Synchronising Software Clocks on the Internet

Darryl Veitch*

Collaboration with

Satish Babu† and Attila Pásztor↑

Supported by Ericsson Australia

* CUBIN, Dept. of Electrical & Electronic Engineering, University of Melbourne.
† IIT, New Delhi.
↑ Ericsson Hungary R&D, Budapest.

Web Page: *http://www.cubinlab.ee.mu.oz.au/∼darryl*

CUBIN

Motivation

Everyone needs a *Software Clock* :

- Physical support
- Synchronisation (absolute, and rate)
- Timestamping

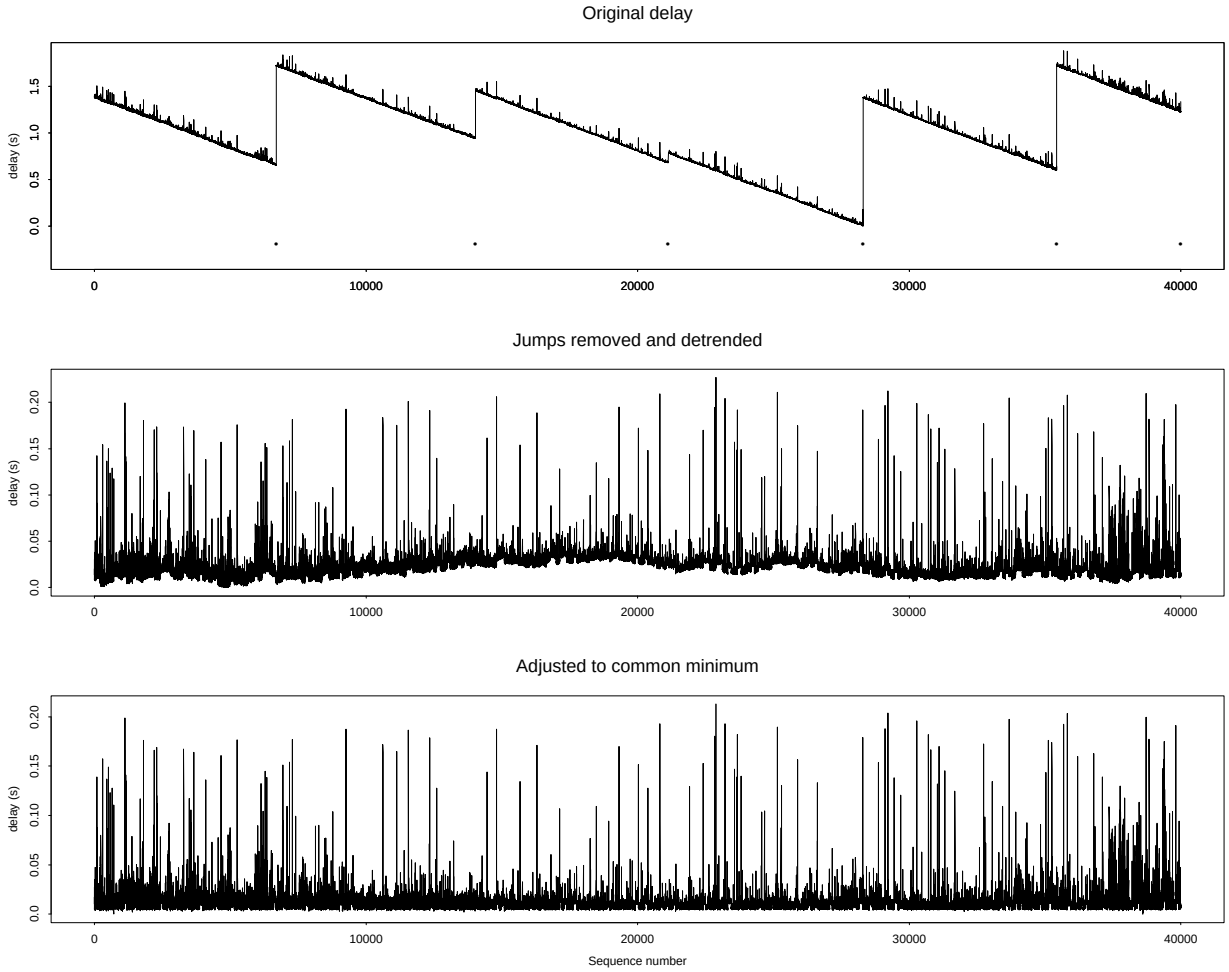
And wants performance :

- Inexpensive (off the shelf hardware)
- Inexpensive, convenient synchronisation (off a network)
- Accurate, and Robust

Widely used SW-NTP solution in PC's not good enough

End-to-End Delay: does this look familiar?

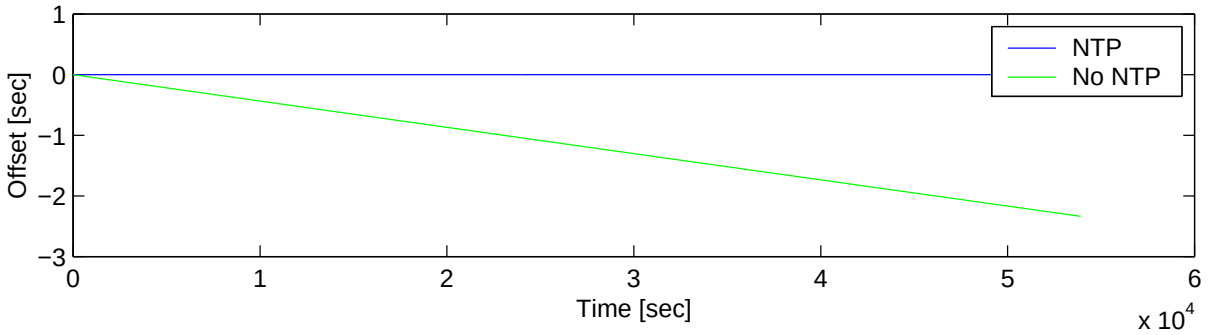
SW-NTP can be bad... ad-hoc data ‘cleaning’ is problematic.



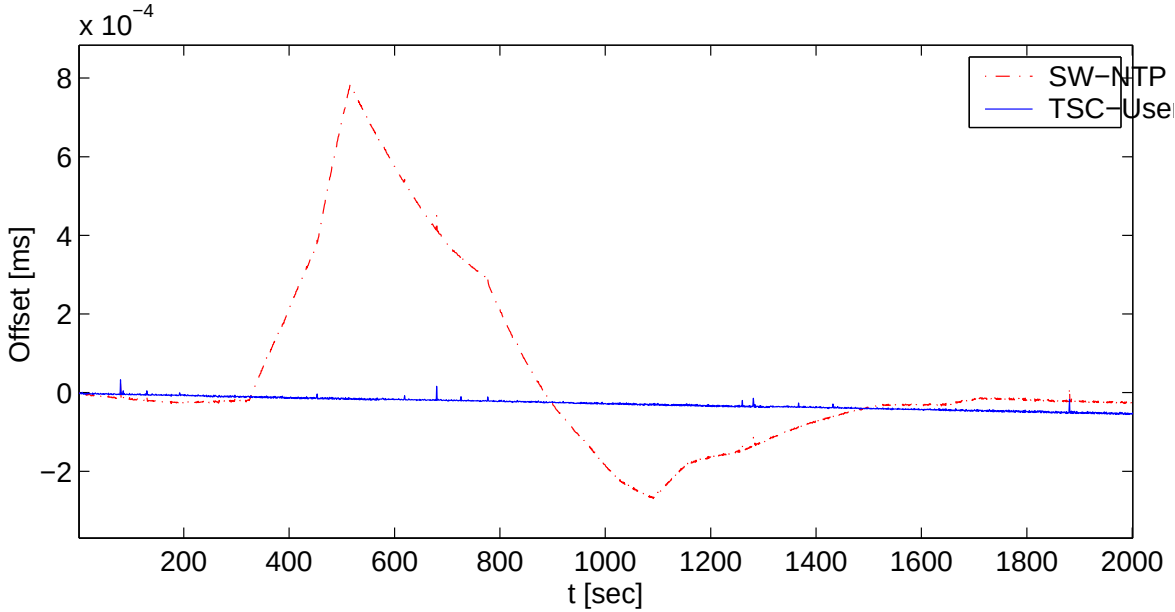
Problem: how to distinguish non-obvious timing errors from delay variations?

Offset and Rate Under SW-NTP

Under ‘good conditions’, SW-NTP controls offset over large time scales.



But looking more closely, we see the induced rate irregularities.



So Why Can We Do Better?

Advances in hardware have changed the status quo,

From

Local clock is very unreliable, must ask (and trust) the expert

To

Local clock is excellent, must simply calibrate it (and mistrust the expert)

So Why Can We Do Better?

Advances in hardware have changed the status quo,

From

Local clock is very unreliable, must ask (and trust) the expert

To

Local clock is excellent, must simply calibrate it (and mistrust the expert)

Still need reference source

- use (nearby stratum–1) NTP servers, and NTP protocol

So Why Can We Do Better?

Advances in hardware have changed the status quo,

From

Local clock is very unreliable, must ask (and trust) the expert

To

Local clock is excellent, must simply calibrate it (and mistrust the expert)

Still need reference source

- use (nearby stratum–1) NTP servers, and NTP protocol

But new perspective: local centric & rate centric

- inspires new filtering philosophy → greater accuracy and robustness
- delivers rate and offset, through separate difference and absolute clocks

The CPU Oscillator as a Clock

TSC (or CCC) register counts CPU cycles, 1 cycle per p [sec].

Two simple ideas:

Difference Clock: $\Delta(t) = \Delta(\text{TSC register value}) \times p$

Absolute Clock: $t = \theta_0 + (\text{TSC register value}) \times p$

The CPU Oscillator as a Clock

TSC (or CCC) register counts CPU cycles, 1 cycle per p [sec].

Two simple ideas:

Difference Clock: $\Delta(t) = \Delta(\text{TSC register value}) \times p$

Absolute Clock: $t = \theta_0 + (\text{TSC register value}) \times p$

Features/Advantages

- CPU oscillator and TSC Register stable features of PC architecture
- Hardware updating
- Ultra fast raw timestamping (read register): (< 50 ns)
- Nanosecond resolution

The CPU Oscillator as a Clock

TSC (or CCC) register counts CPU cycles, 1 cycle per p [sec].

Two simple ideas:

Difference Clock: $\Delta(t) = \Delta(\text{TSC register value}) \times p$

Absolute Clock: $t = \theta_0 + (\text{TSC register value}) \times p$

Features/Advantages

- CPU oscillator and TSC Register stable features of PC architecture
- Hardware updating
- Ultra fast raw timestamping (read register): (< 50 ns)
- Nanosecond resolution

The Catch:

– must estimate the cycle period p and offset θ_0 without special hardware.

The Simple Skew Model

- The Clock $C(t)$:
- Offset $\theta(t) = C(t) - t$: difference from true time t at time t.
- Skew γ: average difference in rate from true rate.

A Simple Model for Offset Error:

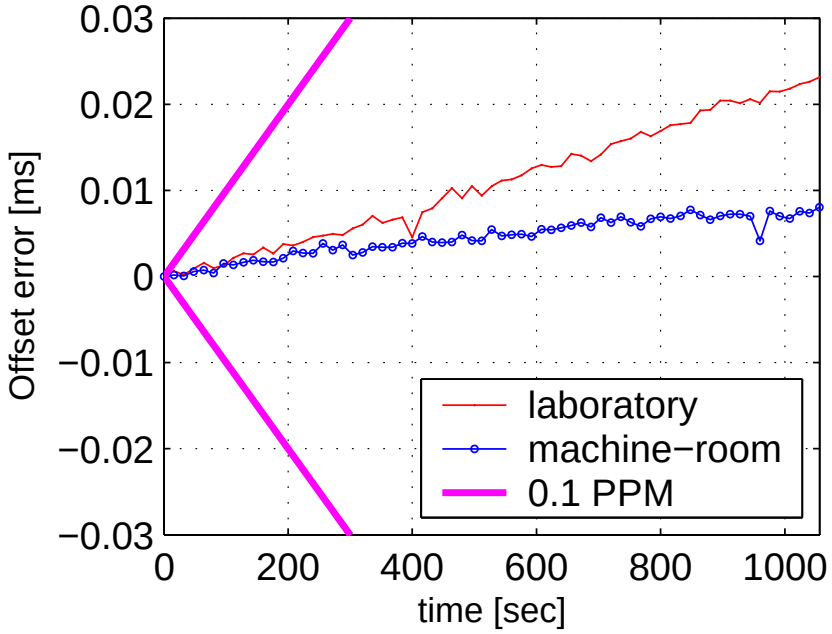
$$\theta(t) = \theta_0 + \gamma t + \omega(t)$$

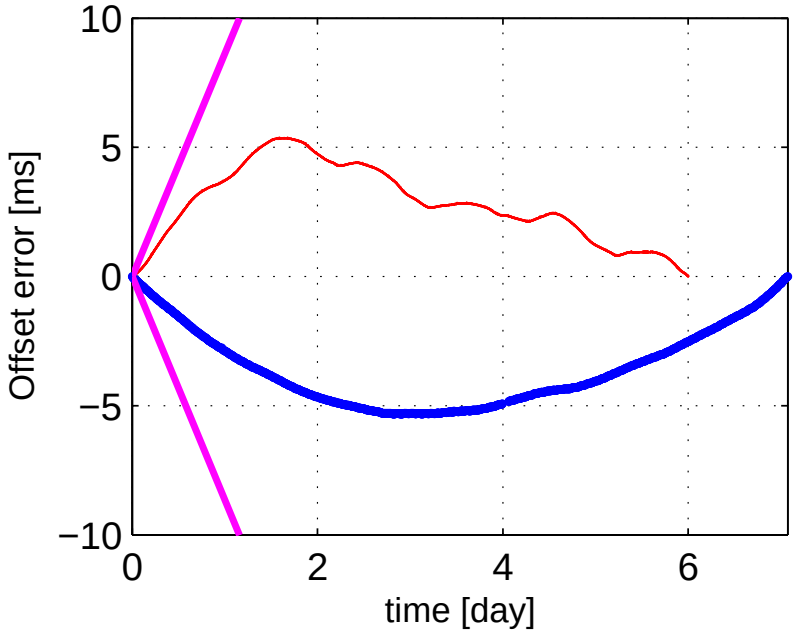
Simple Skew Model (SKM) plus fluctuations.

Limitations of the Simple Skew Model

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

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





Measured using reference: GPS synchronised DAG3.2E card (100ns)

Oscillator Stability

- The Clock $C(t)$:
- Offset $\theta(t) = C(t) - t$: difference from true time t at time t.
- Skew γ: average difference in rate from true rate.

A Simple Model for Offset Error:

$$\theta(t) = \theta_0 + \gamma t + \omega(t)$$

Simple Skew Model (SKM) plus fluctuations.

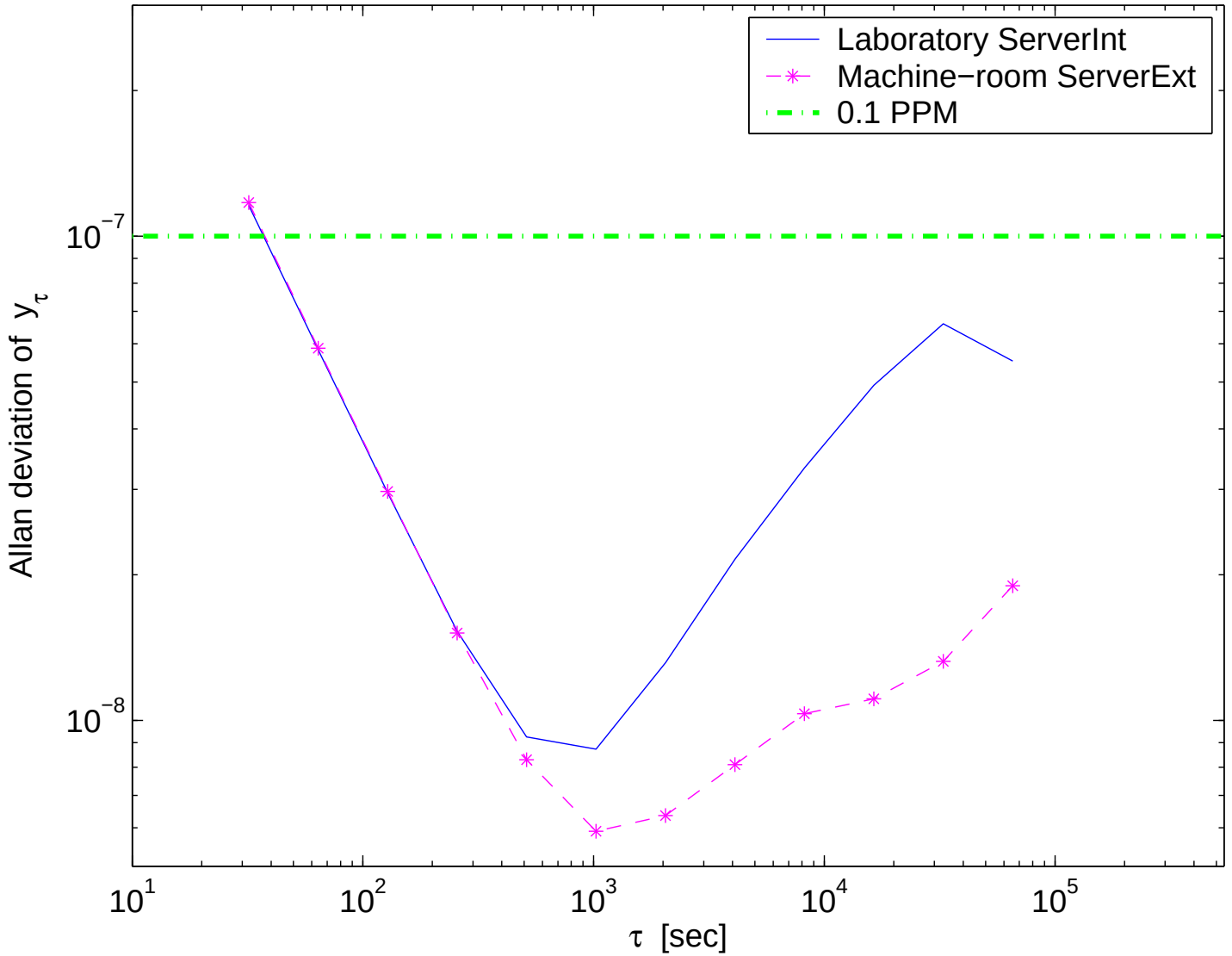
Oscillator Stability:

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

Studied through the Allan Variance, a scale-dependent 'variance' estimation of the family $\{y_\tau(t)\}$ of relative offset errors.

Allan deviation of $y_\tau(t)$ is size of rate fluctuations at scale τ

The SKM Timescale τ^*, and the 10^{-7} Bound



$\tau^* = 1000$ [sec]

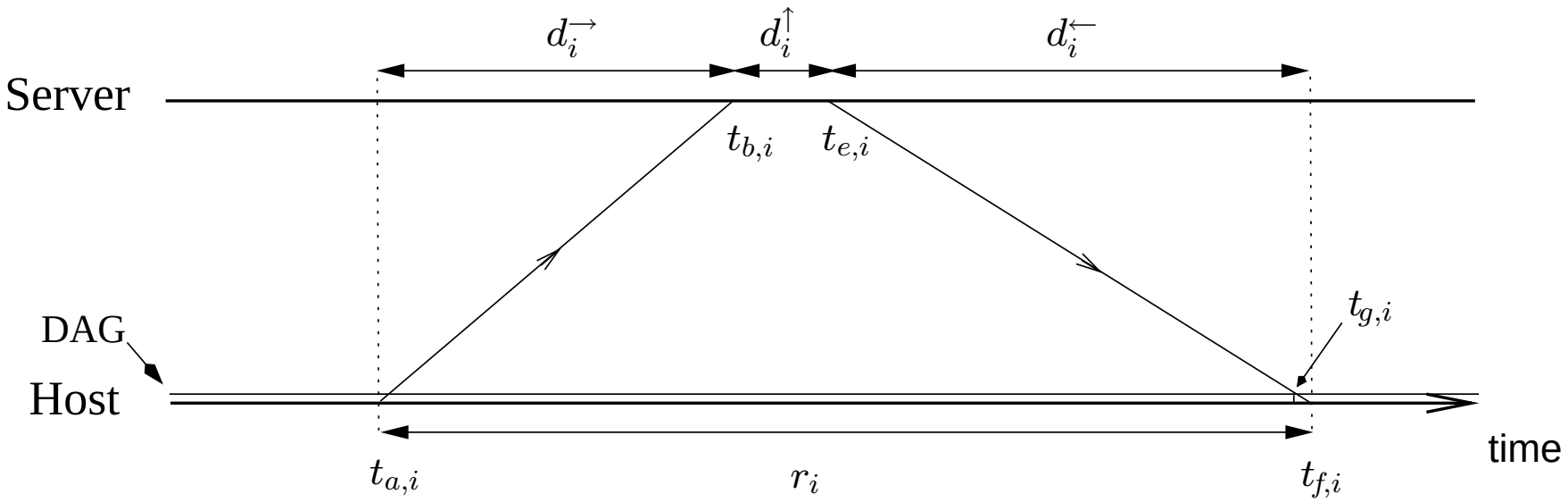
Key Features of Hardware Performance

Study of Oscillator Stability (variability) over different timescales shows:

- Simple Skew Model: holds strictly for $\tau^* = 1000$ [sec]
- Average rate error never more than 0.1 PPM no matter the scale

Very smooth constant rate: must take advantage!

A Supply of NTP Packets



Timestamps $\{T_{a,i}, T_{b,i}, T_{e,i}, T_{f,i}\}$ are the raw data from the i-th NTP packet.

- $\{T_{a,i}, T_{f,i}\}$: host timestamps in TSC counter units
- $\{T_{b,i}, T_{e,i}\}$: server timestamps in seconds

Network and Server Delay

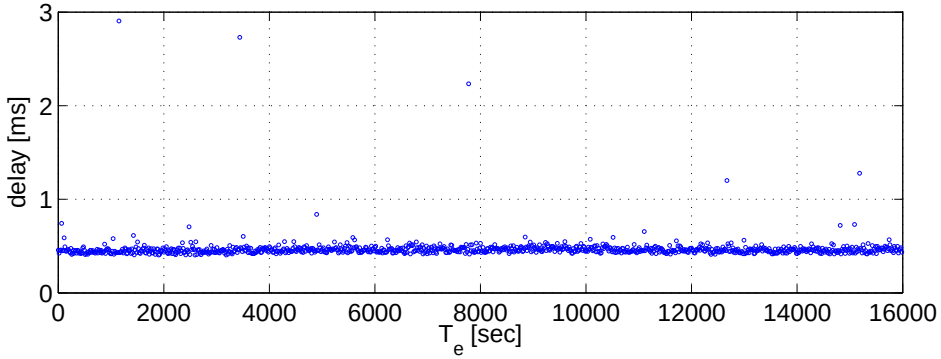
Forward network delay : $d_i^{\rightarrow} \equiv t_{b,i} - t_{a,i} = d^{\rightarrow} + q_i^{\rightarrow}$

Server Delay : $d_i^{\uparrow} \equiv t_{e,i} - t_{b,i} = d^{\uparrow} + q_i^{\uparrow}$

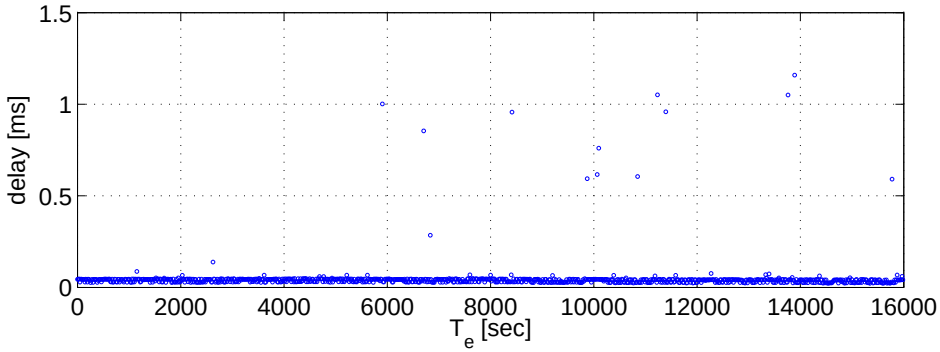
Backward network delay : $d_i^{\leftarrow} \equiv t_{f,i} - t_{e,i} = d^{\leftarrow} + q_i^{\leftarrow}$

Round Trip Time : $r_i \equiv t_{f,i} - t_{a,i} = d_i^{\rightarrow} + d_i^{\uparrow} + d_i^{\leftarrow}. = r + q_i^{\rightarrow} + q_i^{\uparrow} + q_i^{\leftarrow}$

Backward Network Delay $d_i^{\leftarrow}$



Server Delay $d_i^{\uparrow}$



Rate Synchronisation in the SKM World

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

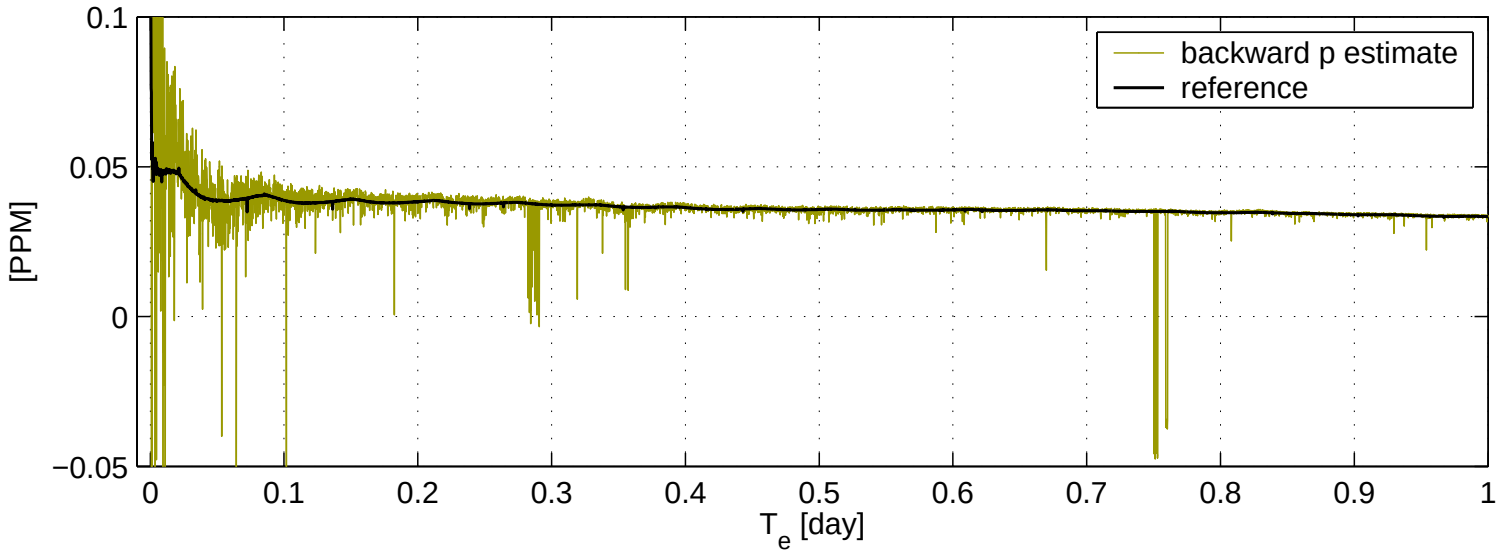
On forward path:

$$p = \frac{t_{b,i} - t_{b,j} - (q_i^{\rightarrow} - q_j^{\rightarrow})}{\mathrm{TSC}(t_{a,i}) - \mathrm{TSC}(t_{a,j})}, \qquad i > j.$$

Inspires the naive estimate

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

Let $j = 1$, and i the current packet.



Network delay and timestamping noise $\sim 1/\Delta(t)$, but errors not bounded.

RTT Approach to Filtering

Need to assess quality of packets (timestamps, queueing/system delays)

Basic Procedure

- Model for RTT: $r_i = r + (\text{positive random noise})$ allows minimum filtering
- Define point error: $r_i - r$
- Estimate robustly as: $E_i = r_i - \min_{i=1}^{\lfloor t \rfloor} r_i$
- Compare E_i to threshold E^* or use as basis of weight

RTT Approach to Filtering

Need to assess quality of packets (timestamps, queueing/system delays)

Basic Procedure

- Model for RTT: $r_i = r + (\text{positive random noise})$ allows minimum filtering
- Define point error: $r_i - r$
- Estimate robustly as: $E_i = r_i - \min_{i=1}^{\lfloor t \rfloor} r_i$
- Compare E_i to threshold E^* or use as basis of weight

But what about prior calibration of $C(t) = \text{TSC}(t) * p + \theta_0$?

RTT Approach to Filtering

Need to assess quality of packets (timestamps, queueing/system delays)

Basic Procedure

- Model for RTT: $r_i = r + (\text{positive random noise})$ allows minimum filtering
- Define point error: $r_i - r$
- Estimate robustly as: $E_i = r_i - \min_{i=1}^{\lfloor t \rfloor} r_i$
- Compare E_i to threshold E^* or use as basis of weight

But what about prior calibration of $C(t) = \text{TSC}(t) * p + \theta_0$?

Choose RTT based filtering, not one-way

- Since same clock, θ_0 irrelevant
- No prior p needed! point error can be in TSC units,
- Rough global $\widehat{p}$ sufficient to set threshold in [sec]
- $\longrightarrow$ decoupling of pkt quality assessment from estimation – no chicken and egg dynamics
- Lack of drift **enables** minimum filtering!

Disadvantage

- Very demanding $\longrightarrow$ few quality packets

RTT Approach to Filtering

Need to assess quality of packets (timestamps, queueing/system delays)

Basic Procedure

- Model for RTT: $r_i = r + (\text{positive random noise})$ allows minimum filtering
- Define point error: $r_i - r$
- Estimate robustly as: $E_i = r_i - \min_{i=1}^{\lfloor t \rfloor} r_i$
- Compare E_i to threshold E^* or use as basis of weight

But what about prior calibration of $C(t) = \text{TSC}(t) * p + \theta_0$?

Choose RTT based filtering, not one-way

- Since same clock, θ_0 irrelevant
- No prior p needed! point error can be in TSC units,
- Rough global $\widehat{p}$ sufficient to set threshold in [sec]
- $\longrightarrow$ decoupling of pkt quality assessment from estimation – no chicken and egg dynamics
- Lack of drift **enables** minimum filtering!

Disadvantage

- Very demanding $\longrightarrow$ few quality packets

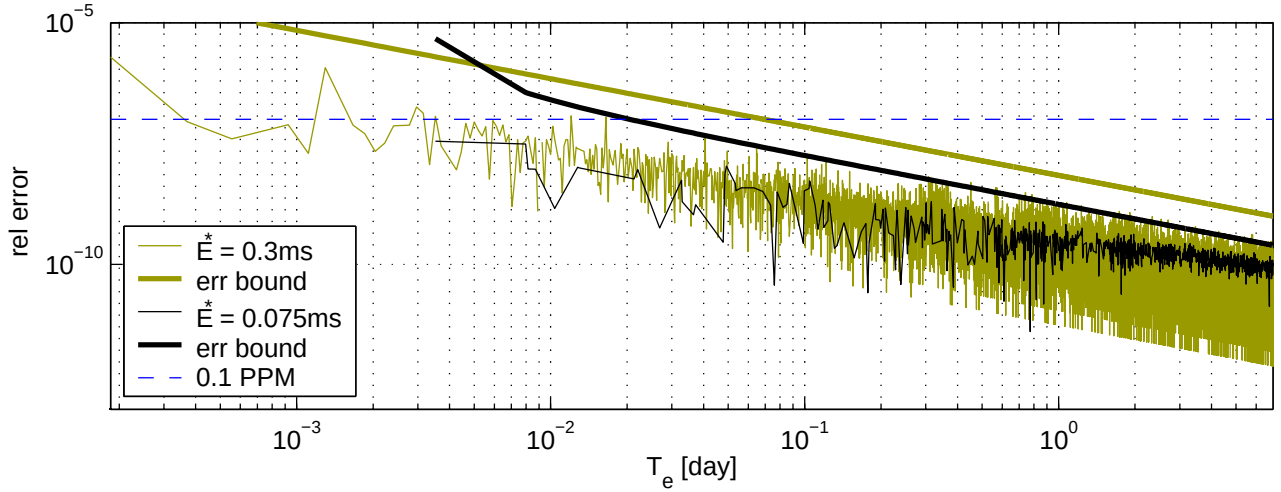
Still using SKM! but $\gamma < 10^{-7}$: *increase in offset over RTT negligible* [0.1μs over 1sec]

Rate Synchronisation Algorithm

Use selected naive estimates, be patient

Algorithm for $\widehat{p}(t)$

- Let j and i be 1st and 2nd pkts: $E_k < E^*$ ($E^* = 5\delta$, $\delta = 15\mu$s (interrupt latency))
- Naive estimate $\widehat{p}_{i,j}^{\rightarrow}$ holds until next quality i
- Error of $\widehat{p}$ bounded by $2E^*/((T_{f,i} - T_{f,j})\bar{p})$



Key Features

- Error quickly $< 10^{-7}$, In 10mins, better than GPS for most active probing!
- Error reduction (in timestamping, latency, queueing) guaranteed by increasing $\Delta(t)$
- Inherently robust to packet loss, congestion (quality scarcity ok), loss of server..
- Simple algorithm, no need for local estimates

Offset Synchronisation in the SKM World

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

- through filtering
- exploiting $\gamma < 10^{-7}$: slow evolution means single θ_i per packet

Offset Synchronisation in the SKM World

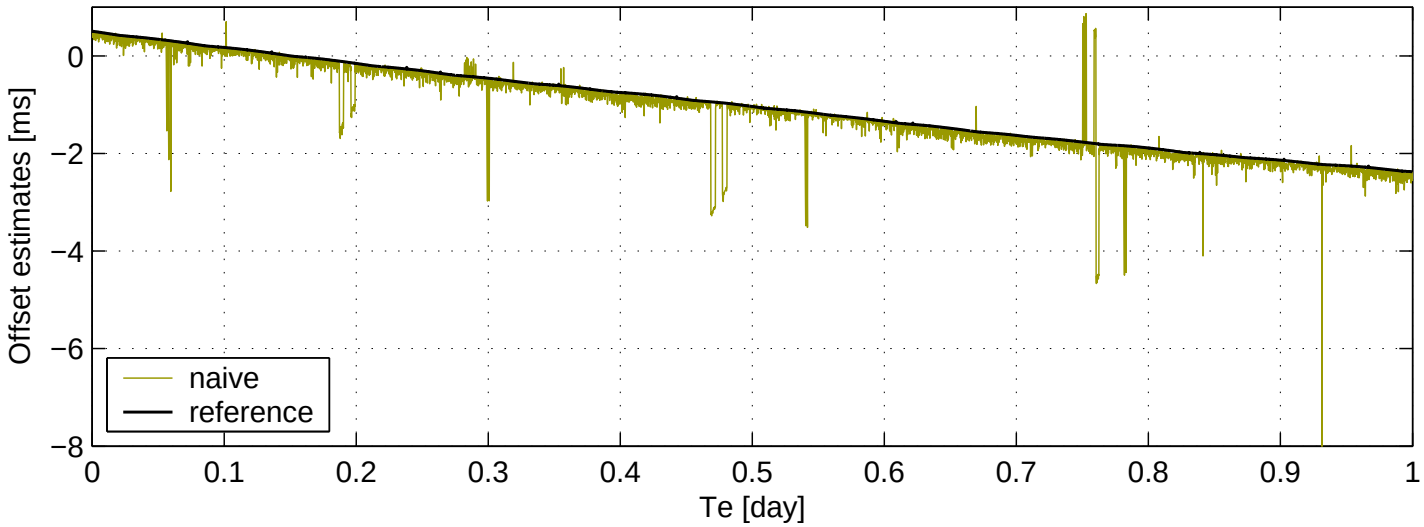
Wish to exploit SKM over small scales to measure $\theta(t)$

- through filtering
- exploiting $\gamma < 10^{-7}$: slow evolution means single θ_i per packet

$$\theta_i = \frac{1}{2}(C(t_{a,i}) + C(t_{f,i})) - \frac{1}{2}(t_{b,i} + t_{e,i}) + \frac{1}{2}\Delta + \frac{1}{2}(q_i^{\rightarrow} - q_i^{\leftarrow}),$$

Inspires the naive estimate (assumes $\Delta \equiv d^{\rightarrow} - d^{\leftarrow} = 0$)

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



Offset Synchronisation Algorithm

Must track, so use all naive estimates, but be very fussy!

Algorithm for $\widehat{\theta}(t)$

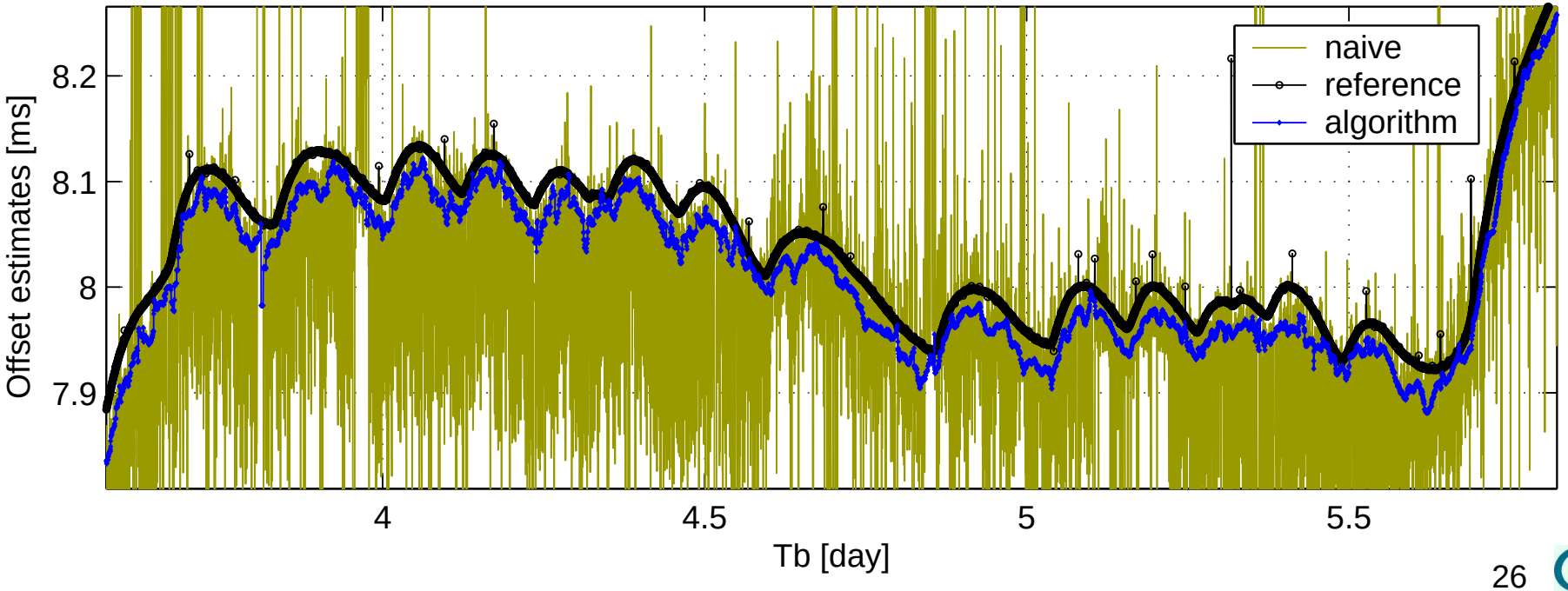
- Total (aged) per-packet error: $E_i^T = E_i + \epsilon(C_d(t) - C_d(T_{f,i}))$, $(\epsilon = 10^{-7})$
- For i in SKM window τ' wide before t, form weight: $w_i = \exp(-(E_i^T/E)^2)$, $(E = 4\delta > 0)$
- Form weighted estimate: $\widehat{\theta}(t) = \sum_i w_i \widehat{\theta}_i / \sum_i w_i$
  If quality very bad: $\min_i(E_i^T) > E^{**}$ $(E^{**} = 6E)$, do nothing: $\widehat{\theta}(t) = \widehat{\theta}(t_{f,i})$
- Sanity check: if impossible: $|\widehat{\theta}(t) - \widehat{\theta}(t_{f,i})| > E^s$, $(E^s = 1\text{ms})$, ignore: $\widehat{\theta}(t) = \widehat{\theta}(t_{f,i})$

Offset Synchronisation Algorithm

Must track, so use all naive estimates, but be very fussy

Algorithm for $\widehat{\theta}(t)$

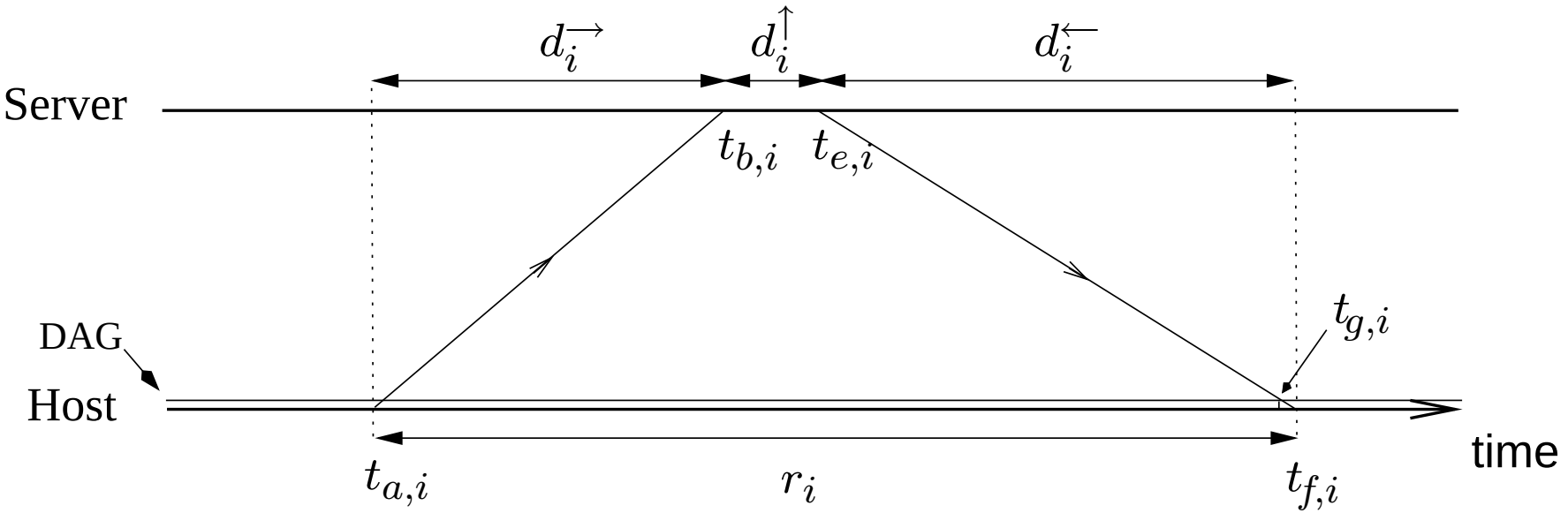
- Total (aged) per-packet error: $E_i^T = E_i + \epsilon(C_d(t) - C_d(T_{f,i}))$, $(\epsilon = 10^{-7})$
- For i in SKM window τ' wide before t, form weight: $w_i = \exp(-(E_i^T/E)^2)$, $(E = 4\delta > 0)$
- Form weighted estimate: $\widehat{\theta}(t) = \sum_i w_i \widehat{\theta}_i / \sum_i w_i$
  If quality very bad: $\min_i(E_i^T) > E^{**}$ $(E^{**} = 6E)$, do nothing: $\widehat{\theta}(t) = \widehat{\theta}(t_{f,i})$
- Sanity check: if impossible: $|\widehat{\theta}(t) - \widehat{\theta}(t_{f,i})| > E^s$, $(E^s = 1\text{ms})$, ignore: $\widehat{\theta}(t) = \widehat{\theta}(t_{f,i})$



The Path Asymmetry Δ

Fundamental ambiguity: $\Delta \equiv d^{\rightarrow} - d^{\leftarrow}$ and 2θ indistinguishable up to a constant.

Δ unknown: forced to assume $\Delta = 0$



However, a causality bound applies: $\Delta \in (-r, r)$

Server Characteristics: RTT and Δ

Server	Reference	Distance	RTT	Hops	Δ
ServerLoc	GPS	3 m	0.38 ms	2	$\approx$ 0.05 ms
ServerInt	GPS	300 m	0.89 ms	5	$\approx$ 0.05 ms
ServerExt	Atomic	1000 km	14.2 ms	$\approx$10	$\approx$ 0.5 ms

For *ServerInt*: $\Delta \approx 50\mu$s

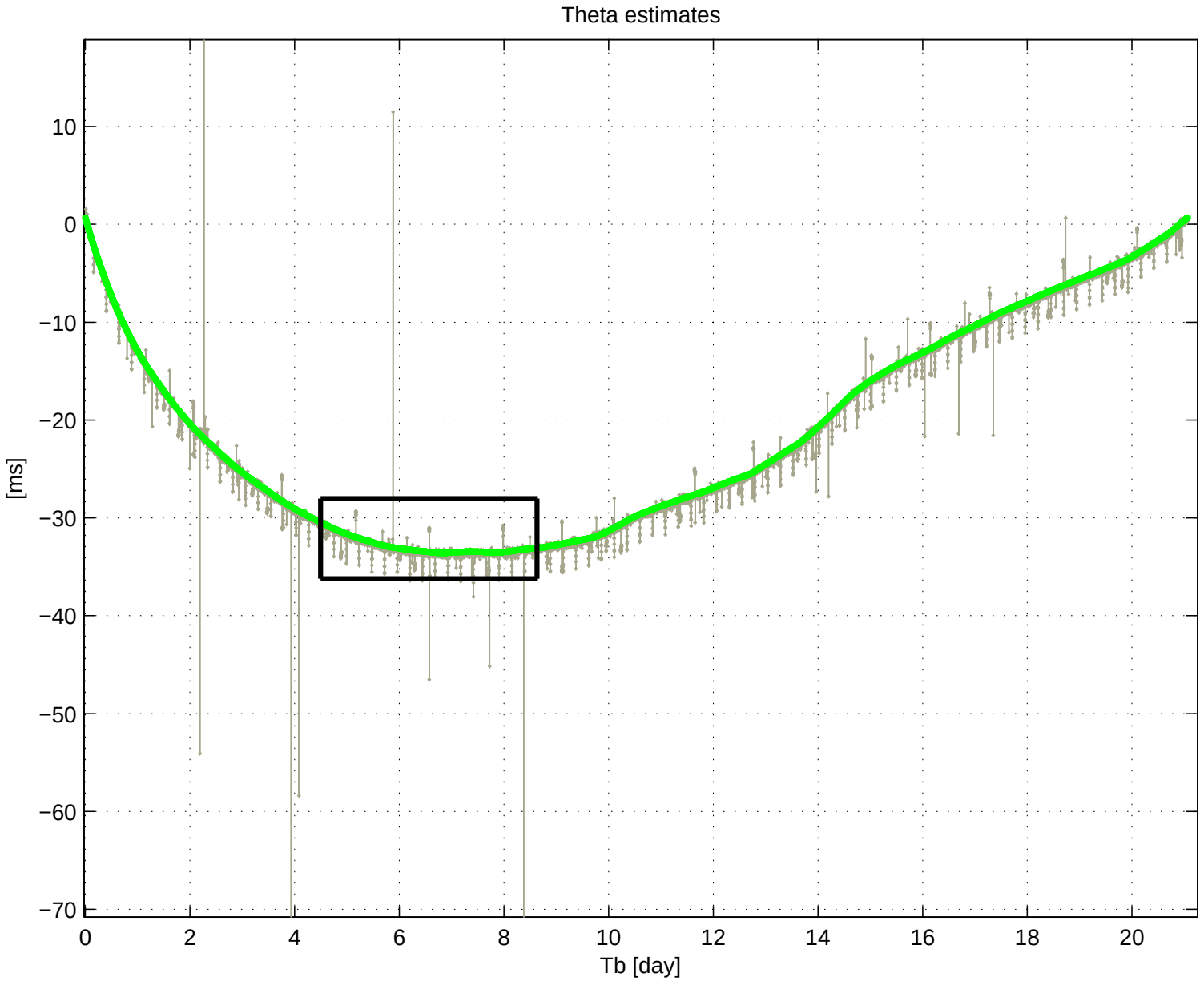
Theoretical best offset error is: $\Delta/2 \approx 25\mu$s

Measured median offset error only: 28μs

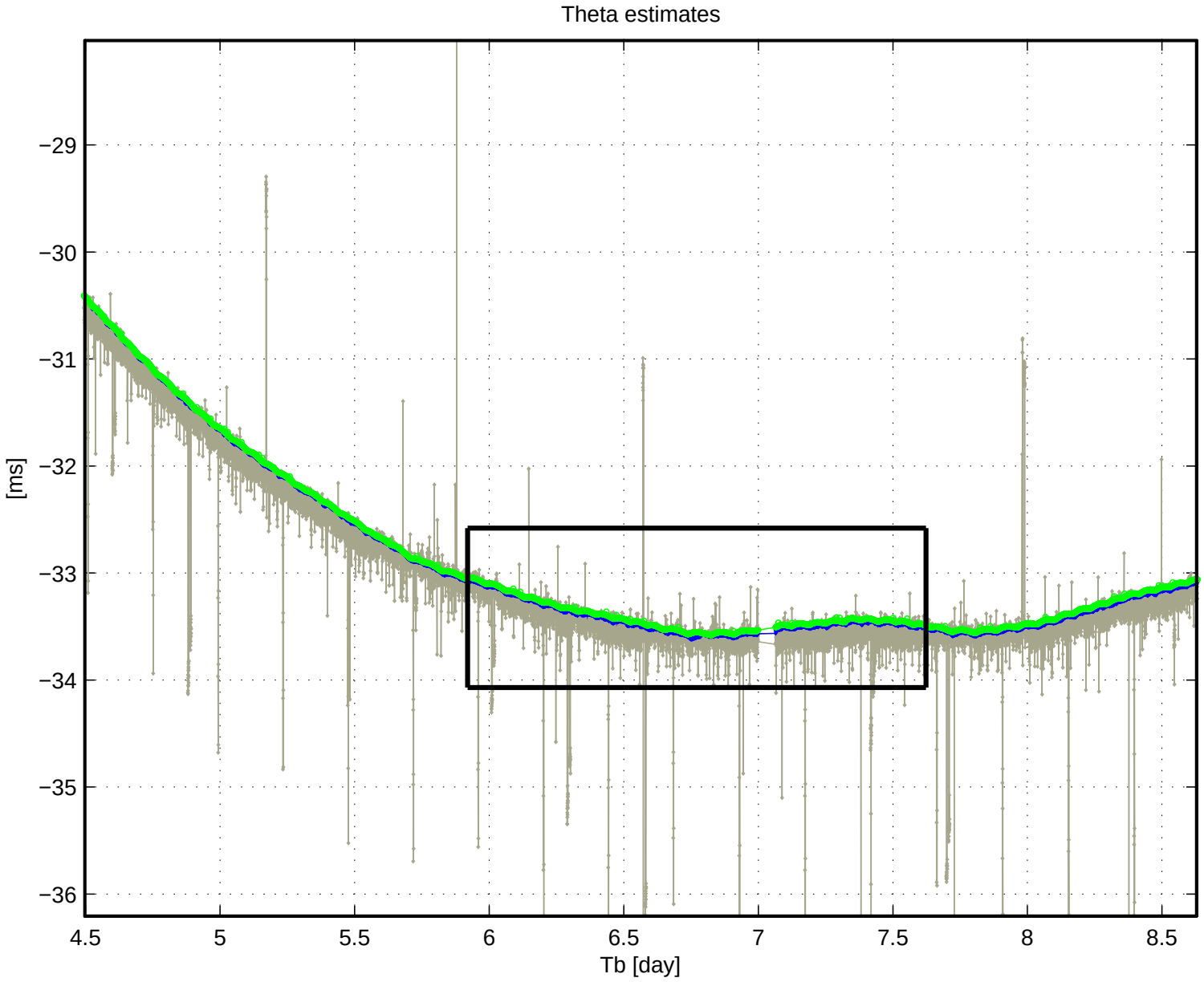
Choice of server is crucial:

- Close server has small RTT
- Close server more likely to have symmetric path: $\Delta \ll r$
- Aim: find a stratum–1 NTP server, with small $r \sim 1$ ms, and known, symmetric path.

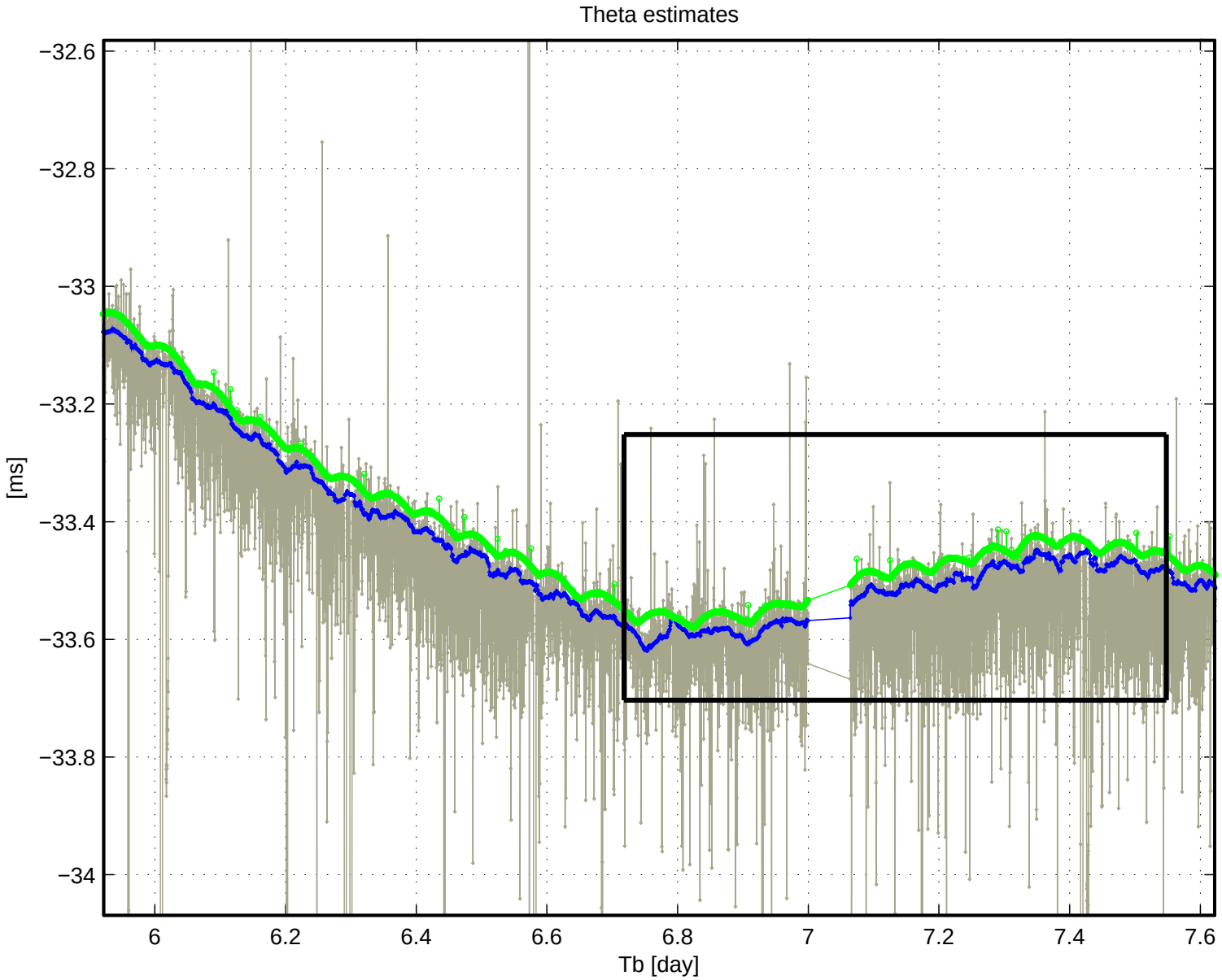
Zooming on Offset



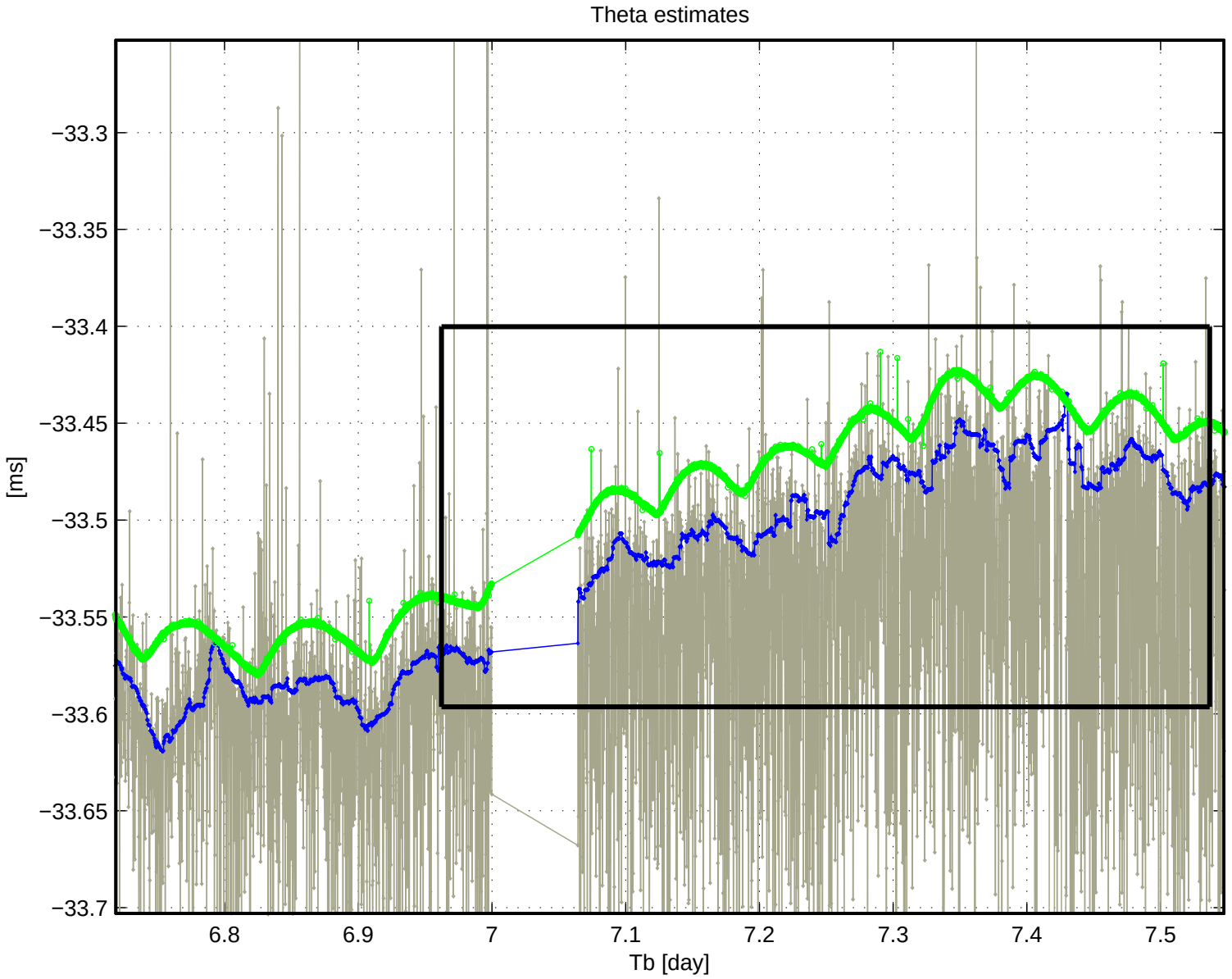
Zooming on Offset



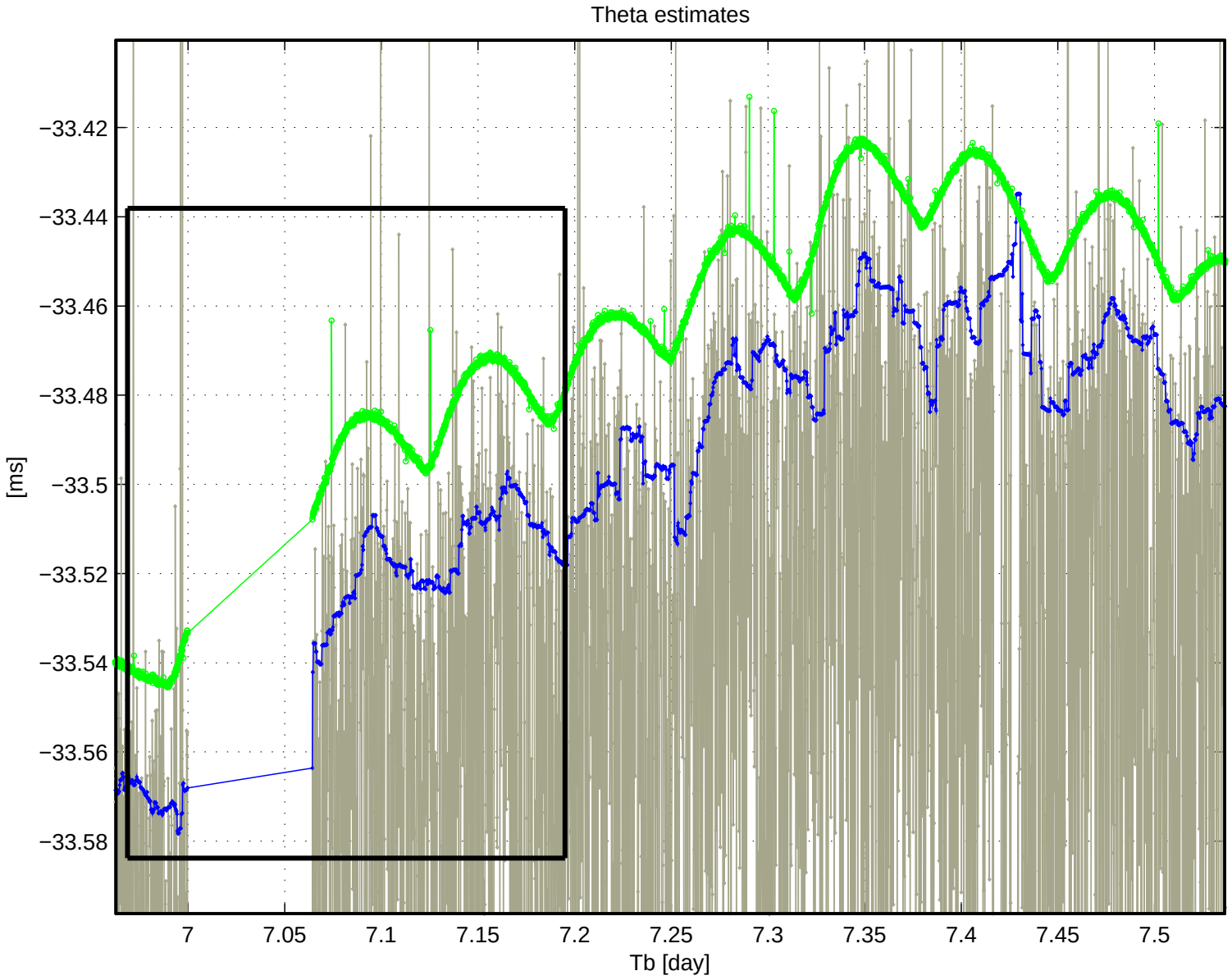
Zooming on Offset



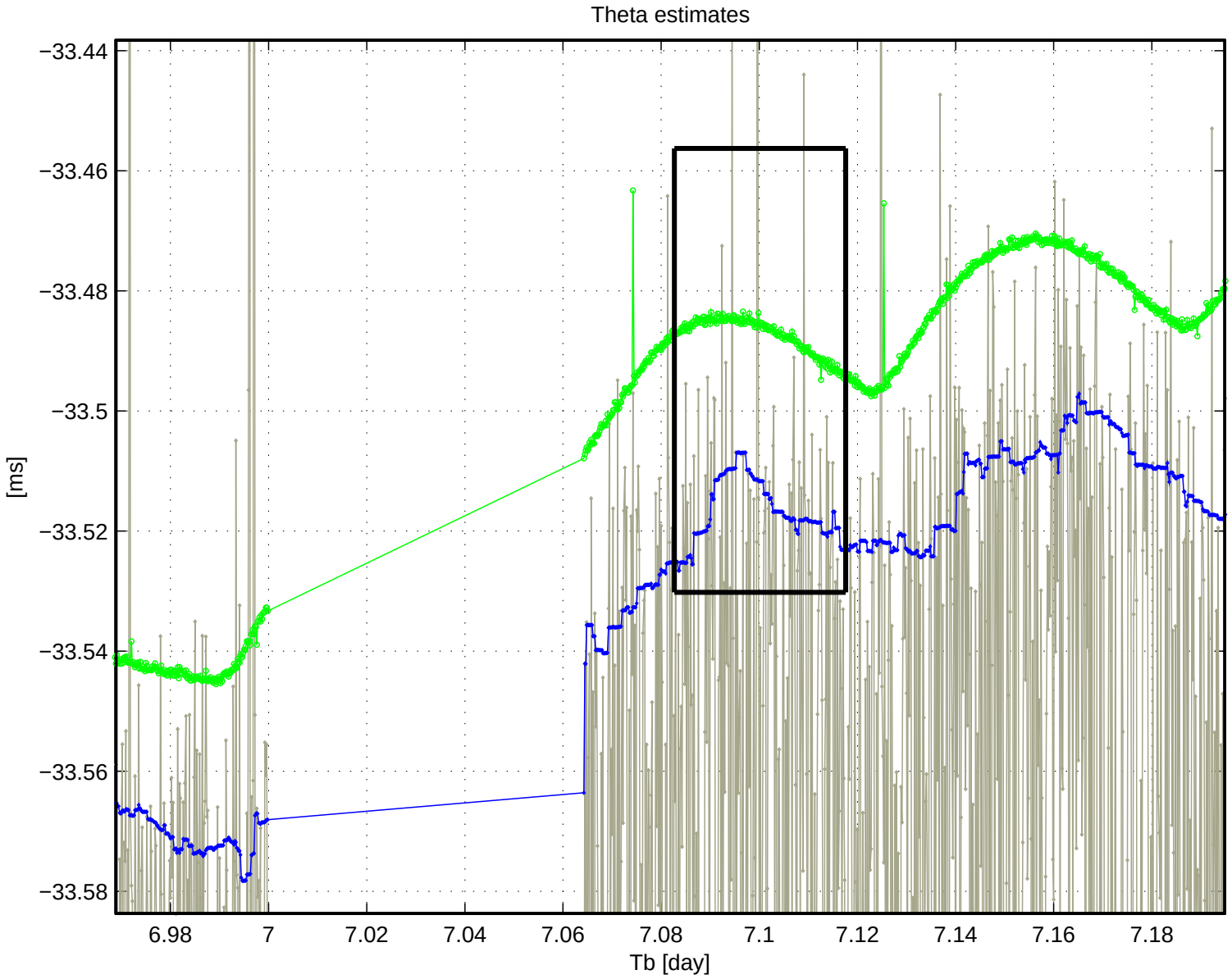
Zooming on Offset



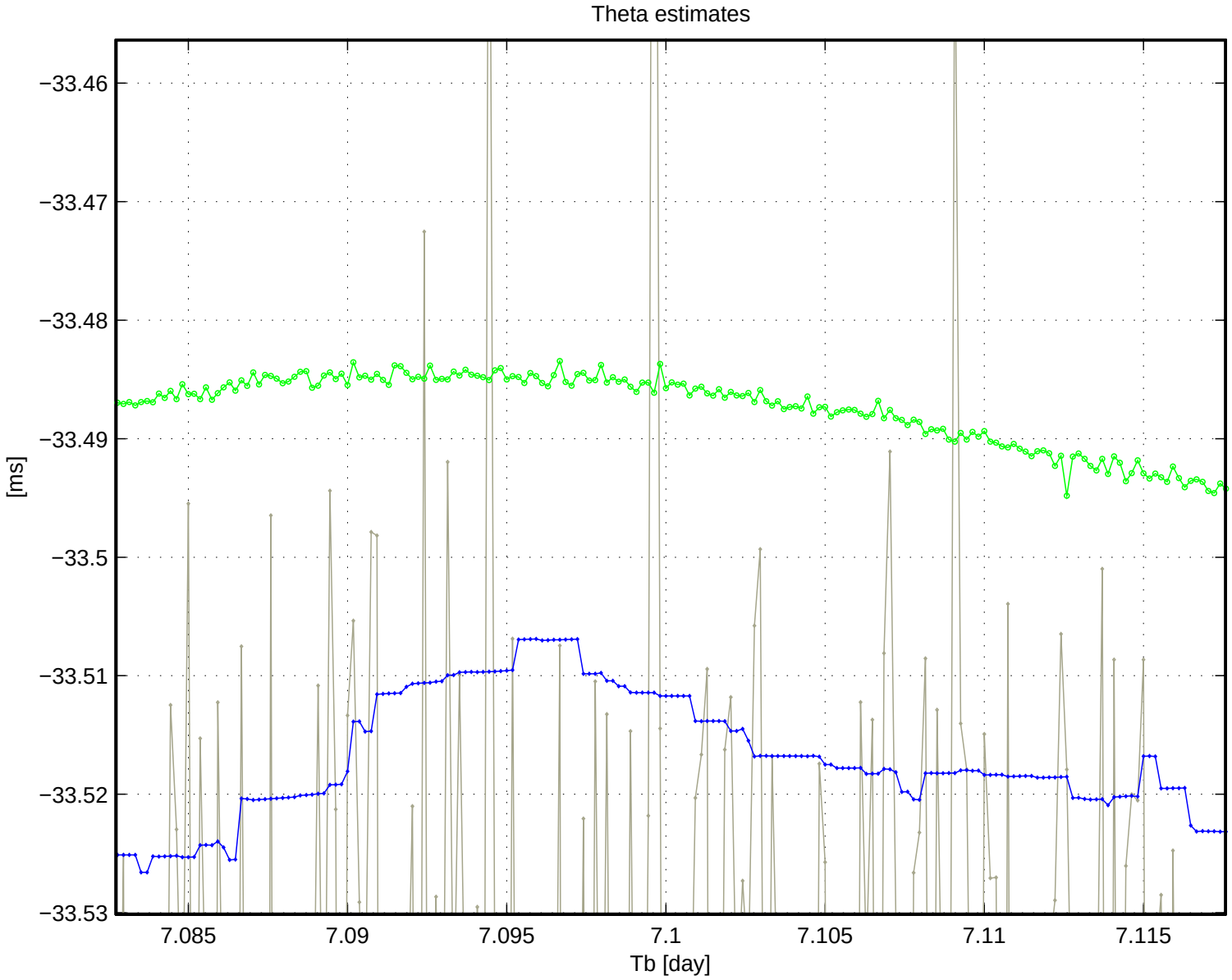
Zooming on Offset



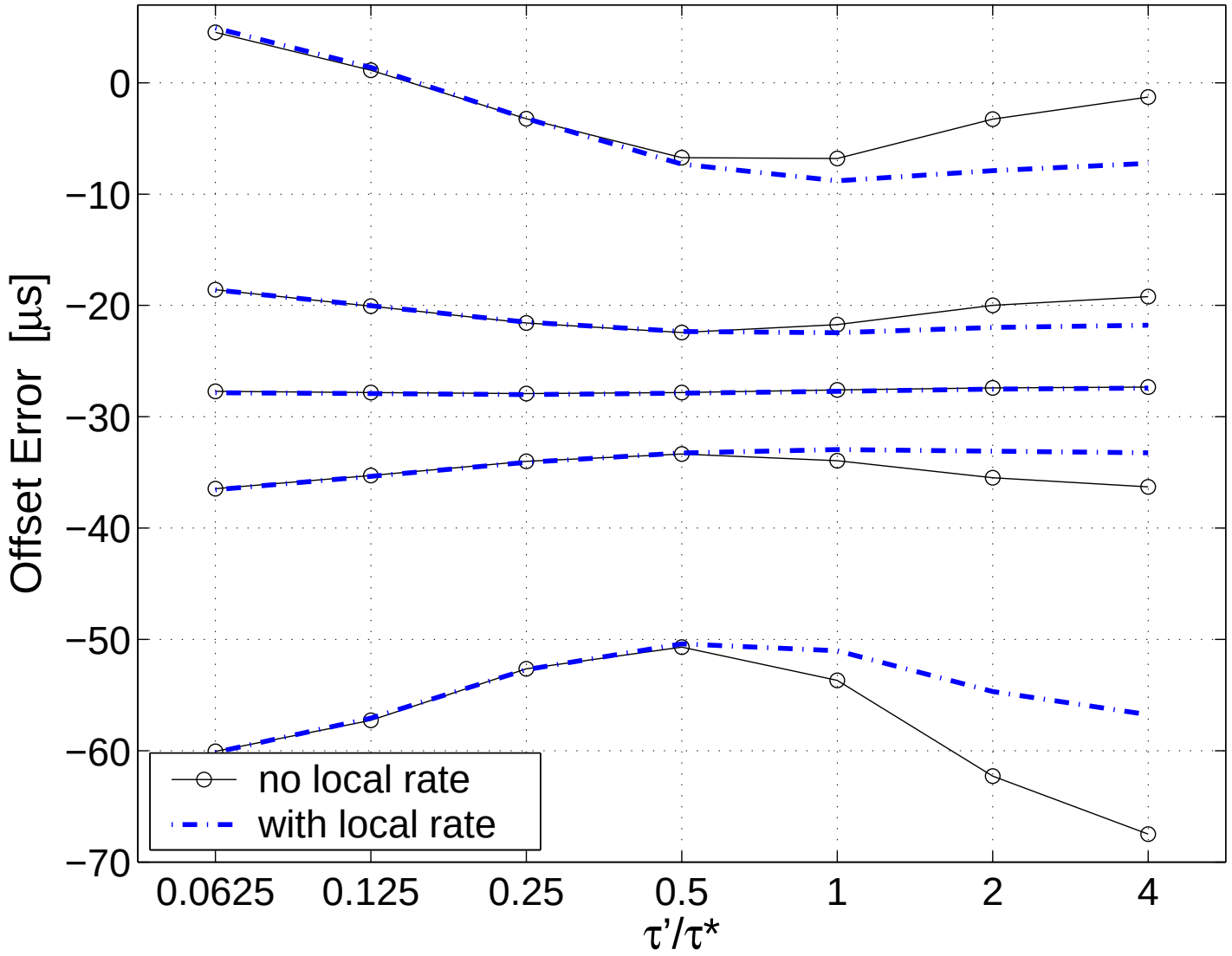
Zooming on Offset



Zooming on Offset

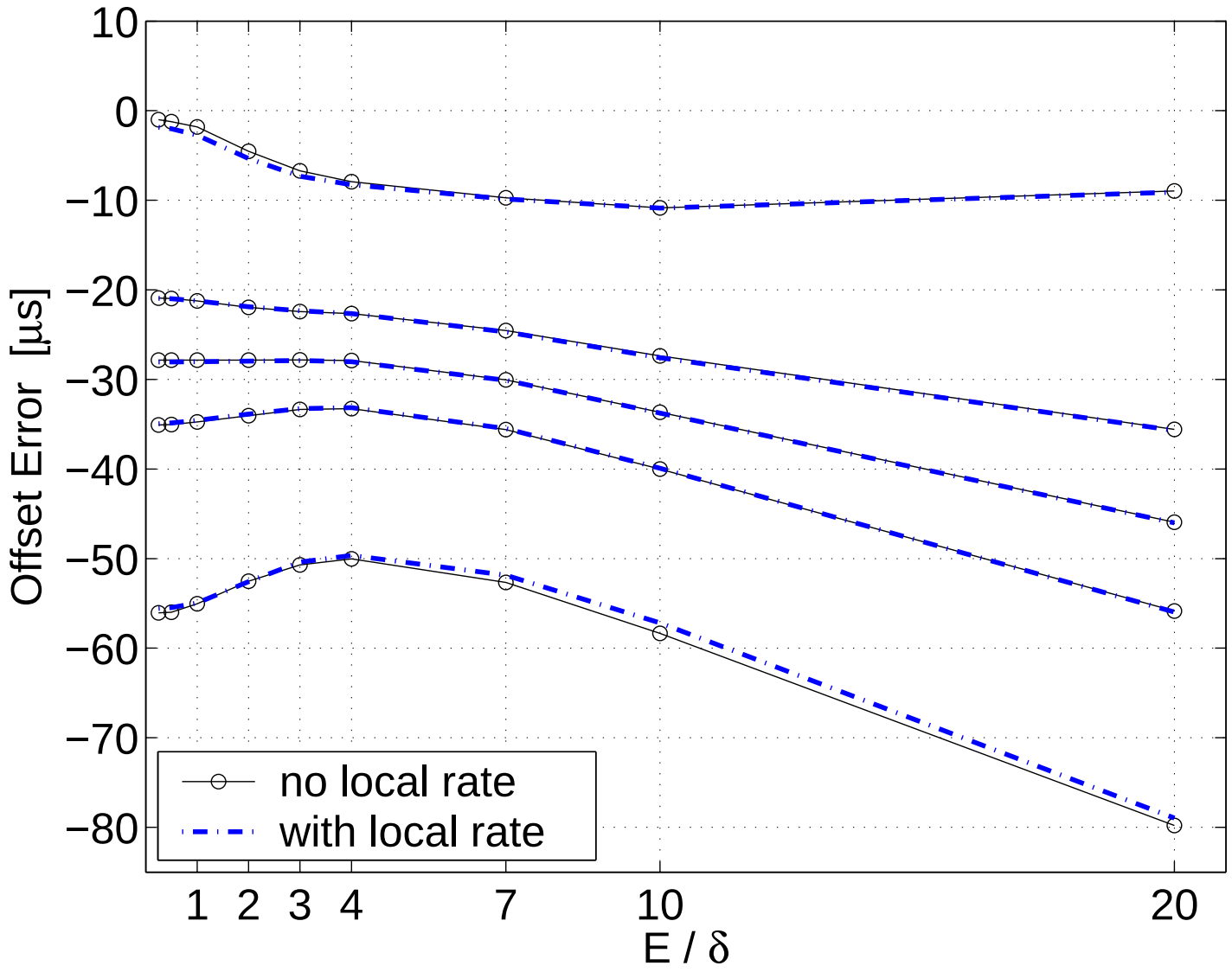


Performance as Function of Window Size τ'



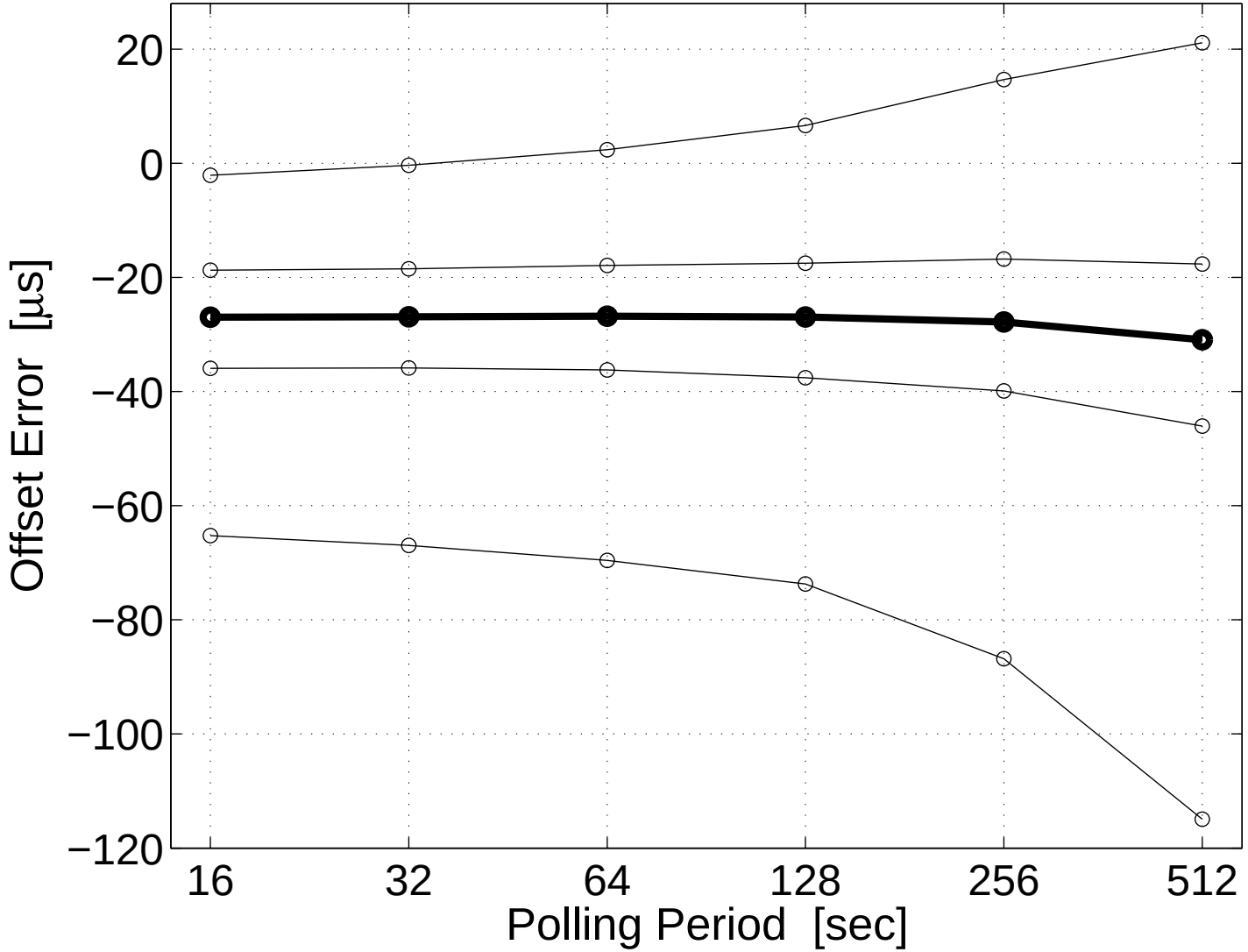
ServerInt, Machine-Room, $E = 4\delta$ (with and without local rate: $\bar{\tau} = 20\tau^*$)

Performance as Function of Quality Band E



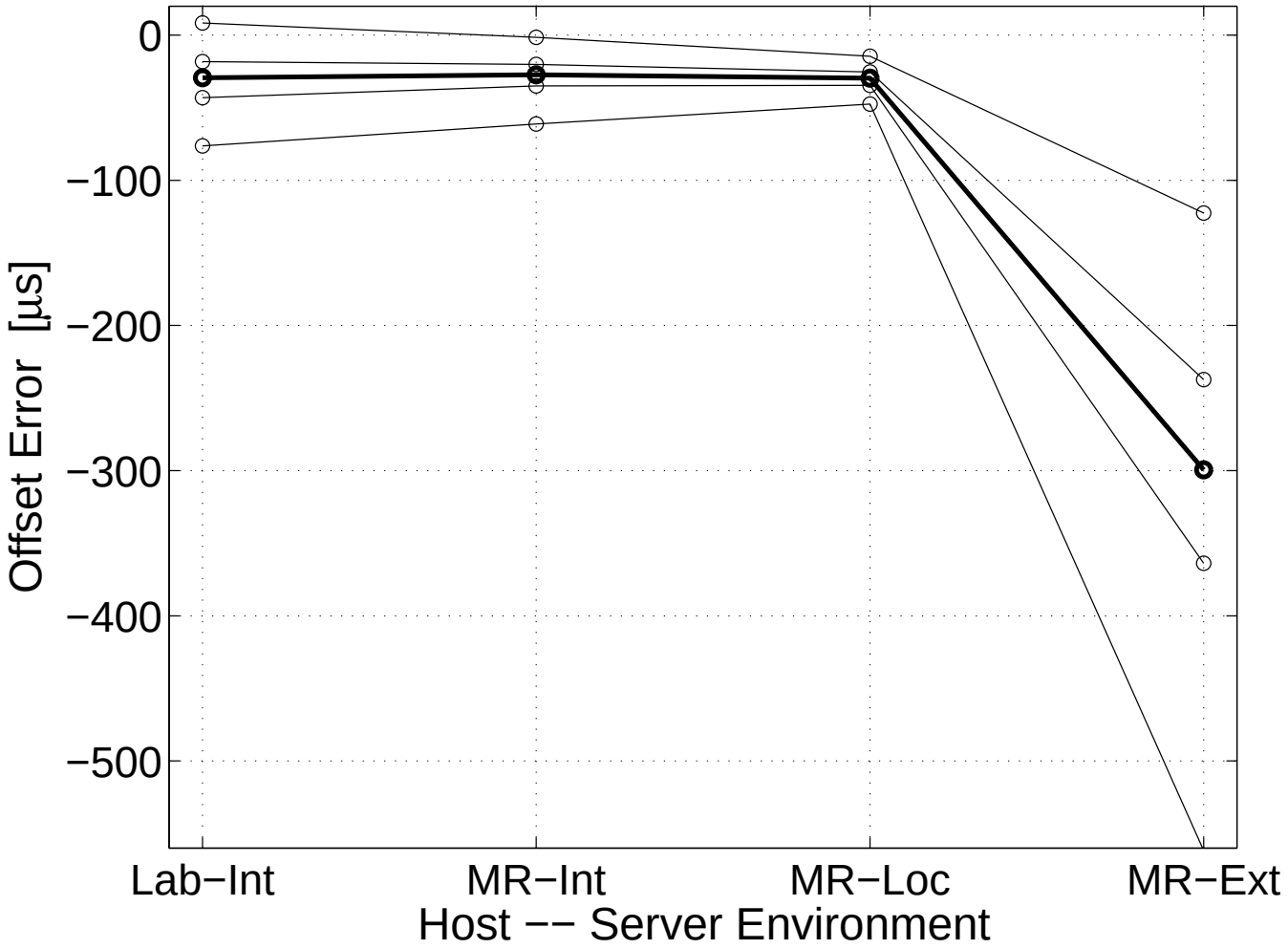
ServerInt, Machine-Room, $\tau' = \tau^*/2$ (with and without local rate: $\bar{\tau} = 20\tau^*$)

Performance as Function of Polling Period



ServerInt, Machine-Room, $\tau' = \tau^*$, $E = 4\delta$

Performance as Function of Host – Server Environment



$\tau' = \tau^*$, $E = 4\delta$, Polling Period $= 64$

A Robust Working System

Additional Issues for Full System

- Windowing: sliding window of width T ($= 1$ week). Update $\widehat{r}(t)$, $\widehat{p}(t)$ after $T/2$
- Re-evaluation of packet qualities: whenever $\widehat{r}(t)$, $\widehat{p}(t)$ changed
- Clock origin consistency: after $\widehat{p}(t)$ changes, adjust constant C so old/new agree
- Lost packets: define all windows $(\tau^*, \tau', \bar{\tau}, T, T_s)$ by # packets, not time!
- Unforeseen events: generic robustness needed, but $\rightarrow$ level shifts

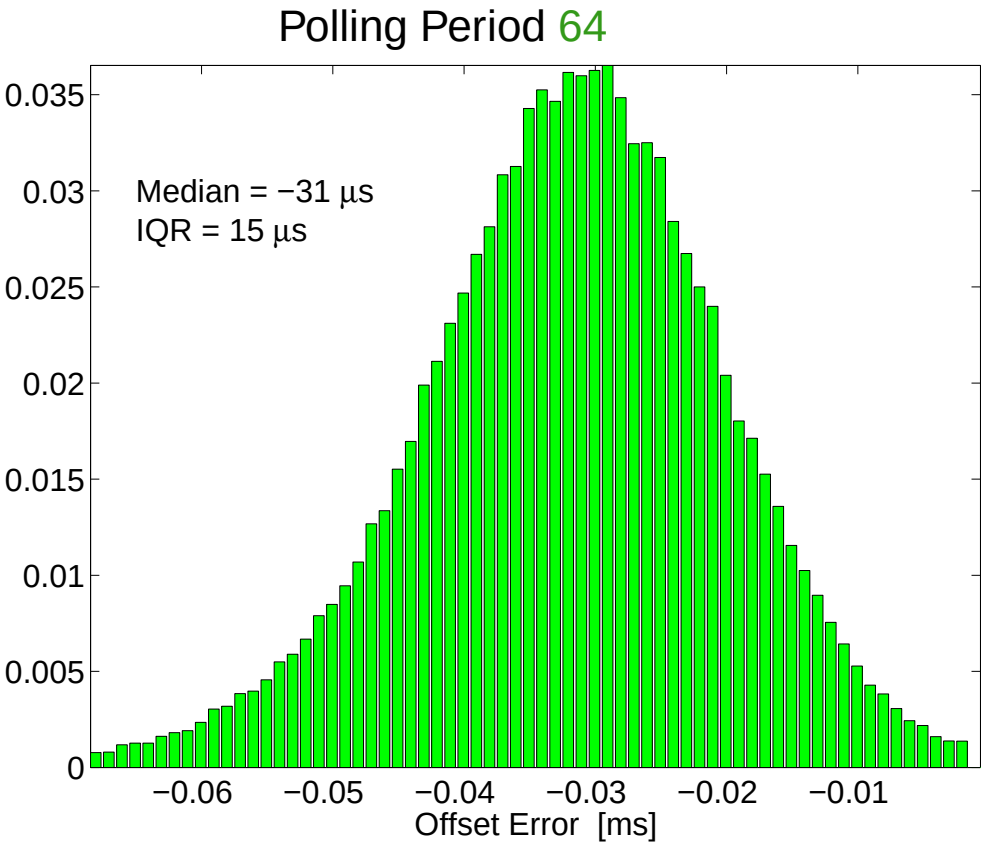
Robustness to Level Shift in RTT

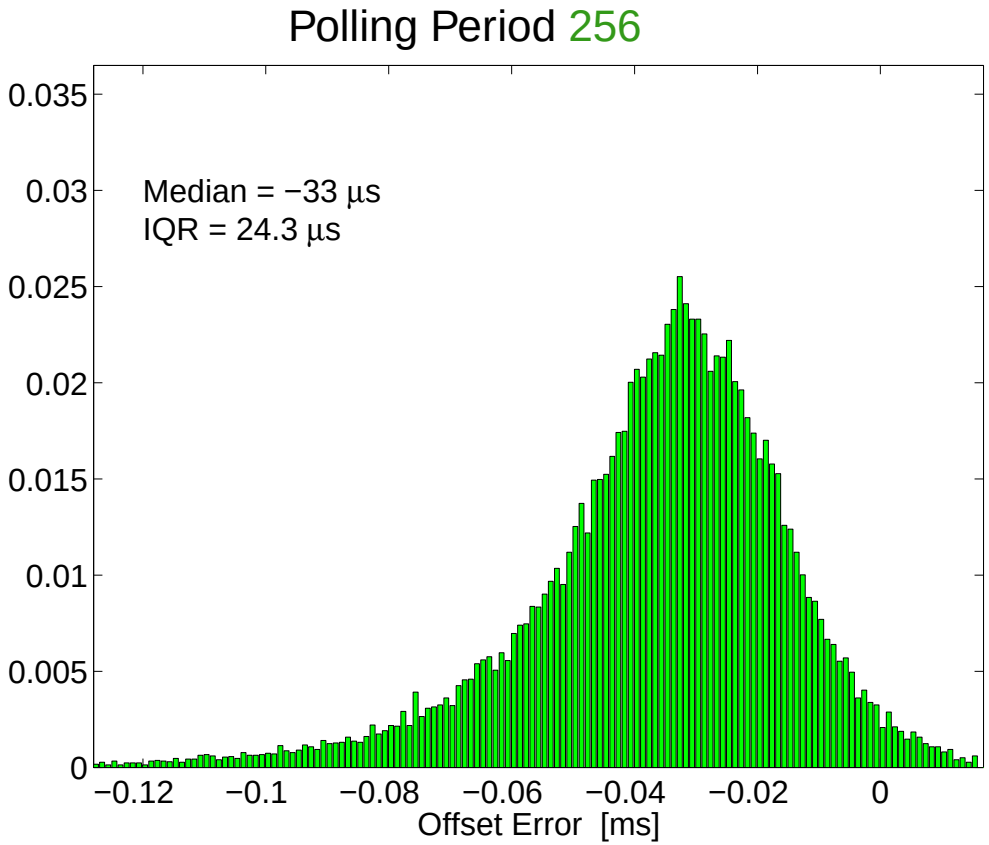
- Asymmetry of shift direction: down easy, up very hard
- Asymmetry of detection errors: misjudge quality ok, misjudge bad disastrous
- Asymmetry of offset and rate: old offsets ok without change, old rate not
- Algorithm :
  - Shift directions treated differently
  - Downward: Detection: automatic Reaction: none
  - Upward: Detection: if local min different (only if certain – large window)
    Reaction: update $\widehat{r}$ estimates and point errors
- Possible to handle level shifts reliably with almost no dedicated code

Robust to Diverse Factors

- Changes in environment/oscillator
- Changes in temperature
- Packet Loss (loss of connectivity..)
- Network congestion
- Timestamping errors (scheduling)
- Server errors
- Route/Server changes – Level shifts in RTT

Full C Version, 3 months of Data with *ServerInt* – Machine-Room

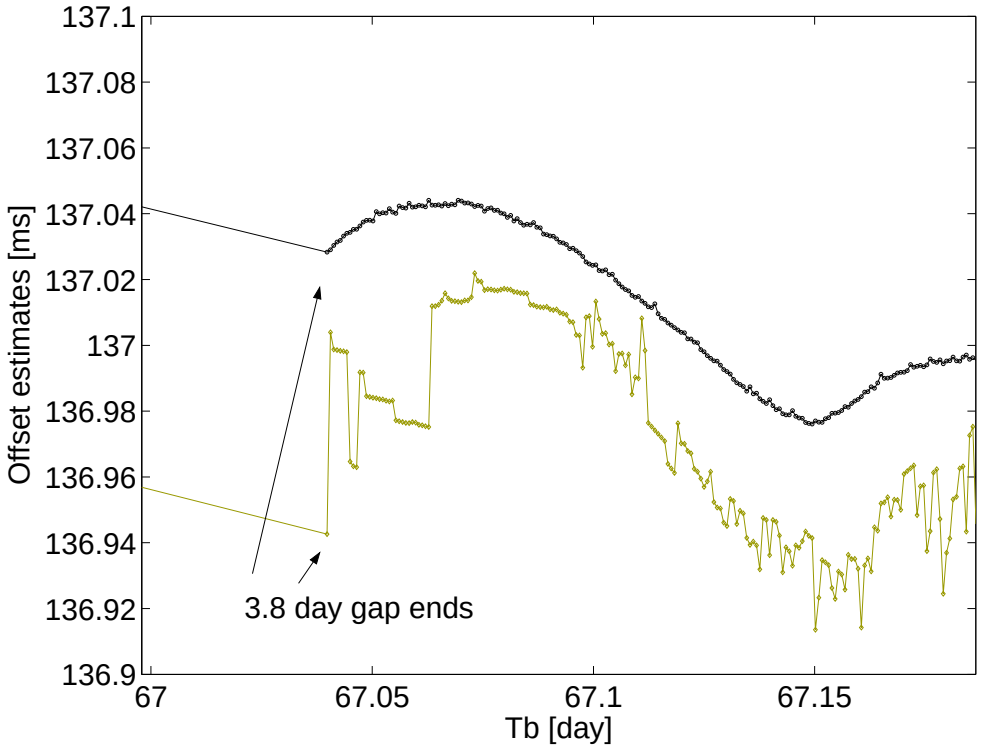




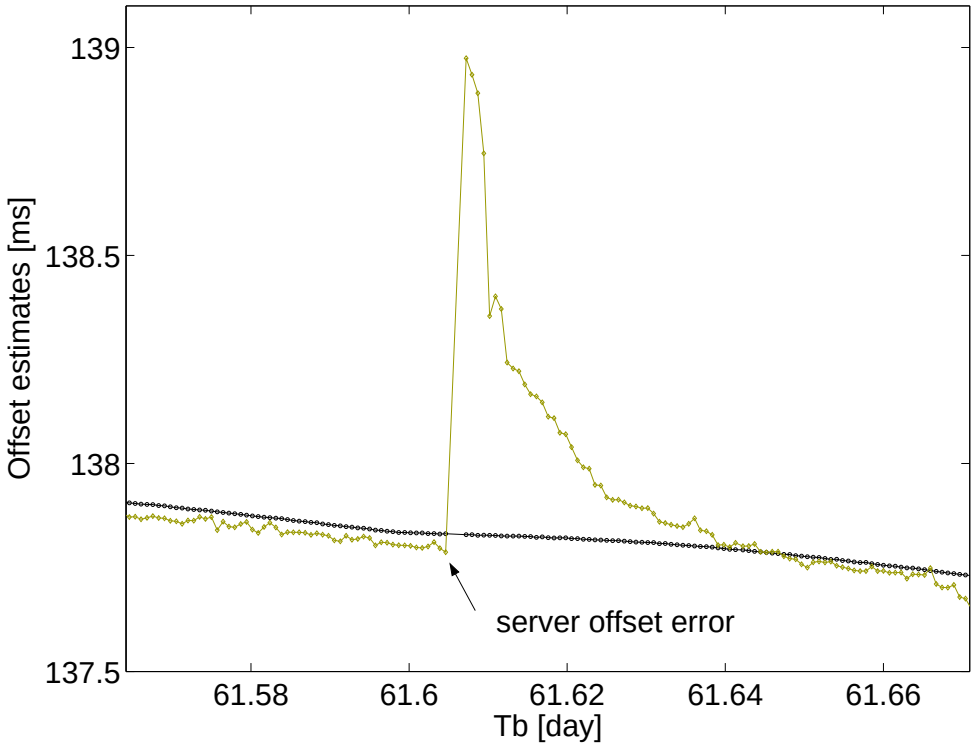
Histograms show 99% of all values, $\tau' = 2\tau^*$

Zooming in on Extreme Events

Long Gap

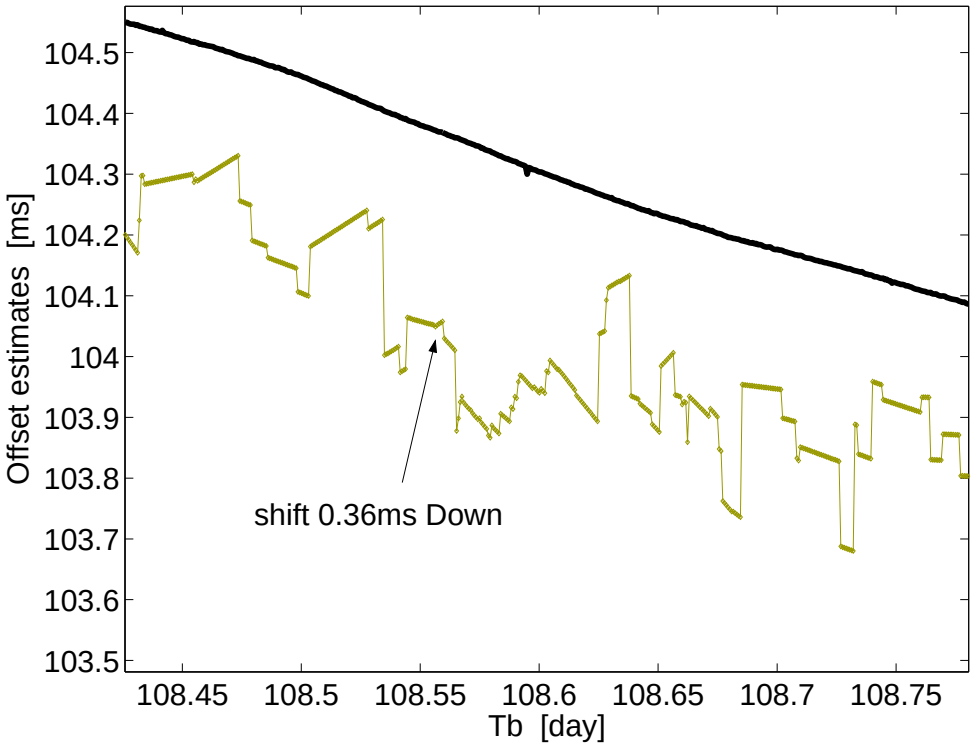


Server Error

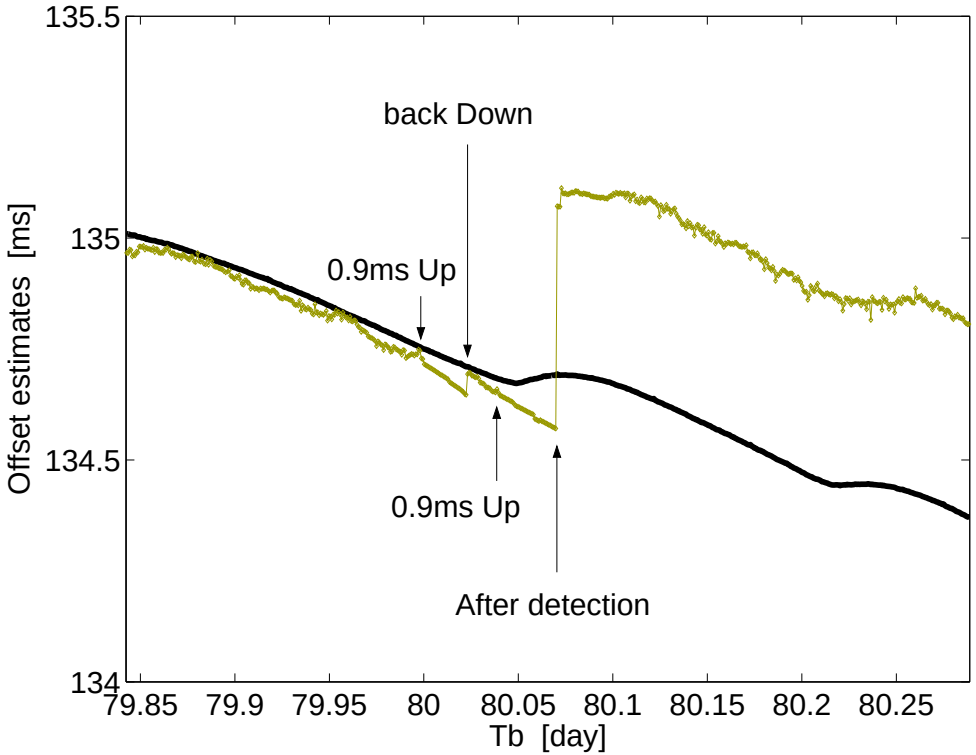


Zooming in on Extreme Events – Shifts

Natural Nice Shift



Induced Nasty Shifts



Conclusions

- Defined CPU oscillator based software clock for devices with TSC registers
- Absolute Clock:
  - more accurate than *SW–NTP*
  - far more robust
- Difference Clock:
  - not available under *SW–NTP*
  - more accurate than *SW–GPS* for many applications
- Low computational requirements
- Suited to network measurement, or as generic replacement for *SW–NTP*

Web Resources

- Publications (including preprint):
  http://www.cubinlab.ee.mu.oz.au/probing/articles.shtml
- Software and documentation (Linux) for TSC clock and early p calibration:
  http://www.cubinlab.ee.mu.oz.au/probing/software.shtml
- Software for new algorithms: BSD implementation coming
  http://www.cubinlab.ee.mu.oz.au/probing/software.shtml
- TSC based sender/receiver (Linux, LinuxTSC, RT-Linux) for active probing:
  http://www.cubinlab.ee.mu.oz.au/probing/software.shtml