- Separate **Absolute** and **Difference** clocks
**OFFSET, SKEW AND DRIFT**

- **Offset:** error $\theta(t) = C(t) - t$ of clock $C(t)$ at time $t$
- **Skew:** error in rate. E.g.: $\theta(t) = C + \gamma t$ (Simple Skew Model (SKM))
- **Drift:** non-linear evolution of $\theta(t)$
**Offset, Skew and Drift**

**Offset:** error $\theta(t) = C(t) - t$ of clock $C(t)$ at time $t$

**Skew:** error in rate. E.g.: $\theta(t) = C + \gamma t$ (Simple Skew Model (SKM))

**Drift:** non-linear evolution of $\theta(t)$
THE ROLE OF TIME SCALE

Laboratory: \( \bar{p} = 1.82263812 \times 10^{-9} \) (548.65527 Mhz)

Machine Room: \( \bar{p} = 1.82263832 \times 10^{-9} \) (548.65521 Mhz)

Short timescales: Simple Skew Model applies
Large timescales: unpredictable drift must be tracked
THE ROLE OF TIME SCALE

Laboratory: \( \bar{p} = 1.82263812 \times 10^{-9} \) (548.65527 Mhz)

Machine Room: \( \bar{p} = 1.82263832 \times 10^{-9} \) (548.65521 Mhz)

Short timescales: Simple Skew Model applies

Large timescales: unpredictable drift must be tracked
Oscillator Stability

Allan deviation: scale dependent rate errors: \[ y_\tau(t) = \frac{\theta(t + \tau) - \theta(t)}{\tau} \]

- SKM holds for \( \tau^* = 1000 \) [sec], (here TSC period \( p \) meaningful)
- Average rate error upper bounded by 0.1 PPM no matter the scale
**Oscillator Stability**

Allan deviation: scale dependent rate errors: 

$$y_\tau(t) = \frac{\theta(t + \tau) - \theta(t)}{\tau}$$

- SKM holds for $\tau^* = 1000$ [sec], (here TSC period $p$ meaningful)
- Average rate error upper bounded by $0.1$ PPM no matter the scale
The Need for Difference Clocks

Stable rate \((p \text{ good to } 10^{-7})\) implies accurate \(\Delta(t)\) measurement:

Example: error in RTT of 100ms just 10ns

However an absolute clock \(C_a(t)\) requires constant correction to negate drift:

- To synchronize \(C_a(t)\), could
  - continuously modulate rate \((ntpd \text{ uses } \pm 500 \text{ PPM band})\)
  - regularly add corrective jumps
- Either way, rate is disturbed
- Effect large! since drift estimation inherently difficult

Result: high native stability degraded by unbounded amount!
THE NEED FOR DIFFERENCE CLOCKS

Stable rate (\( p \) good to \( 10^{-7} \)) implies accurate \( \Delta(t) \) measurement:

**Example**: error in RTT of 100ms just 10ns

However an absolute clock \( C_a(t) \) requires constant correction to negate drift:

- To synchronize \( C_a(t) \), could
  - continuously modulate rate (\( ntpd \) uses \( \pm 500 \) PPM band)
  - regularly add corrective jumps
- Either way, rate is disturbed
- Effect large! since drift estimation inherently difficult

**Result**: high native stability degraded by unbounded amount!
**THE NEED FOR DIFFERENCE CLOCKS**

Stable rate (\( p \) good to \( 10^{-7} \)) implies accurate \( \Delta(t) \) measurement:

**Example:** error in RTT of 100ms just 10ns

However an absolute clock \( C_a(t) \) requires constant correction to negate drift:

- To synchronize \( C_a(t) \), could
  - continuously modulate rate (\( ntpd \) uses \( \pm 500 \) PPM band)
  - regularly add corrective jumps
- Either way, rate is disturbed
- Effect large! since drift estimation inherently difficult

**Result:** high native stability degraded by **unbounded** amount!
A DUAL CLOCK ARCHITECTURE

Foundation is the *uncorrected clock*: \[ C_u(t) = \bar{p} \cdot TSC(t) + K \]

**DIFFERENCE CLOCK**

- Used for time differences below \( \tau^* \sim 1000 \text{ sec} \)
- \( C_d(t) = C_u(t) \)  Example: \( C_d(t_2) - C_d(t_1) = \bar{p} \cdot (TSC(t_2) - TSC(t_1)) \)
- Immune from errors in drift correction
- Use: RTTs, delay jitter, execution time, local event ordering ..

**ABSOLUTE CLOCK**

- Absolute timestamps (and time differences above \( \tau^* \))
- \( C_a(t) = C_u(t) - \hat{\theta}(t) \)
- Drift correction estimate \( \hat{\theta}(t) \) only applied when clock read
- Use: latency, global event ordering and scheduling ..

Require robust, accurate algorithms for \( \bar{p} \) and \( \hat{\theta} \)
A DUAL CLOCK ARCHITECTURE

Foundation is the uncorrected clock: \[ C_u(t) = \bar{p} \cdot \text{TSC}(t) + K \]

**DIFFERENCE CLOCK**

- Used for time differences below \( \tau^* \sim 1000 \text{ sec} \)
- \( C_d(t) = C_u(t) \)  
  Example: \( C_d(t_2) - C_d(t_1) = \bar{p} \cdot (\text{TSC}(t_2) - \text{TSC}(t_1)) \)
- Immune from errors in drift correction
- Use: RTTs, delay jitter, execution time, local event ordering ..

**ABSOLUTE CLOCK**

- Absolute timestamps (and time differences above \( \tau^* \))
- \( C_a(t) = C_u(t) - \hat{\theta}(t) \)
- Drift correction estimate \( \hat{\theta}(t) \) only applied when clock read
- Use: latency, global event ordering and scheduling ..

Require robust, accurate algorithms for \( \bar{p} \) and \( \hat{\theta} \)
A Dual Clock Architecture

Foundation is the *uncorrected clock*: $C_u(t) = \bar{p} \cdot TSC(t) + K$

**Difference Clock**

- Used for time differences below $\tau^* \sim 1000$ sec
- $C_d(t) = C_u(t)$  
  Example: $C_d(t_2) - C_d(t_1) = \bar{p} \cdot (TSC(t_2) - TSC(t_1))$
- Immune from errors in drift correction
- Use: RTTs, delay jitter, execution time, local event ordering ..

**Absolute Clock**

- Absolute timestamps (and time differences above $\tau^*$)
- $C_a(t) = C_u(t) - \hat{\theta}(t)$
- Drift correction estimate $\hat{\theta}(t)$ only applied when clock read
- Use: latency, global event ordering and scheduling ..

Require robust, accurate algorithms for $\bar{p}$ and $\hat{\theta}$
A DUAL CLOCK ARCHITECTURE

Foundation is the uncorrected clock: \( C_u(t) = \bar{p} \cdot \text{TSC}(t) + K \)

**DIFFERENCE CLOCK**

- Used for time differences below \( \tau^* \sim 1000 \) sec
- \( C_d(t) = C_u(t) \)  
  Example: \( C_d(t_2) - C_d(t_1) = \bar{p} \cdot (\text{TSC}(t_2) - \text{TSC}(t_1)) \)
- Immune from errors in drift correction
- Use: RTTs, delay jitter, execution time, local event ordering ..

**ABSOLUTE CLOCK**

- Absolute timestamps (and time differences above \( \tau^* \))
- \( C_a(t) = C_u(t) - \hat{\theta}(t) \)
- Drift correction estimate \( \hat{\theta}(t) \) only applied when clock read
- Use: latency, global event ordering and scheduling ..

Require robust, accurate algorithms for \( \bar{p} \) and \( \hat{\theta} \)
A Client-Server Paradigm

Obtain timestamps \( \{T_{a,i}, T_{b,i}, T_{e,i}, T_{f,i}\} \) from \( i \)-th exchange

- \( \{T_{a,i}, T_{f,i}\} \): host timestamps in TSC counter units
- \( \{T_{b,i}, T_{e,i}\} \): server timestamps in seconds
Choose RTT based filtering, not one-way (using same clock good!)

Round–Trip Times $r_i$ of packet $i$

Model for RTT: $r_i = r + \text{positive random noise}$
Filter using point error: excess over minimum RTT
Naive Rate Synchronization

Wish to exploit the relation $\Delta(t) = \Delta(TSC) \times \bar{p}$

**Naive estimate** based on widely separated packets $j$ and $i$:

$$\hat{p}_{i,j}^\uparrow \equiv \frac{T_{b,i} - T_{b,j}}{T_{a,i} - T_{a,j}}$$

Network delay and timestamping noise $\sim \frac{1}{\Delta(TSC)}$, but errors not bounded.
**RATE SYNCHRONIZATION ALGORITHM**

Use selected naive estimates based on point error threshold

**PROPERTIES**

- Error quickly $< 0.1$ PPM, In 10mins, better than GPS!
- Error reduction (in timestamping, latency) guaranteed by $\Delta(t)$
- Inherently robust to packet loss, congestion, loss of server..
- Based on $\bar{p}$, no local rate estimates
RATE SYNCHRONIZATION ALGORITHM

Use selected naive estimates based on point error threshold

Properties

- Error quickly < 0.1 PPM, In 10mins, better than GPS!
- Error reduction (in timestamping, latency) guaranteed by $\Delta(t)$
- Inherently robust to packet loss, congestion, loss of server..
- Based on $\bar{p}$, no local rate estimates
NAIVE ABSOLUTE SYNCHRONIZATION

Wish to exploit SKM over small scales to measure $\theta(t)$

Naive estimate again ignores network congestion, exploits steady rate over RTT

$$\hat{\theta}_i = \frac{1}{2}(C(t_{a,i}) + C(t_{f,i})) - \frac{1}{2}(T_{b,i} + T_{e,i})$$
ABSOLUTE SYNCHRONIZATION ALGORITHM

Must track, so use all naive estimates, but carefully

ALGORITHM FOR $\hat{\theta}(t)$

- Weighted estimate of naive $\theta_i$’s over SKM window
- Weights very strict, based on RTT quality (if quality very bad, freeze)
- Meaningful sanity check: ignore if hardware rate bound exceeded
THE PATH ASYMMETRY

FUNDAMENTAL AMBIGUITY

Asymmetry $A \equiv d^\uparrow - d^\downarrow$ and $2\theta(t)$ non-unique up to a constant.

IMPACT ON ABSOLUTE CLOCK

- $A$ unknown: generally forced to assume $A = 0$
- However, bounded by minimum RTT: $A \in (-r, r)$
- Create constant errors from $5\mu s$ to $100$’s ms
- Causes jumps when server changed
- $\rightarrow$ Important to use a single, close, server.

IMPACT ON DIFFERENCE CLOCK

- None
- Difference clock can be used to measure $r$
• GPS synchronized DAG card for *external* validation
• GPS synchronized SW and modified kernels for *internal* validation
TESTBED

- GPS synchronized DAG card for *external* validation
  - timestamps accurate to 100ns, but
  - comparison polluted by ‘system noise’
  - splits asymmetry: \( A = A_n + A_h \)
  - allows network component \( A_n \) to be measured
  - host component \( A_h \) can only be bounded, can be >200\( \mu s \)!
- GPS synchronized SW and modified kernels for *internal* validation
• GPS synchronized DAG card for external validation
• GPS synchronized SW and modified kernels for internal validation
  • side by side timestamps cancels noise, but
  • only relative performance measurable, not absolute
A QUICK COMPARISON WITH *ntpd*

**ntpd:** sync’d to stratum-1 NTP server on LAN (broadcast mode)

**TSCclock:** sync’d to stratum-1 NTP server outside LAN
**EXTERNAL VALIDATION: TSC CLOCK VS DAG**

**Server:** Stratum-1 NTP on LAN  
**Polling Period:** 256 sec  
**System Noise:** $\sim 20 \mu s$  
**Asymmetry:** Measured at $36 \mu s$ and removed
INTERNAL VALIDATION: TSCclock vs SW-GPS

Server: Stratum-1 outside LAN
Polling Period: 16 sec
System Noise: ≪ 1 µs
Asymmetry: As before for TSCclock, but SW-GPS component?
**PARAMETER DEPENDENCE**

**window width**

![Graph showing Offset Error vs. \( \tau' / \tau^* \) for window width, with lines representing no local rate and with local rate.

**polling period**

![Graph showing Offset Error vs. Polling Period, with lines representing no local rate and with local rate.]

**Duration:** 32 days  
**Server:** Stratum-1 NTP, 5 hops away, \( r = 0.61 \text{ ms} \)  
**Poll Period:** 16 sec  
**Asymmetry:** \( A_n = 70 \mu s \)  
**Median IQR:** 12\( \mu s \) (corrected for \( A \))  
**IQR:** 15\( \mu s \) (including external validation noise)
**Comparison with nptd**

**Server on LAN**

<table>
<thead>
<tr>
<th>Polling Period [sec]</th>
<th>16</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ms]</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Server outside LAN**

<table>
<thead>
<tr>
<th>Polling Period [sec]</th>
<th>16</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ms]</td>
<td>0</td>
<td>0.1</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Asymmetry:** Same for each clock
DIFFERENCE CLOCK VERSUS GPS
DIFFERENCE CLOCK VERSUS GPS

Now compare if no connectivity with server
DIFFERENCE CLOCK VERSUS GPS

Live $C_d(t)$: med = 0.03 iqr = 0.71 [mus]

Frozen $C_d(t)$: med = 0.19 iqr = 0.741 [mus]
THE SYSTEM

- **API**
  - absolute and difference clock reading
  - mode setting/reading
  - diagnostics

- **Timestamping solution**
  - better with kernel support

- **Synchronization algorithm:**
  - runs as daemon or on command line
  - can store and replay log files

- **Server**
  - no server side solution, yet
  - client compatible with existing NTP servers
  - designed (and recommended) for use with a single server
TIMESTAMPING

KERNEL

USER
TIMESTAMPING

KERNEL

- Packet timestamping:
  - Normal mode: TSCclock works in parallel with SW
  - TSCclock mode: SW also returns $C_a(t)$ transparently

- Other timestamping:
  - TSCclock works in parallel with SW

USER
TIMESTAMPING

KERNEL

- Packet timestamping:
  - Normal mode: TSCclock works in parallel with SW
  - TSCclock mode: SW also returns $C_a(t)$ transparently

- Other timestamping:
  - TSCclock works in parallel with SW

USER

- Packet timestamping:
  - Kernel packet timestamps inferred from userland
  - TSCclock works in parallel with SW
• Ubuntu 6.10 (Edgy)
• Ubuntu 7.04 (Feisty)
• Debian 4.0 (Etch)
• Fedora Core 6
• and soon Fedora Core 7 ...
CONCLUSIONS

- TSCclock: for synchronization over networks
- Currently based on CPU oscillator accessible via TSC register
  - Absolute Clock:
    - far more robust than ntpd
    - order of magnitude more accurate
  - Difference Clock:
    - exceptionally robust
    - not available under ntpd
    - more accurate than standard GPS solution for small time intervals
- Kernel and userland packet timestamping solutions
- Low computational requirements
- Runs as daemon in parallel with ntpd
- Works with existing NTP server network
- Packages written for BSD and popular Linux distributions
CONCLUSIONS

- **TSCclock**: for synchronization over networks
- Currently based on CPU oscillator accessible via TSC register
- **Absolute Clock**:
  - far more robust than *ntpd*
  - order of magnitude more accurate
- **Difference Clock**:
  - exceptionally robust
  - not available under *ntpd*
  - more accurate than standard GPS solution for small time intervals
- **Kernel and userland packet timestamping solutions**
  - Low computational requirements
  - Runs as daemon in parallel with *ntpd*
  - Works with existing NTP server network
  - Packages written for BSD and popular Linux distributions
CONCLUSIONS

- TSCclock: for synchronization over networks
- Currently based on CPU oscillator accessible via TSC register
- Absolute Clock:
  - far more robust than ntpd
  - order of magnitude more accurate
- Difference Clock:
  - exceptionally robust
  - not available under ntpd
  - more accurate than standard GPS solution for small time intervals
- Kernel and userland packet timestamping solutions
- Low computational requirements
- Runs as daemon in parallel with ntpd
- Works with existing NTP server network
- Packages written for BSD and popular Linux distributions
Links

- Publications:
  http://www.cubinlab.ee.unimelb.edu.au/articles

- TSCclock page: