Time Synchronization for Protocol Designers

Daniel Franke
Security Architect

Akamai Technologies

2017-05-18T16:20:00.000Z

Why is time synchronization so handy?

- Short-lived credentials
  - Kerberos
  - OCSP stapling
- Sequencing events in distributed systems
  - Cryptocurrency
  - Distributed filesystems
  - Log files during DFIR

Talk outline

- Physics of timekeeping
- How NTP works
- Error bounds & security considerations

How consumer devices keep time

Consumer-grade devices that need to keep time (PCs, wall clocks, etc.) typically use a quartz oscillator (abbreviated XO).

Quartz is *piezoelectric*: mechanical stress causes electric potential, and *vice versa*. Put an amplifier across it, creating a positive feedback loop, and you get a nice sine wave at the resonant frequency of the crystal.

By knowing the resonant frequency and counting oscillations, we can keep time with decent precision.

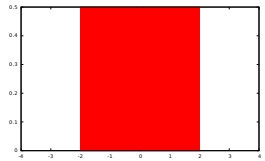
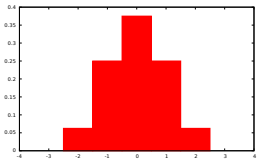
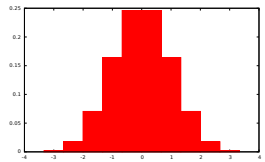
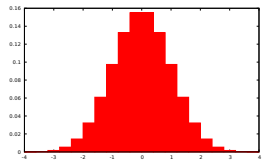
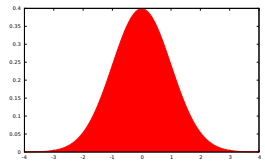
Photo credit: `oomlout.co.uk`

Sources of drift

1. **Manufacturing tolerance** — Typcial low-cost xos have a rated tolerance of 50 ppm (parts per million, or about 26 minutes per year.
2. **Aging** — Slow change (over years) in resonant frequency due primarily to outgassing.
3. **Temperature** — Thermal expansion & contraction alter resonant frequency.
4. **Wander** — Seemingly random fluctuation due to everything else we can't measure or control.

Random walks

Every $\frac{1}{n^2}$ seconds, flip a coin and step $\frac{1}{n}$ meters forward or back.
Where will you end up?

Step size	Time between steps	Position at $t = 1$
± 1	1	
$\pm \frac{1}{2}$	$\frac{1}{4}$	
$\pm \frac{1}{3}$	$\frac{1}{9}$	
$\pm \frac{1}{5}$	$\frac{1}{25}$	
$\pm \epsilon$	ϵ^2	

Wander as a random walk

An oscillator's wander can be modeled as a random walk. The speed at which it wanders is written ω. If the RMS (root-mean-square) difference between the oscillator's speed at the beginning and end of the day is 3 ppm, then

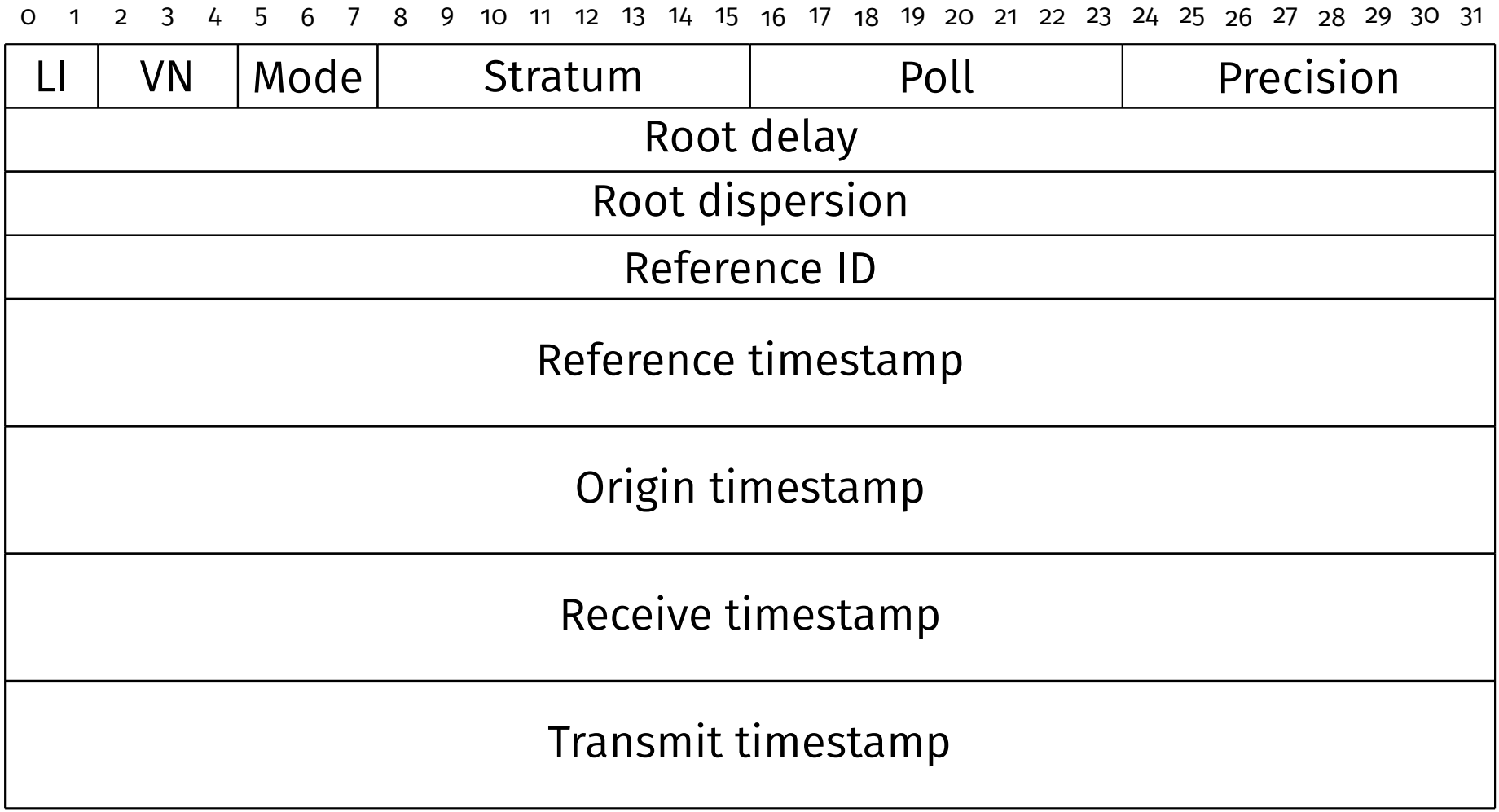
$$\omega = \frac{3\ \mathrm{ppm}}{\sqrt{\mathrm{day}}}.$$

For a lot of calculations it's easier to work with ω^2:

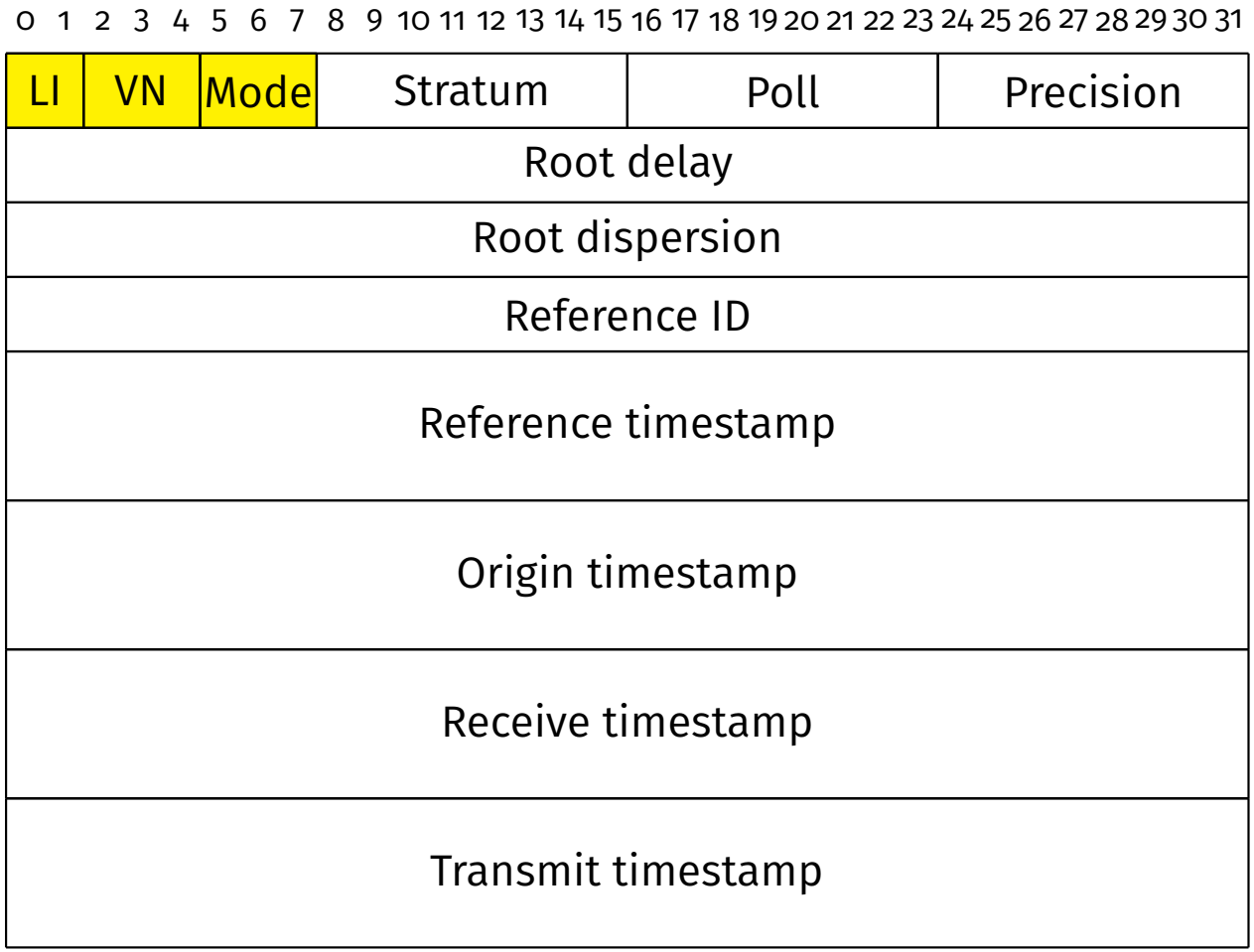
$$\omega^2 = \frac{9\ \mathrm{ppt}}{\mathrm{day}}.$$

ppt is a dimensionless ratio (10^{-12}), so we could write this as 104 aHz (attohertz!) if we so desired.

NTP packet

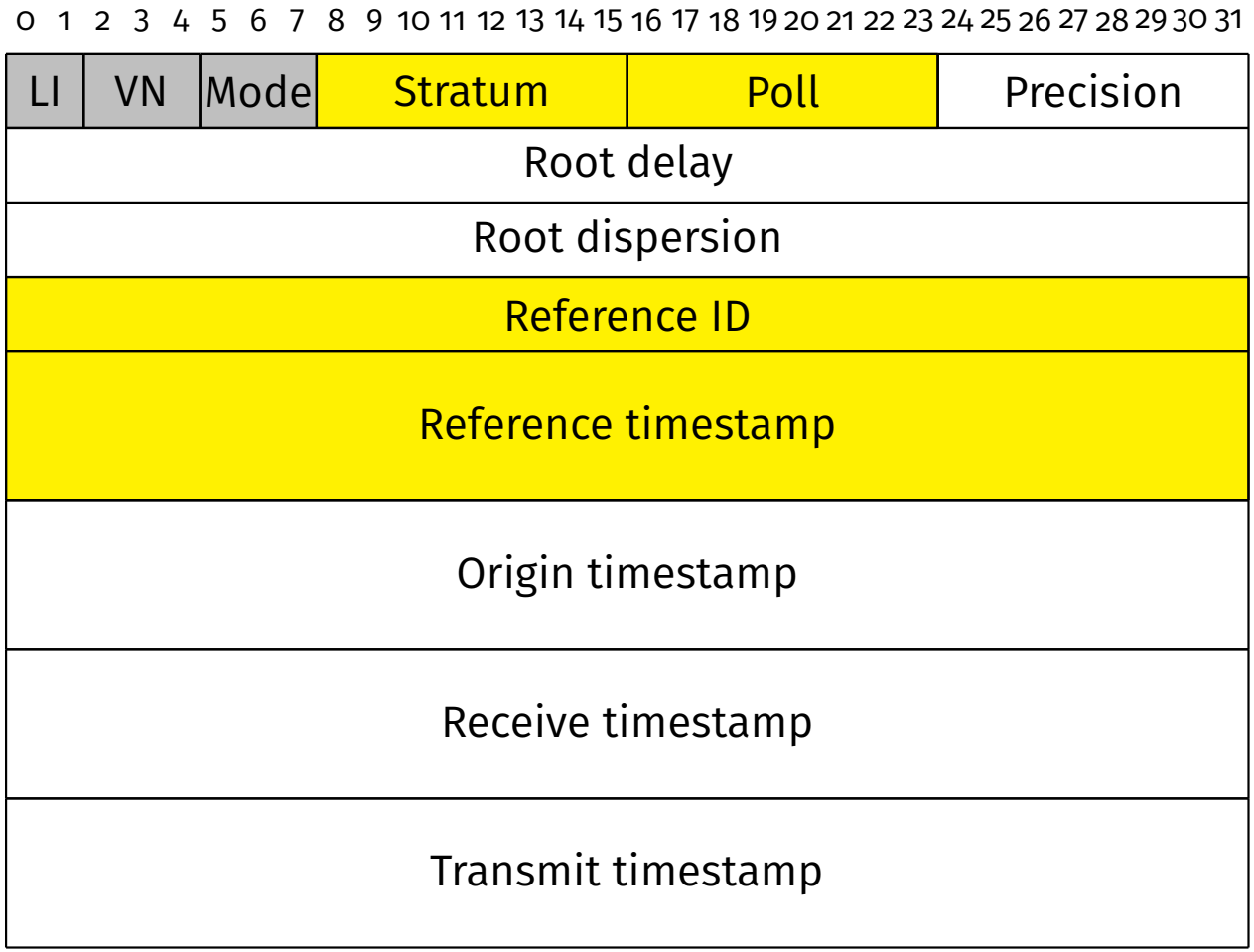

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
LI
VN
Mode
Stratum
Poll
Precision
Root delay
Root dispersion
Reference ID
Reference timestamp
Origin timestamp
Receive timestamp
Transmit timestamp

Uninteresting fields



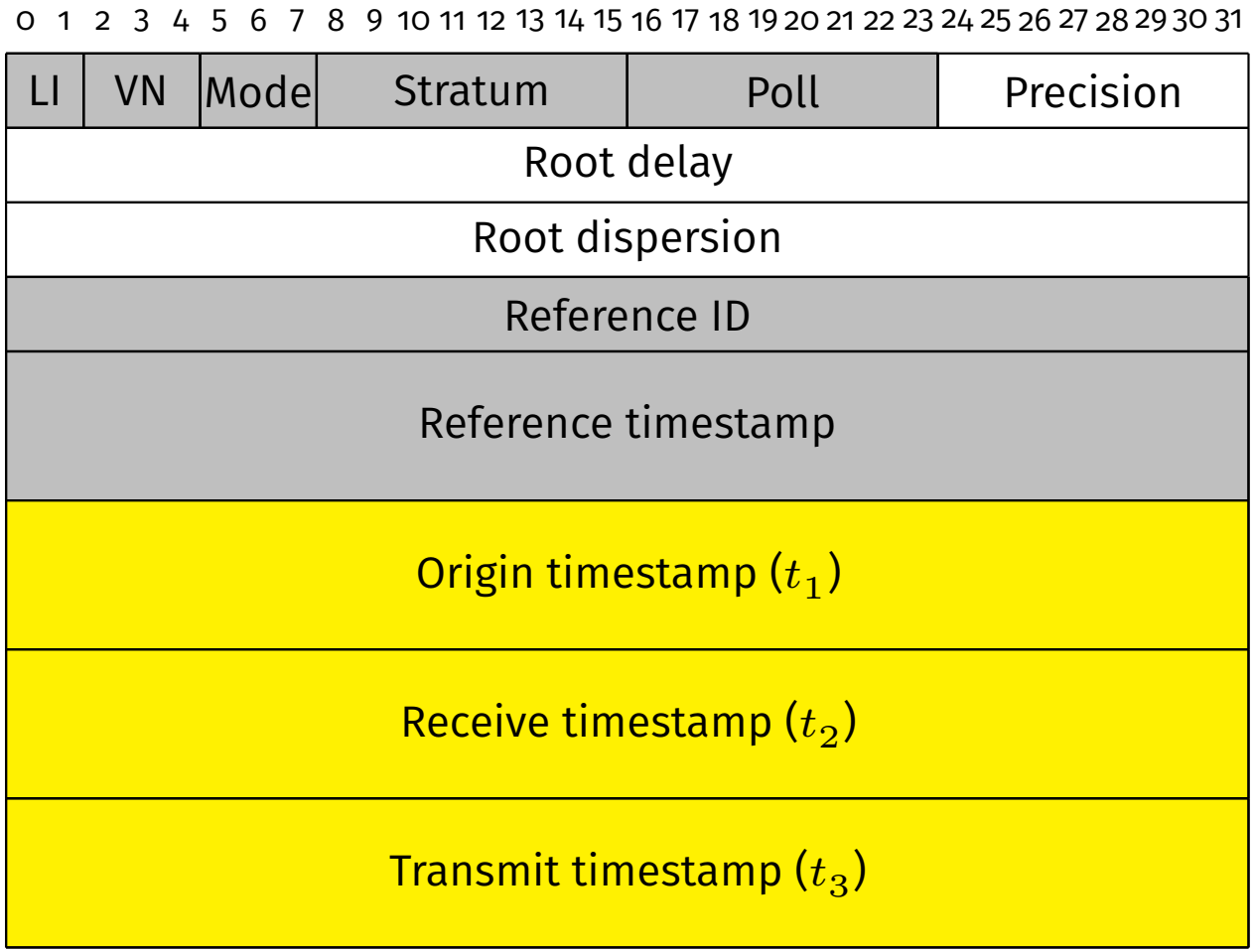
- VN – Protocol version (currently 4)
- Mode – 3 for queries, 4 for responses, other values for less-used modes
- LI – Notifies of upcoming leap second

Uninteresting fields (ctd.)



- Stratum — # of hops away from reference clock
- Poll — Polling interval
- Reference ID — Code identifying reference clock
- Reference timestamp — time of last communication with reference clock

Timestamp fields



NTP's Four Timestamps

- t_1, origin timestamp — time, according to the *client*, when *request was sent*.
- t_2, receive timestamp — time, according to the *server*, when *request was received*.
- t_3, transmit timestamp — time, according to the *server*, when *response was sent*.
- t_4, destination timestamp — time, according to the *client*, when *response was received*.

Basic Time Computation

Offset:

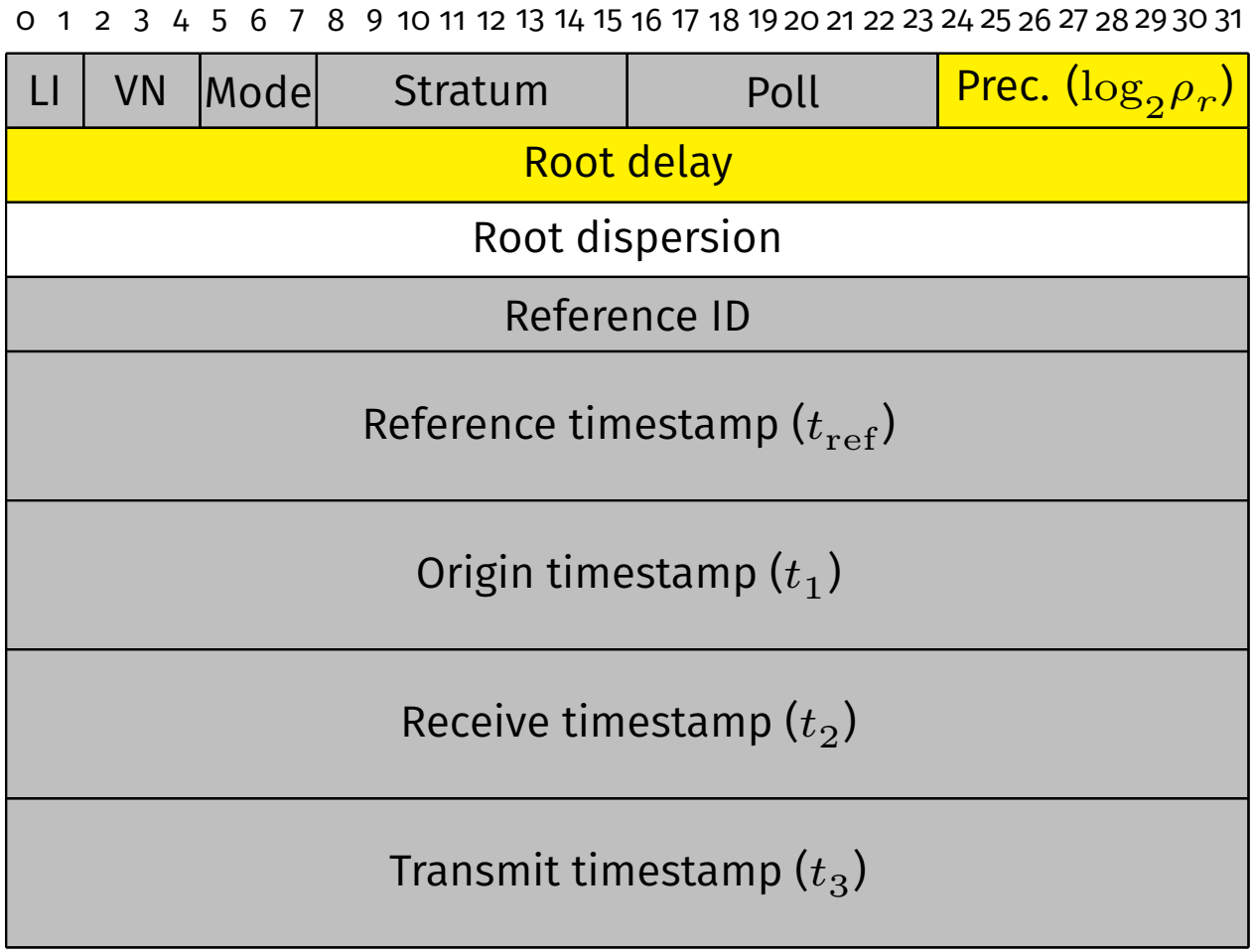
$$\text{Definition:} \quad \theta = t_{\text{server}} - t_{\text{client}}$$

$$\text{Estimate:} \quad \hat{\theta} = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$

Network round trip time:

$$\delta = (t_4 - t_1) - (t_3 - t_2)$$

Finding Maximum Error Bounds



- Precision: inherent error in obtaining system clock measurements due to hardware & os delays
- Root delay: server's own maximum error

Finding Error Bounds (ctd.)

Sources of error:

- Server's own error (root delay).
- Network asymmetry, bounded by $\frac{\delta}{2}$ for local hop.
- Clock drift since last measurement: $\Psi\left(t_{\mathrm{cur}} - t_1\right)$, $\Psi = 50\mathrm{ppm}$
- Clock-measurement imprecision of client and server, $\rho_r + \rho_s$

Sum these up, and you have a worst-case bound on the accuracy of your clock.

You can query this from the kernel! `man ntp_gettime(2)`

Cryptographic Security Options for NTP

- Legacy symmetric authentication: crypto is dubious but serviceable, key management can be painful
- Autokey: completely broken, don't use, even if you have no alternative
- Network Time Security: upcoming standard, switch when it's available
- IPSec tunnel: reasonable, overlooked option

Byzantine Errors

- NTP clients can (and should) be configured to take time from multiple sources
- For each server, assign an interval of plausible times: estimated time $\pm$ maximum error
- Search for majority clique of “truechimers” which are all plausible to each other
- Remaining “falsetickers” are automatically discarded

Delay Attacks

- NTP offers estimated as well as maximum error...
- ...but on an adversarial network, you can't trust it
- Deliberate introduction of delays in only one direction will push actual error toward the maximum
- Crypto can't possibly save you here

Conclusion

Computer clocks are not inherently unreliable — *if* you engineer your system properly.

Done right, secure and reliable clock synchronization to within one-way network latency is achievable.