Clock Synchronization

Part 2, Chapter 5

Roger Wattenhofer

Clock Synchronization

Clock Synchronization

Overview

- Motivation
- Real World Clock Sources, Hardware and Applications
- Clock Synchronization in Distributed Systems
- Theory of Clock Synchronization
- Protocol: PulseSync

Motivation

- Logical Time (“happened-before”)
  - Determine the order of events in a distributed system
  - Synchronize resources

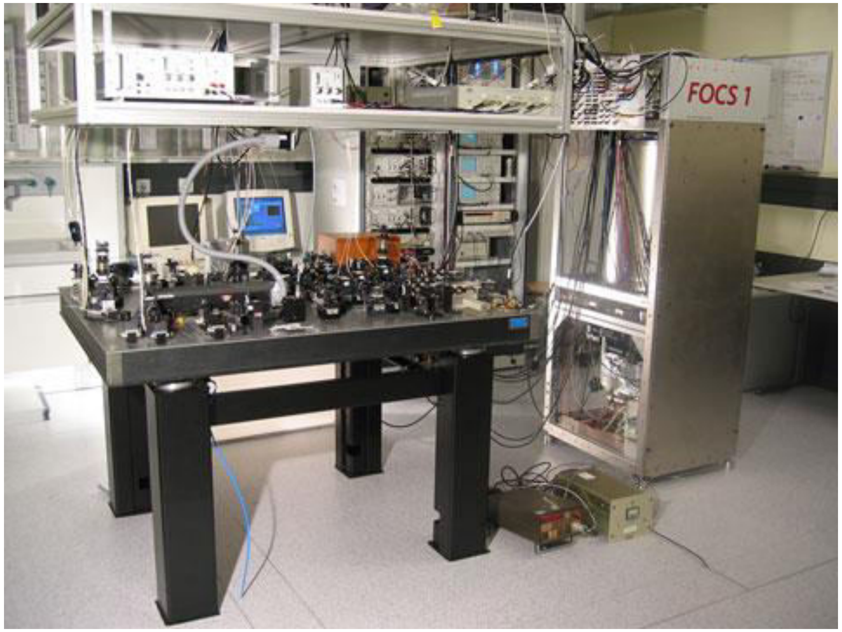
- Physical Time
  - Timestamp events (email, sensor data, file access times etc.)
  - Synchronize audio and video streams
  - Measure signal propagation delays (Localization)
  - Wireless (TDMA, duty cycling)
  - Digital control systems (ESP, airplane autopilot etc.)

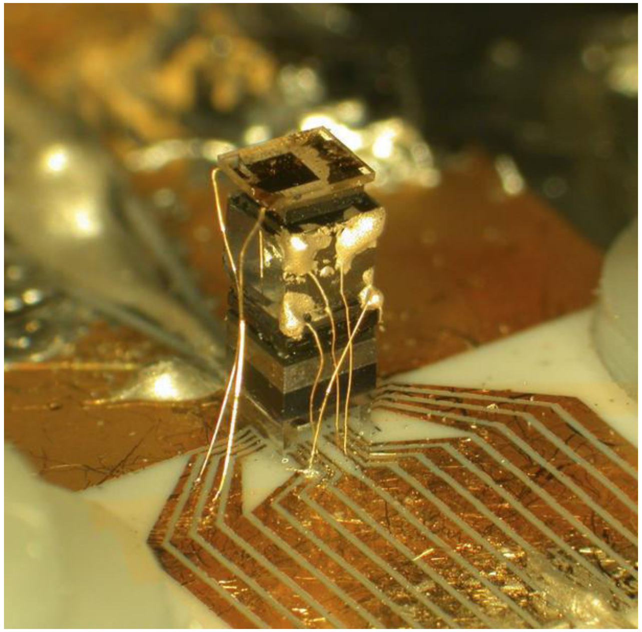
Properties of Clock Synchronization Algorithms

- External vs. internal synchronization
  - External sync: Nodes synchronize with an external clock source (UTC)
  - Internal sync: Nodes synchronize to a common time
    - to a leader, to an averaged time, ...
- One-shot vs. continuous synchronization
  - Periodic synchronization required to compensate clock drift
- Online vs. offline time information
  - Offline: Can reconstruct time of an event when needed
- Global vs. local synchronization (explained later)
- Accuracy vs. convergence time, Byzantine nodes, …

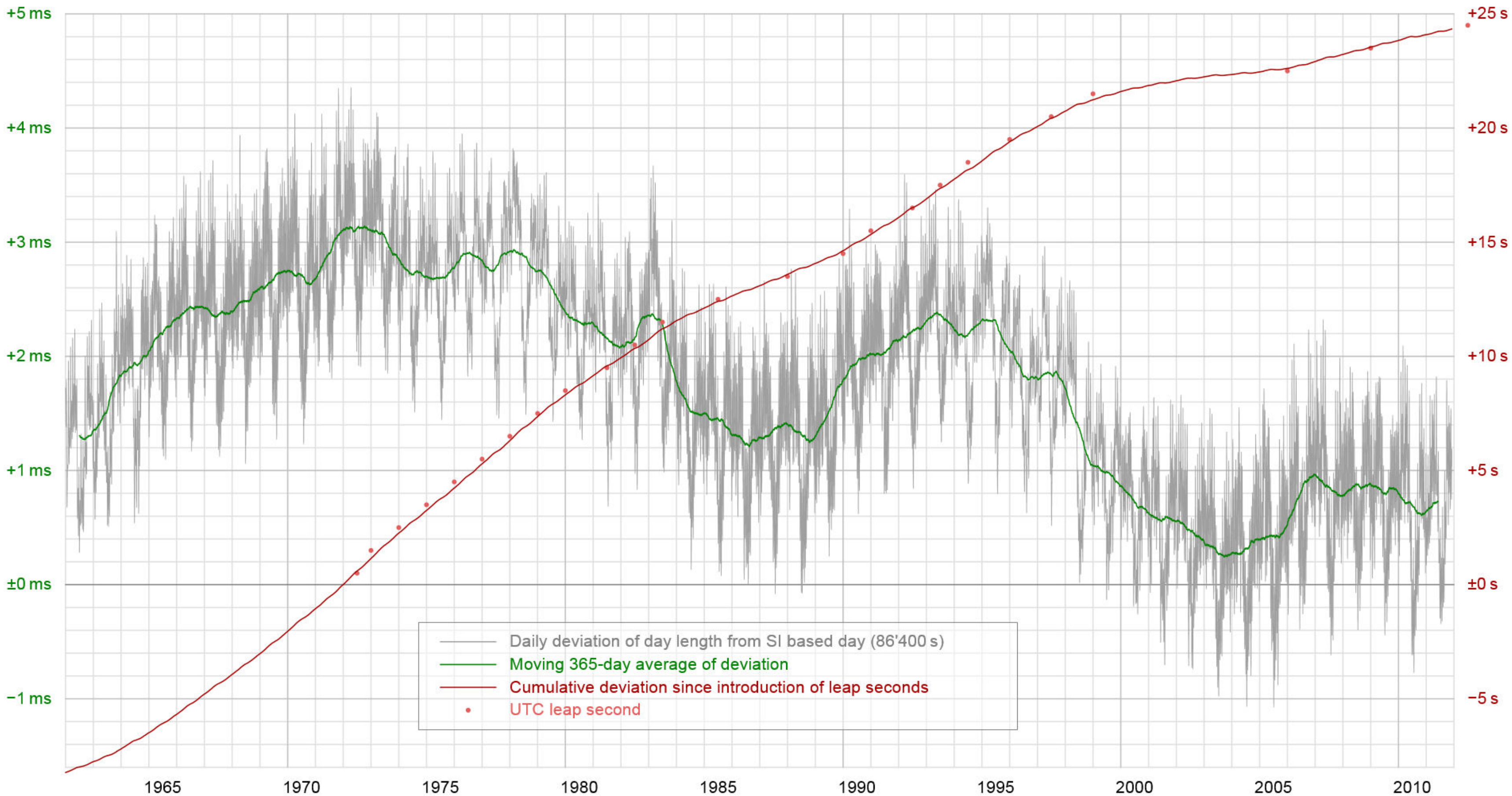
World Time (UTC)

- Atomic Clock
  - UTC: Coordinated Universal Time
  - SI definition 1s := 9192631770 oscillation cycles of the caesium-133 atom
  - Atoms are excited to oscillate at their resonance frequency and cycles can be counted.
  - Almost no drift (about 1s in 10 Million years)
  - Getting smaller and more energy efficient!





Atomic Clocks vs. Length of a Day



Access to UTC

- Radio Clock Signal
  - Clock signal from a reference source (atomic clock) is transmitted over a long wave radio signal
  - DCF77 station near Frankfurt, Germany transmits at 77.5 kHz with a transmission range of up to 2000 km
  - Accuracy limited by the propagation delay of the signal, Frankfurt-Zurich is about 1ms
  - Special antenna/receiver hardware required

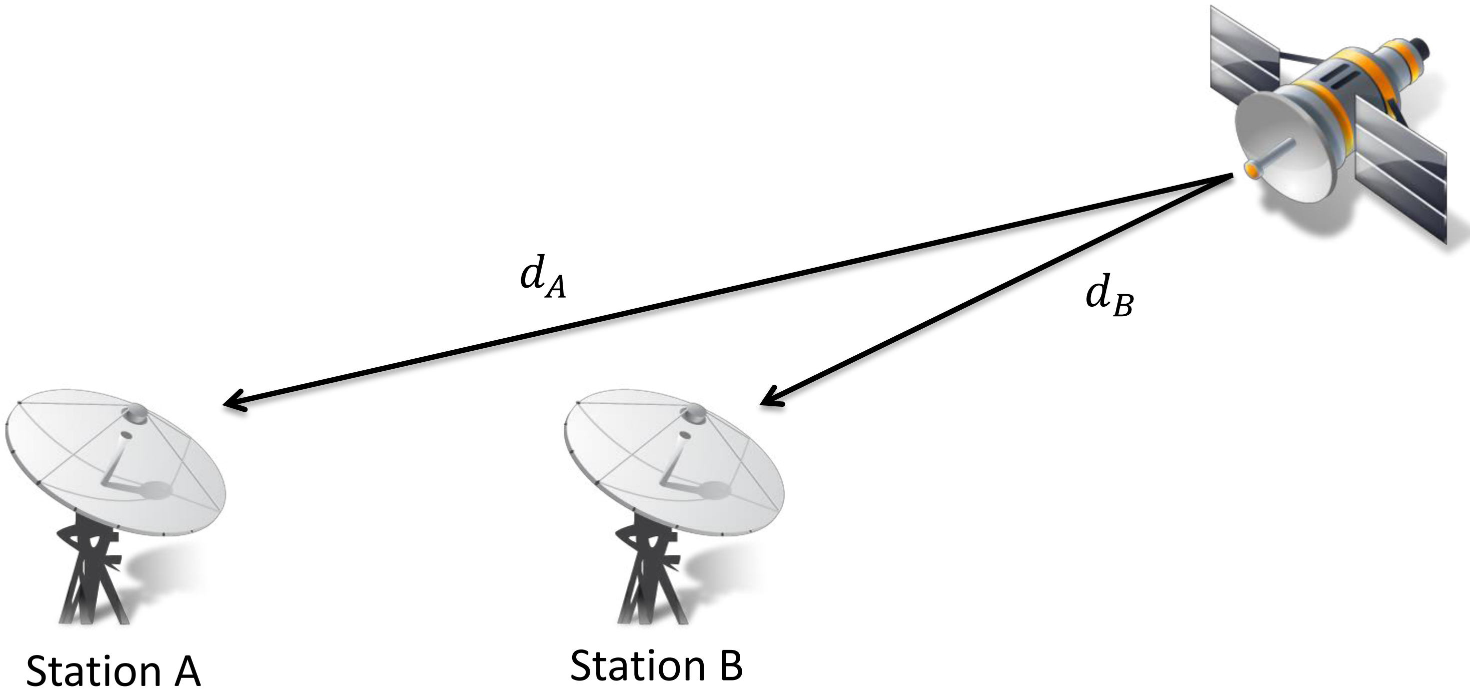
What is UTC, really?

- International Atomic Time (TAI)
  - About 200 atomic clocks
  - About 50 national laboratories
  - Reduce clock skew by comparing and averaging
  - UTC = TAI + UTC leap seconds (irregular rotation of earth)

BIPM

- GPS
  - USNO Time
  - USNO vs. TAI difference is a few nanoseconds

Comparing (and Averaging)



$t_{\Delta A} = t_A - (t_{SV}+d_A)$

$t_{\Delta B} = t_B - (t_{SV}+d_B)$

$$t_\Delta = t_{\Delta B} - t_{\Delta A} = t_B - (t_{SV} + d_B) - t_A + (t_{SV} + d_A) = t_B - t_A + d_A - d_B$$

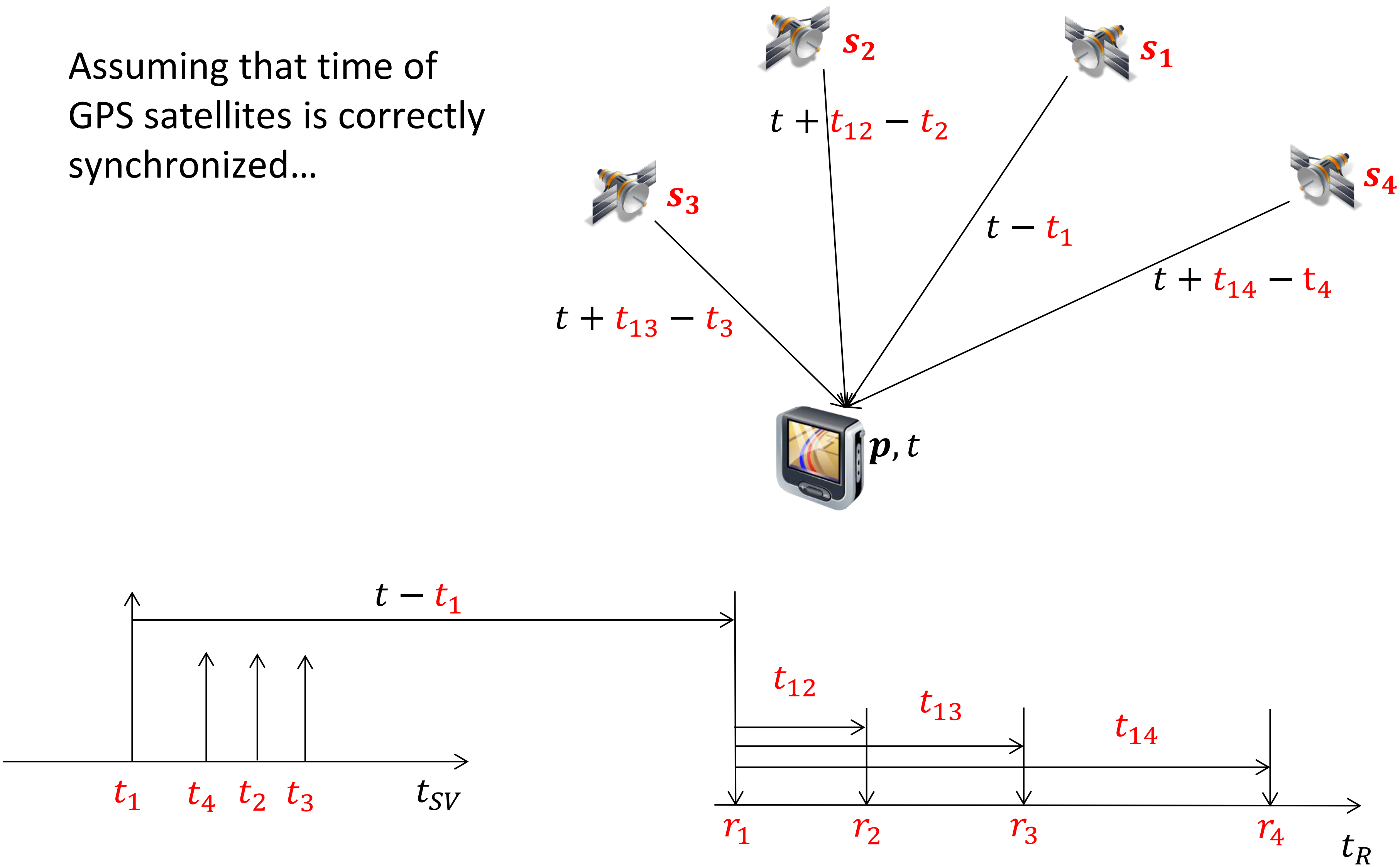
Global Positioning System (GPS)

- Satellites continuously transmit own position and time code
- Line of sight between satellite and receiver required
- Special antenna/receiver hardware required
- Time of flight of GPS signals varies between 64 and 89ms
- Positioning in space and time!

- Which is more accurate,
  GPS or Radio Clock Signal?

GPS Localization

Assuming that time of GPS satellites is correctly synchronized…



GPS Localization

$$\left\| \frac{\mathbf{s}_1 - \mathbf{p}}{c} \right\| = t - t_1$$

$$\left\| \frac{\mathbf{s}_2 - \mathbf{p}}{c} \right\| = t + t_{12} - t_2$$

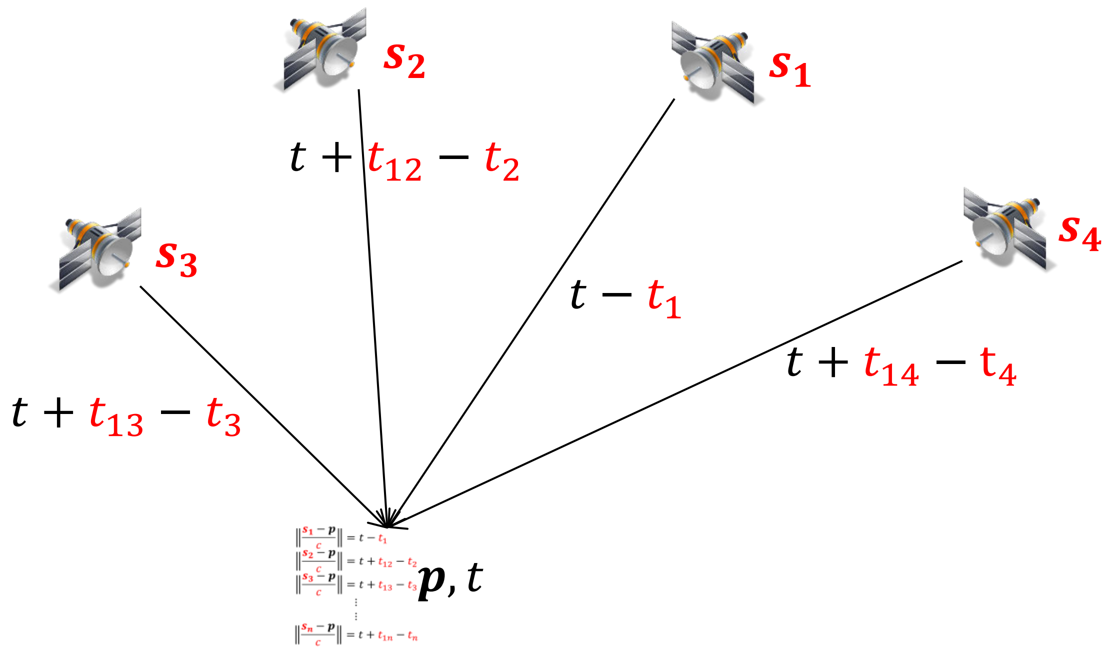
$$\left\| \frac{\mathbf{s}_3 - \mathbf{p}}{c} \right\| = t + t_{13} - t_3$$

$$\vdots$$

$$\vdots$$

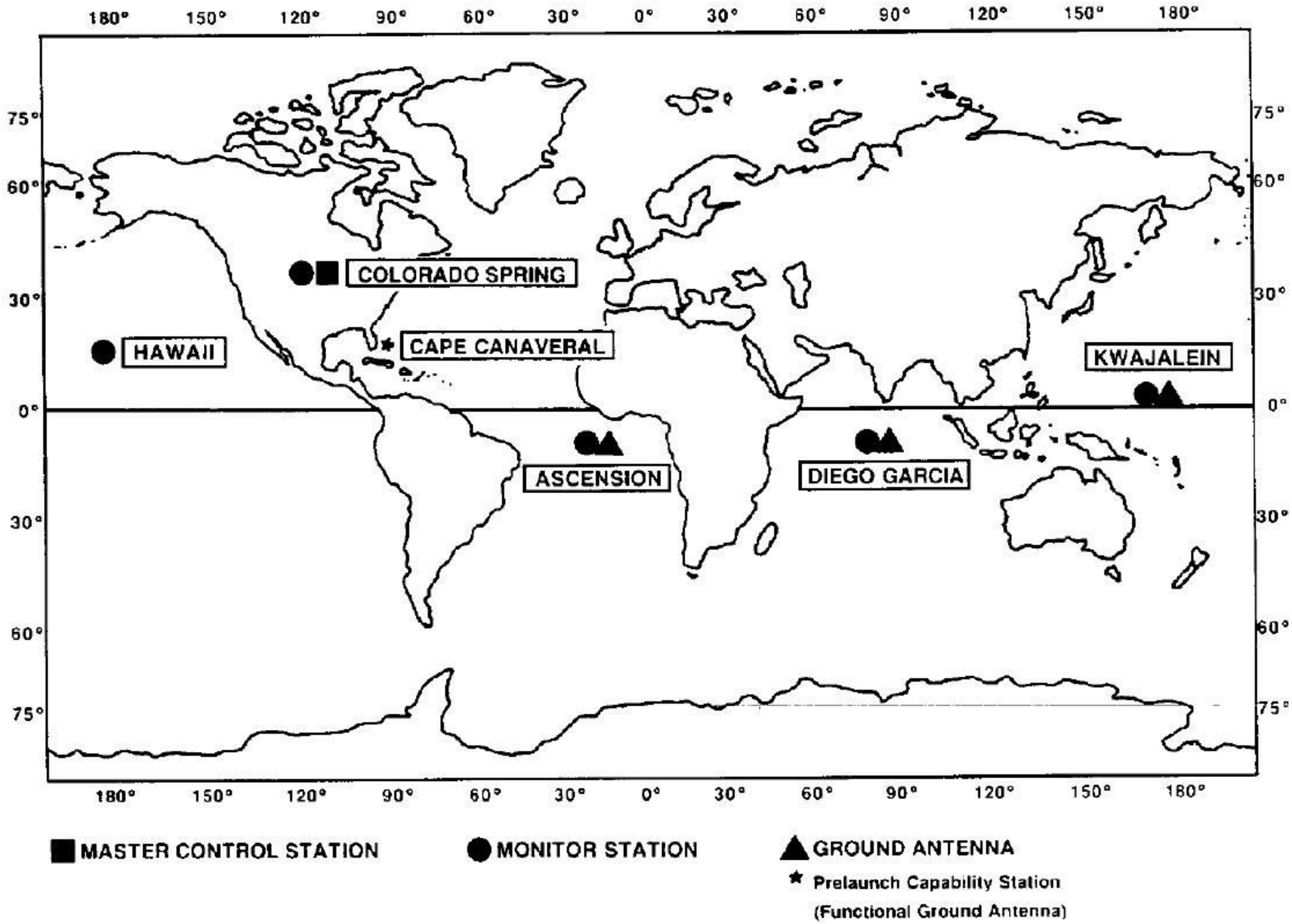
$$\left\| \frac{\mathbf{s}_n - \mathbf{p}}{c} \right\| = t + t_{1n} - t_n$$

c = speed of light



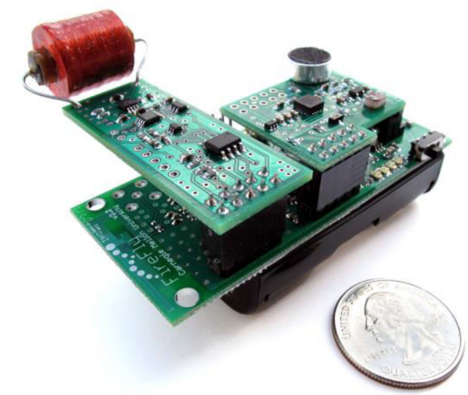
Find least squares solution in t and $\mathbf{p}$

Keeping GPS Satellites synchronized

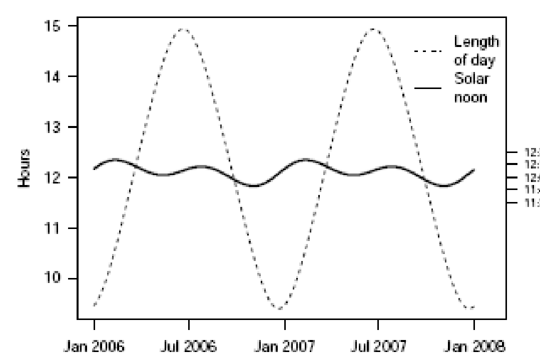


Alternative (Silly) Clock Sources

- AC power lines
  - Use the magnetic field radiating from electric AC power lines
  - AC power line oscillations are extremely stable (drift about 10 ppm, ppm = parts per million)
  - Power efficient, consumes only 58 µW
  - Single communication round required to correct phase offset after initialization

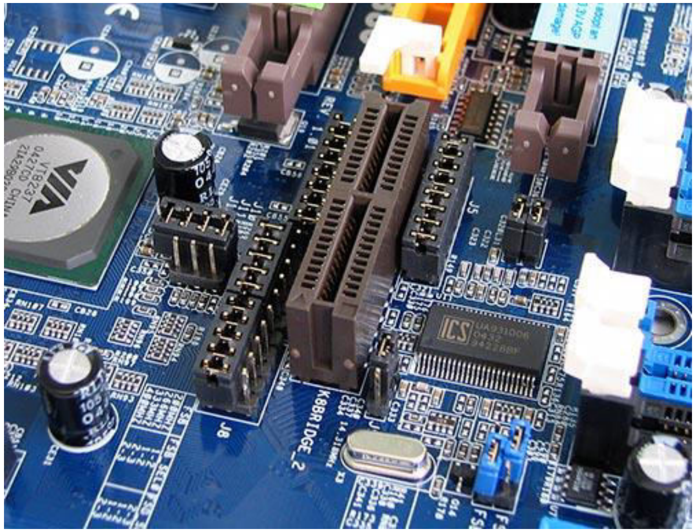


- Sunlight
  - Using a light sensor to measure the length of a day
  - Offline algorithm for reconstructing global timestamps by correlating annual solar patterns (no communication required)



Clock Devices in Computers

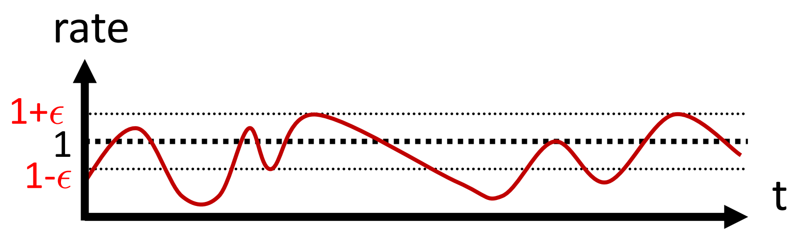
- Real Time Clock (IBM PC)
  - Battery backed up
  - 32.768 kHz oscillator + Counter
  - Get value via interrupt system



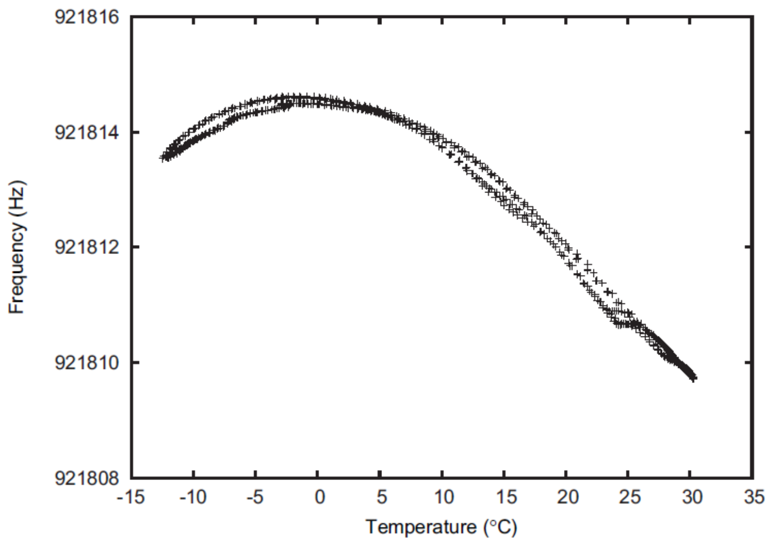
- HPET (High Precision Event Timer)
  - Oscillator: 10 Mhz … 100 Mhz
  - Up to 10 ns resolution!
  - Schedule threads
  - Smooth media playback
  - Usually inside Southbridge

Clock Drift

- Clock drift: random deviation from the nominal rate dependent on power supply, temperature, etc.



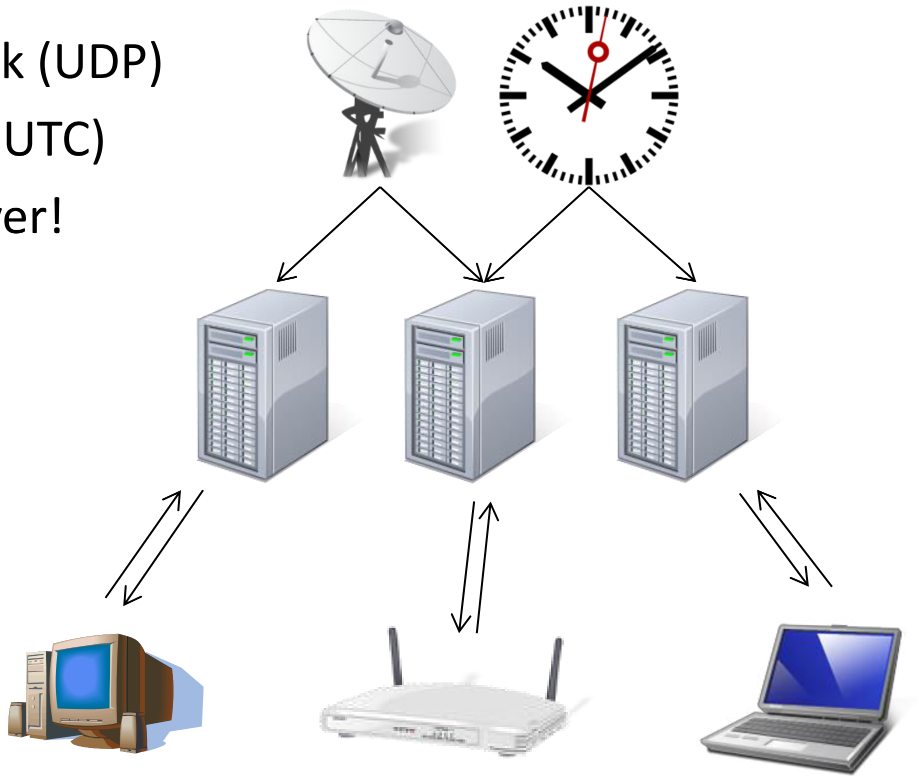
- E.g. TinyNodes have a maximum drift of 30-50 ppm (parts per million)



This is a drift of up to 50µs per second or 0.18s per hour

Clock Synchronization in Computer Networks

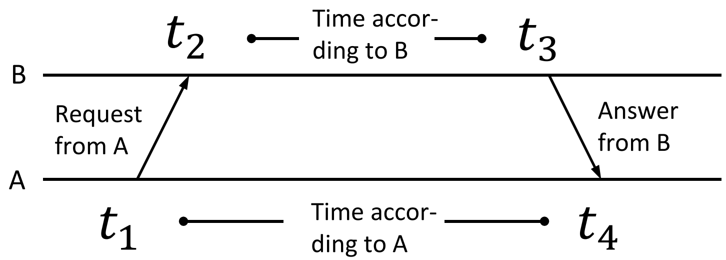
- Network Time Protocol (NTP)
- Clock sync via Internet/Network (UDP)
- Publicly available NTP Servers (UTC)
- You can also run your own server!



- Packet delay is estimated to reduce clock skew

Propagation Delay Estimation (NTP)

- Measuring the Round-Trip Time (RTT)



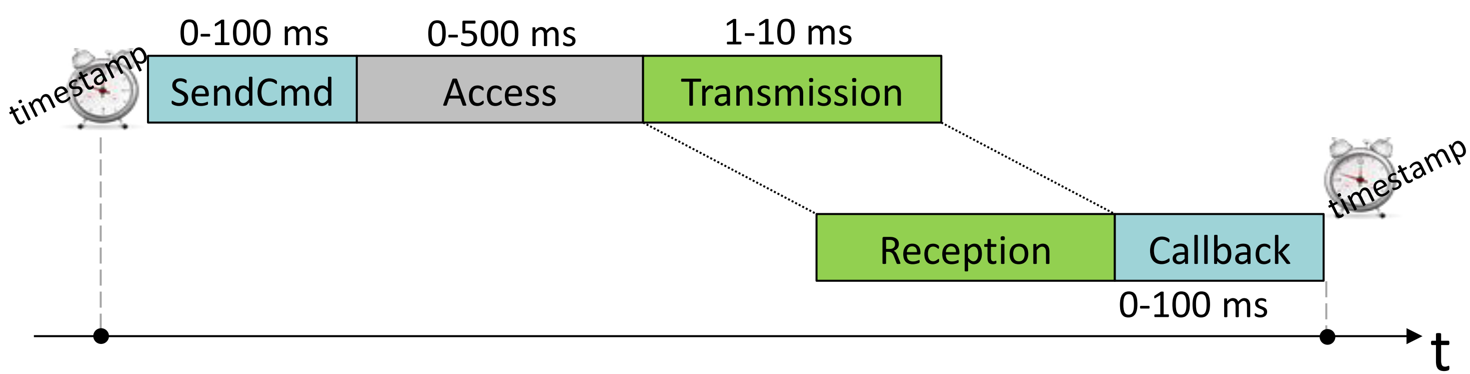
- Propagation delay δ and clock skew Θ can be calculated

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

$$\Theta = \frac{(t_2 - (t_1 + \delta)) - (t_4 - (t_3 + \delta))}{2} = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$

Messages Experience Jitter in the Delay

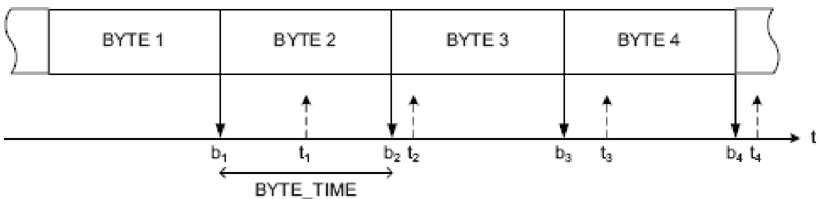
- Problem: Jitter in the message delay
  - Various sources of errors (deterministic and non-deterministic)

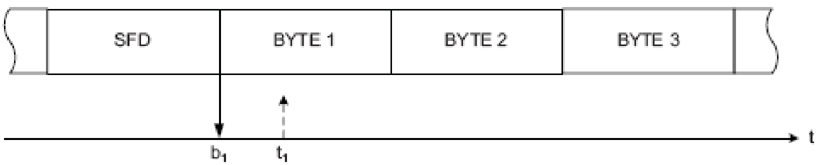


- Solution: Timestamping packets at the MAC layer
  - → Jitter in the message delay is reduced to a few clock ticks

Jitter Measurements

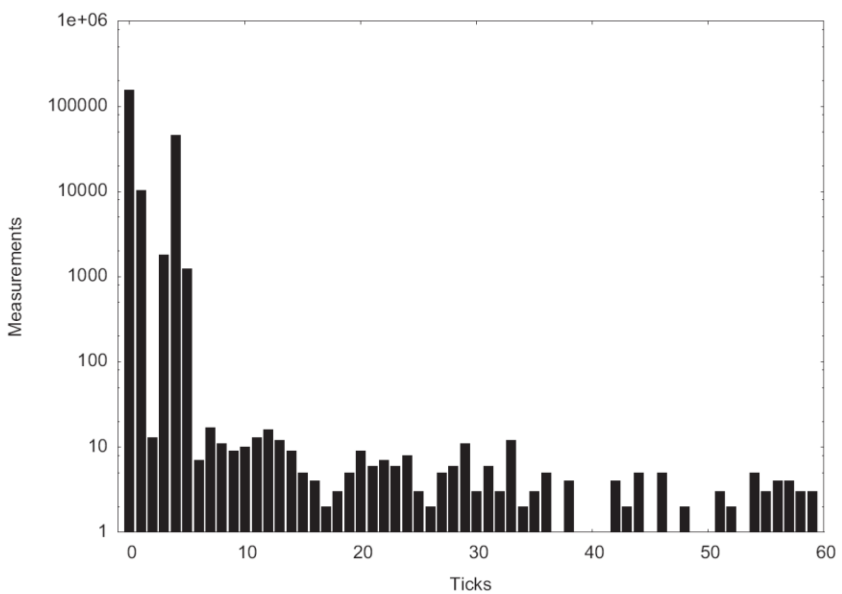
- Different radio chips use different paradigms
  - Left is a CC1000 radio chip which generates an interrupt with each byte.
  - Right is a CC2420 radio chip that generates a single interrupt for the packet after the start frame delimiter is received.





- In wireless networks propagation can be ignored (<1μs for 300m).

- Still there is quite some variance in transmission delay because of latencies in interrupt handling (picture right).

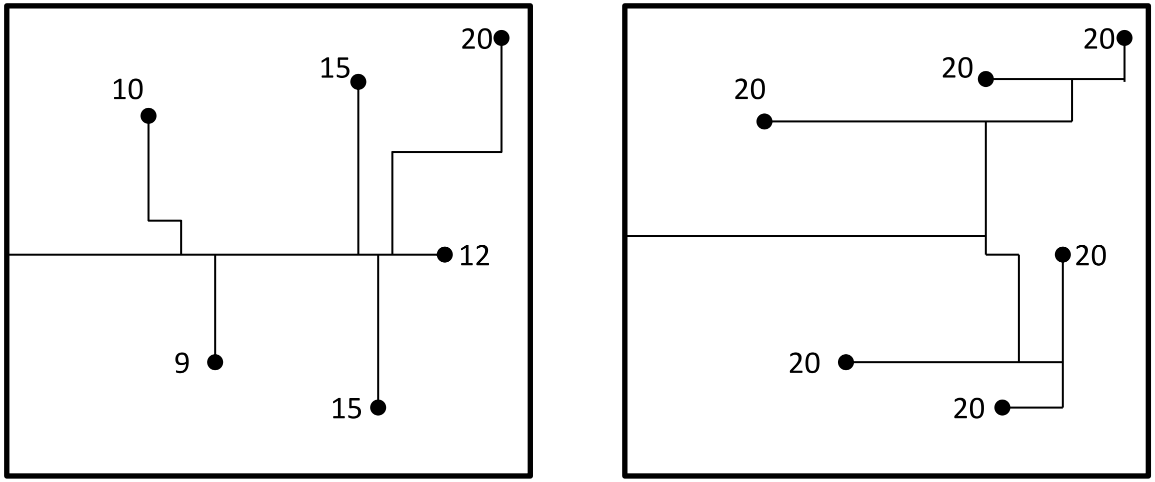


Clock Synchronization in Computer Networks (PTP)

- Precision Time Protocol (PTP) is very similar to NTP
- Commodity network adapters/routers/switches can assist in time sync by timestamping PTP packets at the MAC layer
- Packet delay is only estimated on request
- Synchronization through one packet from server to clients!
- Some newer hardware (1G Intel cards, 82580) can timestamp *any* packet at the MAC layer
- Achieving skew of about 1 microsecond

Hardware Clock Distribution

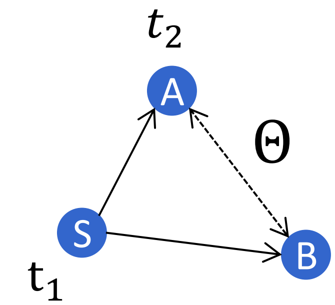
- Synchronous digital circuits require all components to act in sync



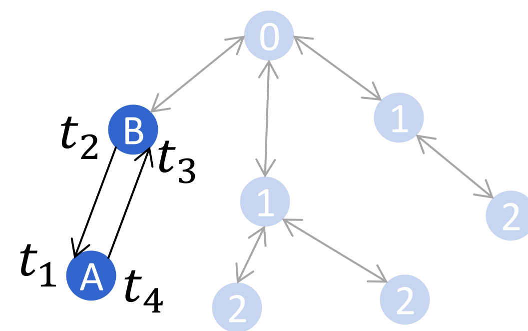
- The bigger the clock skew, the longer the clock period
- The clock signal that governs this rhythm needs to be distributed to all components such that skew and wire length is minimized
- Optimize routing, insert buffers (also to improve signal)

Clock Synchronization Tricks in Wireless Networks

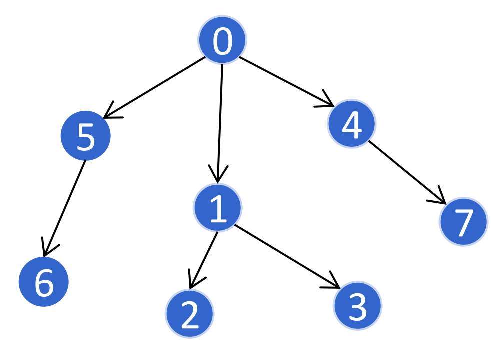
- Reference Broadcast Synchronization (RBS) ←→ Synchronizing atomic clocks
  - Sender synchronizes set of clocks



- Time-sync Protocol for Sensor Networks (TPSN) ←→ Network Time Protocol
  - Estimating round trip time to sync more accurately



- Flooding Time Synchronization Protocol (FTSP)←→Precision Time Protocol
  - Timestamp packets at the MAC Layer to improve accuracy



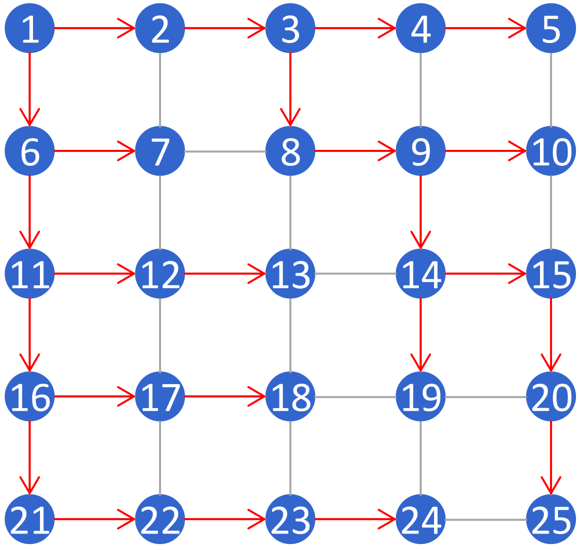
Best tree for tree-based clock synchronization?

- Finding a good tree for clock synchronization is a tough problem
  - Spanning tree with small (maximum or average) stretch.

- Example: Grid network, with $n = m^2$ nodes.

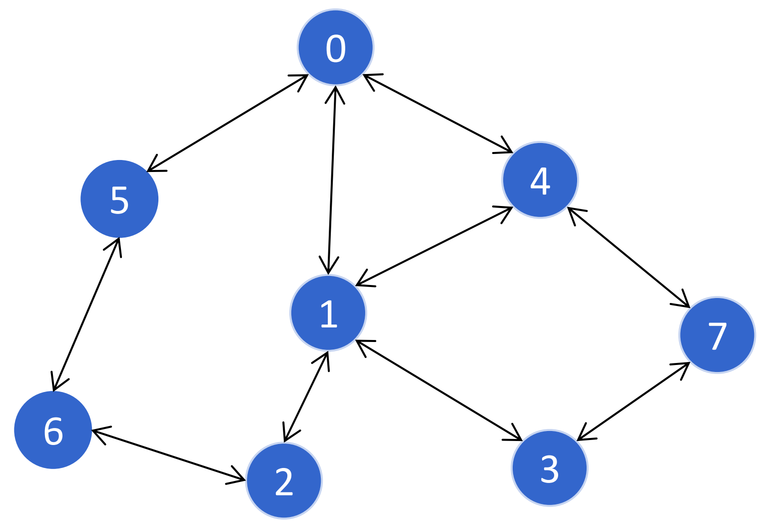
- No matter what tree you use, the maximum stretch of the spanning tree will always be at least m (just try on the grid).

- In general, finding the minimum max stretch spanning tree is a hard problem, however approximation algorithms exist.



Clock Synchronization Tricks (GTSP)

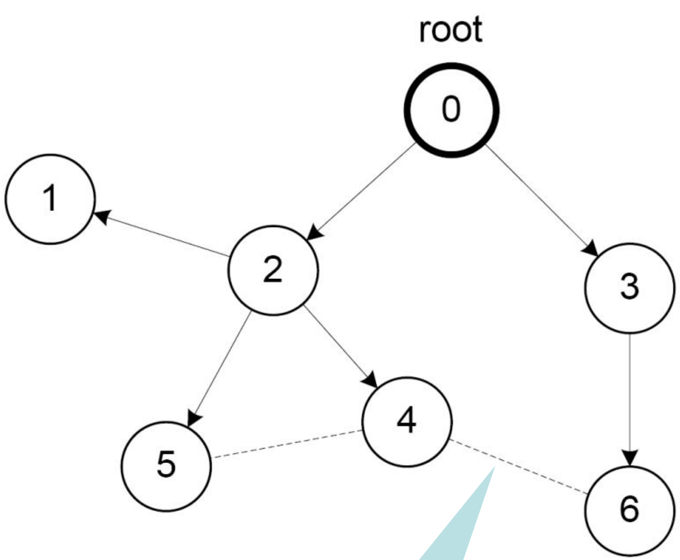
- Synchronize with *all* neighboring nodes
  - Broadcast periodic time beacons, e.g., every 30 s
  - No reference node necessary
- How to synchronize clocks without having a leader?
  - Follow the node with the fastest/slowest clock?
  - Idea: Go to the average clock value/rate of all neighbors (including node itself)



Variants of Clock Synchronization Algorithms

Tree-like Algorithms

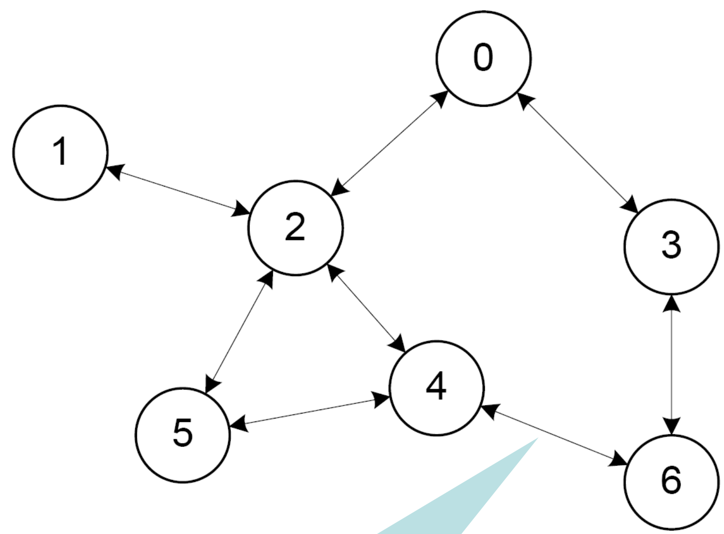
e.g. FTSP



Bad local skew

Distributed Algorithms

e.g. GTSP

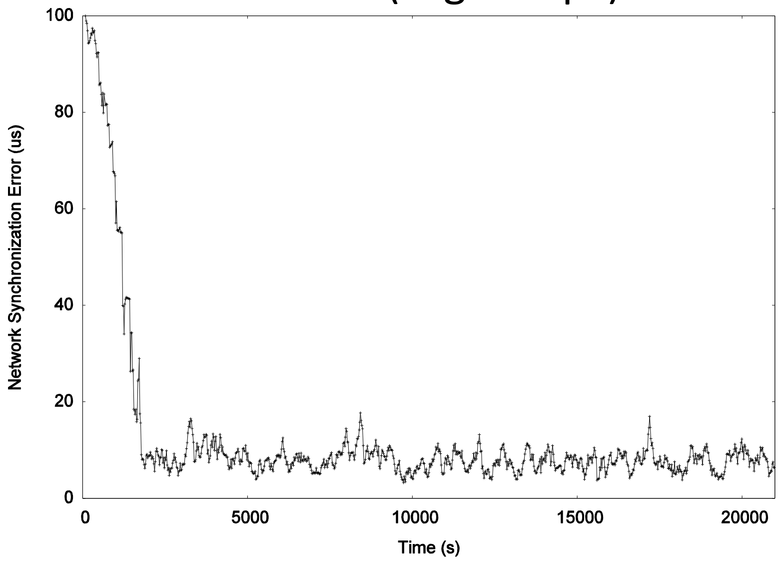


All nodes consistently average errors to *all* neigbhors

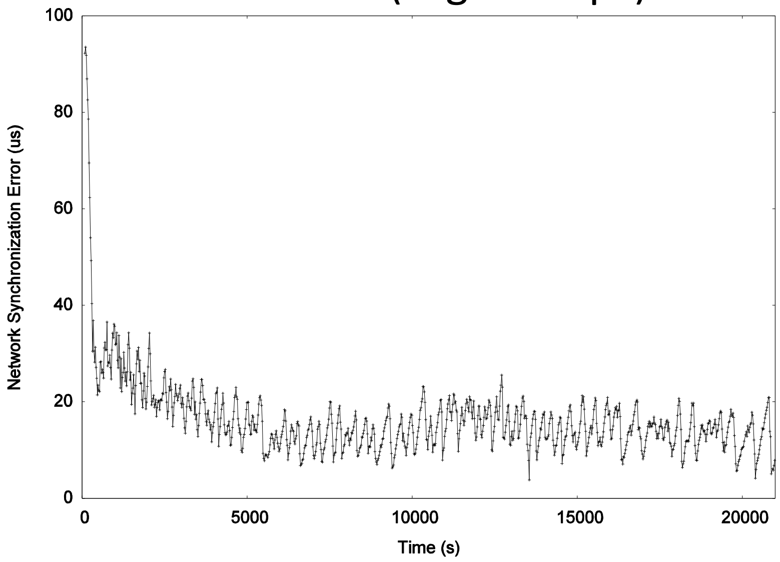
FTSP vs. GTSP: Global Skew

- Network synchronization error (global skew)
  - Pair-wise synchronization error between any two nodes in the network

FTSP (avg: 7.7 µs)



GTSP (avg: 14.0 µs)

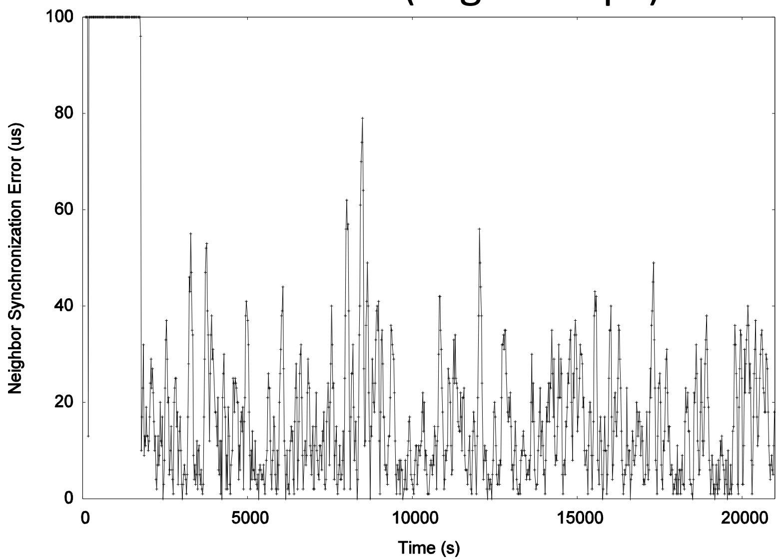


FTSP vs. GTSP: Local Skew

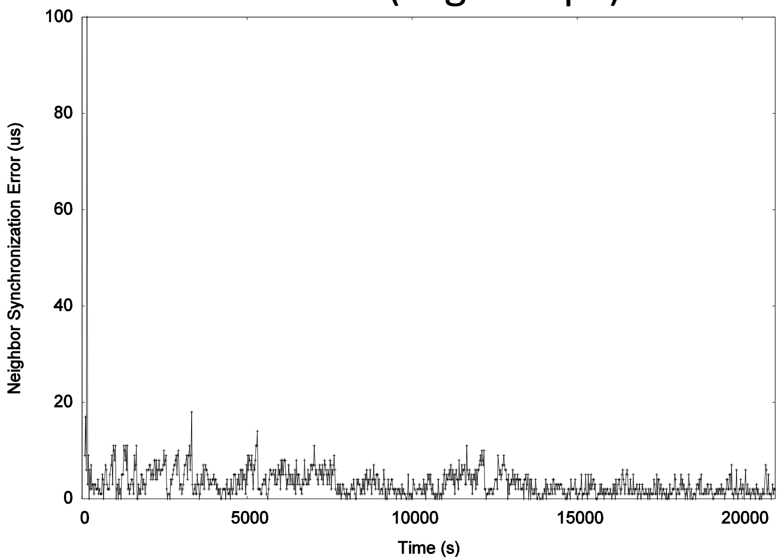
- Neighbor Synchronization error (local skew)
  - Pair-wise synchronization error between neighboring nodes

- Synchronization error between two direct neighbors:

FTSP (avg: 15.0 µs)

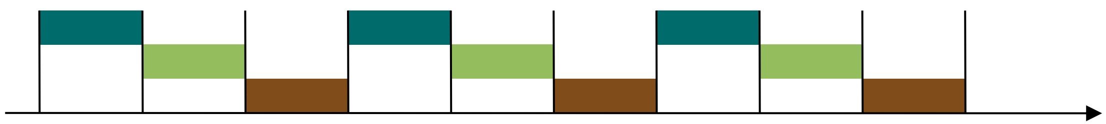


GTSP (avg: 2.8 µs)

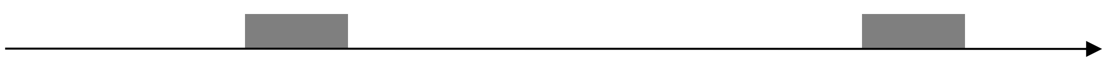


Global vs. Local Time Synchronization

- Common time is essential for many applications:
  - Global – Assigning a timestamp to a globally sensed event (e.g. earthquake)
  - Local – Precise event localization (e.g. shooter detection, multiplayer games)
  - Local – TDMA-based MAC layer in wireless networks



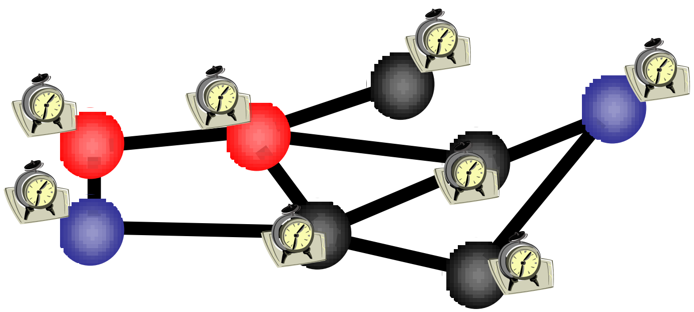
  - Local – Coordination of wake-up and sleeping times (energy efficiency)



Theory of Clock Synchronization

- Given a communication network
  1. Each node equipped with hardware clock with drift
  2. Message delays with jitter

worst-case (but constant)



- Goal: Synchronize Clocks ("Logical Clocks")
  - Both global and local synchronization!

Time Must Behave!

- Time (logical clocks) should not be allowed to stand still or jump

- Let’s be more careful (and ambitious):
- Logical clocks should always move forward
  - Sometimes faster, sometimes slower is OK.
  - But there should be a minimum and a maximum speed.
  - As close to correct time as possible!

Formal Model

- Hardware clock $H_v(t) = \int_{[0,t]} h_v(\tau)\, d\tau$ with clock rate $h_v(t) \in [1-\epsilon, 1+\epsilon]$

Clock drift ϵ is typically small, e.g. $\epsilon \approx 10^{-4}$ for a cheap quartz oscillator

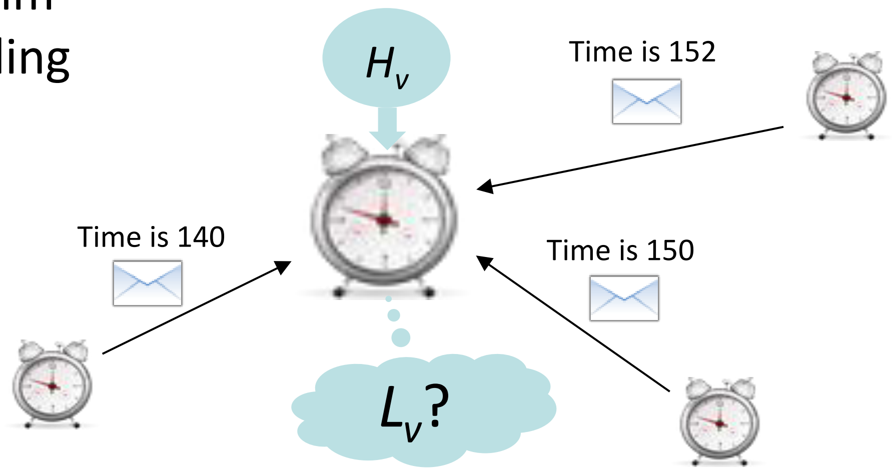
- Logical clock $L_v(\cdot)$ which increases at rate at least 1 and at most β

Logical clocks with rate less than 1 behave differently ("synchronizer")

- Message delays $\in [0,1]$

Neglect fixed share of delay, normalize jitter

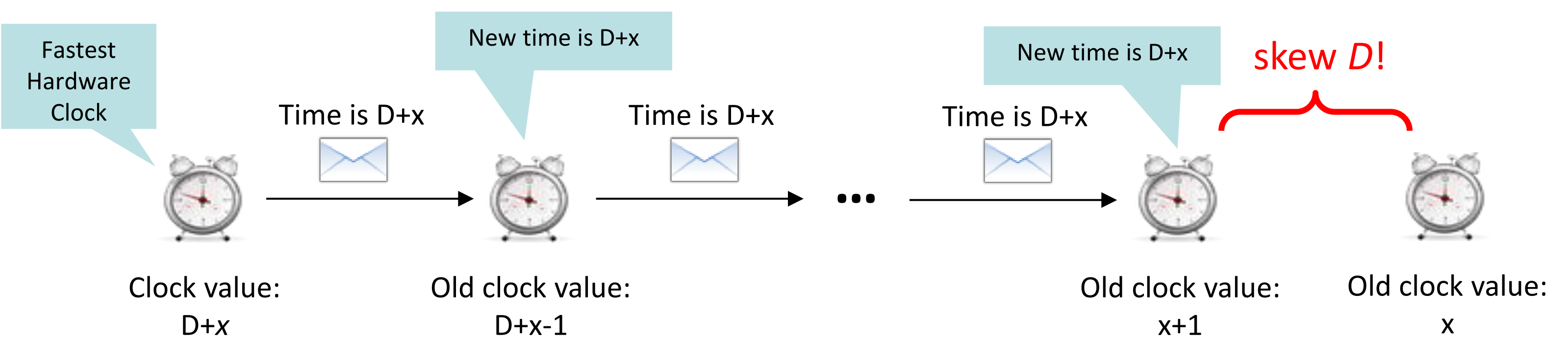
- Employ a synchronization algorithm to update the logical clock according to hardware clock and messages from neighbors



Synchronization Algorithms: An Example ("A^{max}")

Allow $\beta = \infty$

- Question: How to update the logical clock based on the messages from the neighbors?
- Idea: Minimizing the skew to the fastest neighbor
  - Set the clock to the maximum clock value received from any neighbor (if larger than local clock value)
  - forward new values immediately
- Optimum global skew of about *D*
- Poor local property
  - First all messages take 1 time unit…
  - …then we have a fast message!



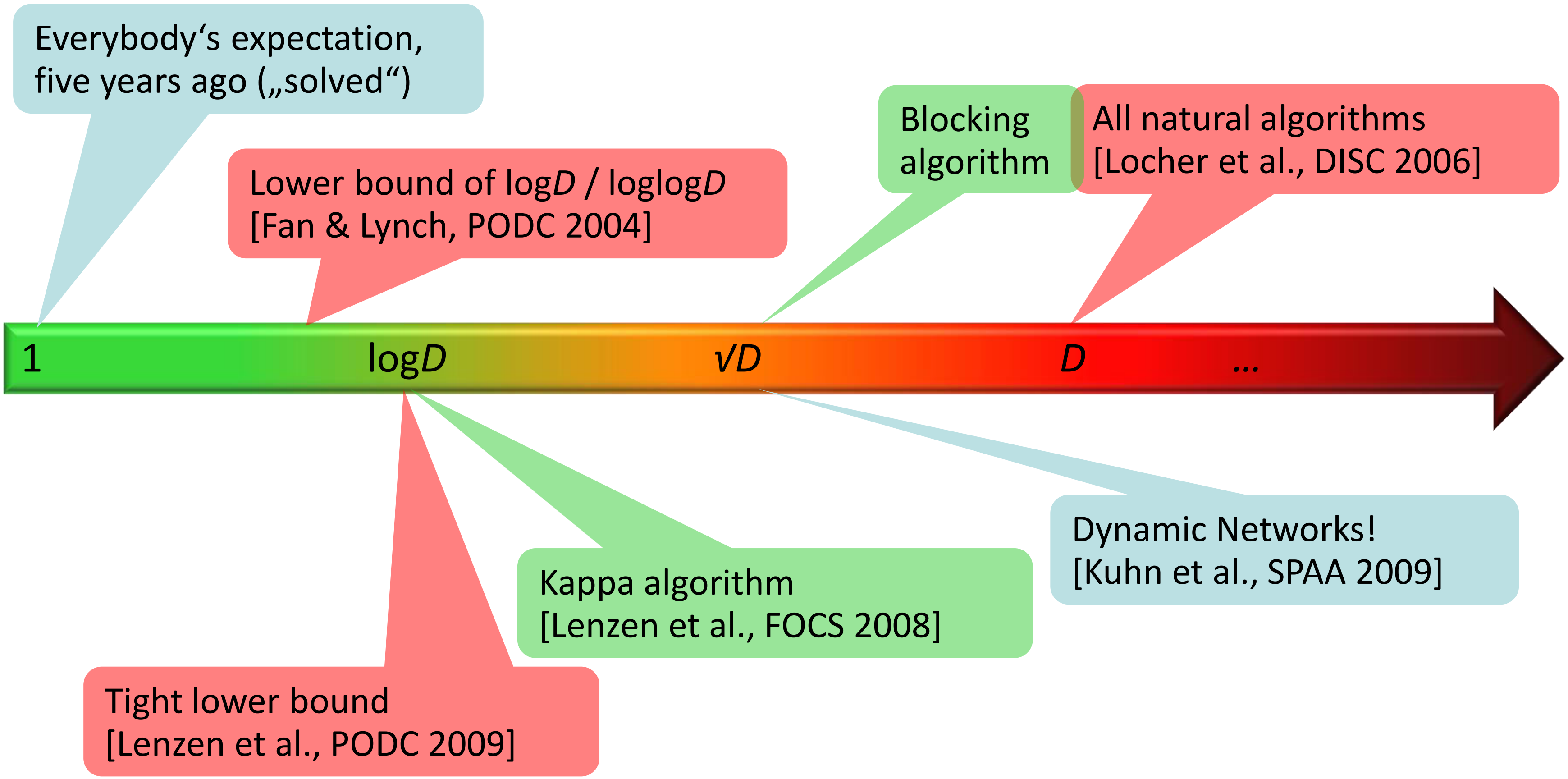
Synchronization Algorithms: $A^{\max'}$

- The problem of A^{max} is that the clock is always increased to the maximum value
- Idea: Allow a constant slack γ between the maximum neighbor clock value and the own clock value
- The algorithm $A^{max'}$ sets the local clock value $L_i(t)$ to
  $L_{i(t)} := \max(L_i(t), \max_{j \in N_i} L_j(t) - \gamma)$

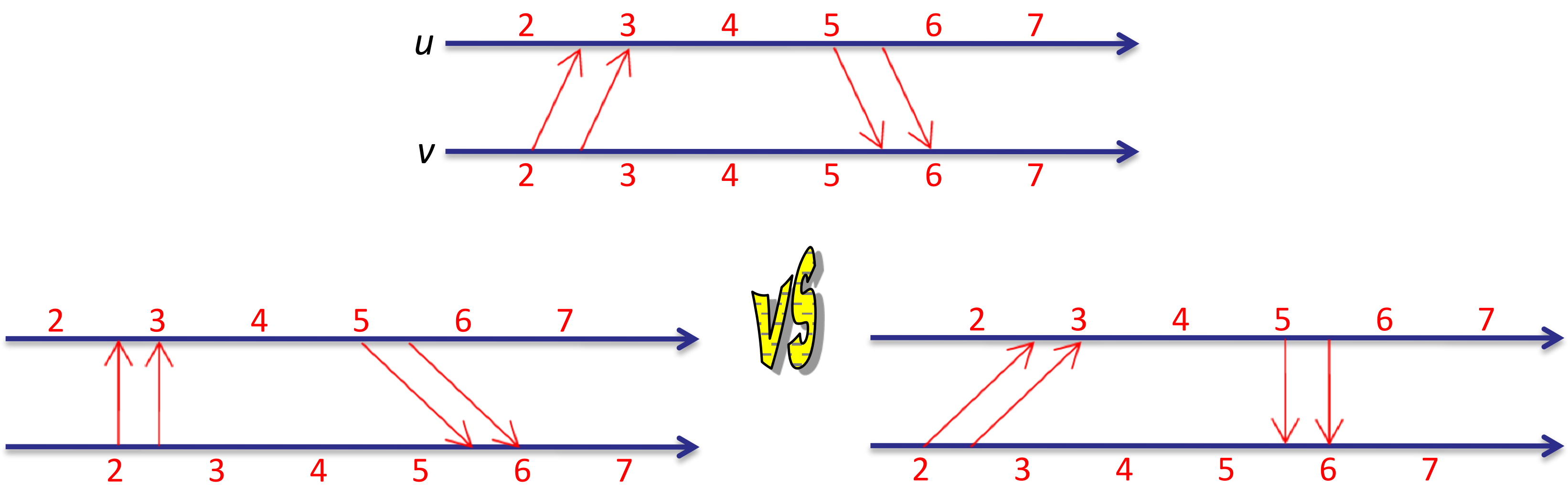
  → Worst-case clock skew between two neighboring nodes is still Θ(D) independent of the choice of γ!

- How can we do better?
  - Adjust logical clock speeds to catch up with fastest node (i.e. no jump)?
  - Idea: Take the clock of all neighbors into account by choosing the average value?

Local Skew: Overview of Results

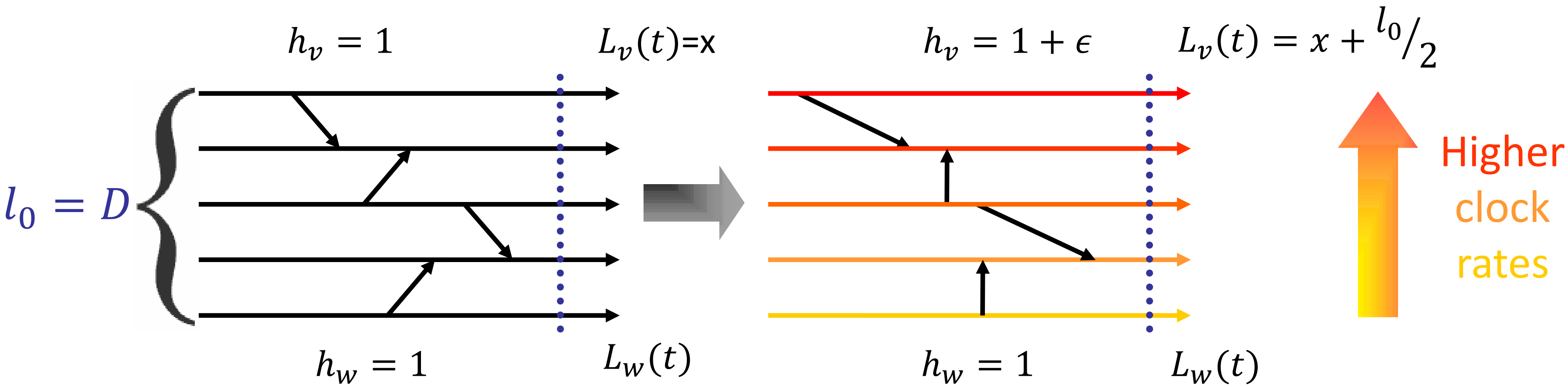


Enforcing Clock Skew



- Messages between two neighboring nodes may be fast in one direction and slow in the other, or vice versa.
- A constant skew between neighbors may be „hidden“.
- In a path, the global skew may be in the order of $D/2$.

Local Skew: Lower Bound



- Add ${}^{l_0}/_2$ skew in ${}^{l_0}/_{2\epsilon}$ time, messing with clock rates and messages
- Afterwards: Continue execution for ${}^{l_0}/_{4(\beta-1)}$ time (all $h_x = 1$)
  - → Skew reduces by at most ${}^{l_0}/_4$ → at least ${}^{l_0}/_4$ skew remains
  - → Consider a subpath of length $l_1 = l_0 \cdot {}^{\epsilon}/_{2(\beta-1)}$ with at least ${}^{l_1}/_4$ skew
  - → Add ${}^{l_1}/_2$ skew in ${}^{l_1}/_{2\epsilon} = {}^{l_0}/_{4(\beta-1)}$ time → at least ${}^3/_4 \cdot l_1$ skew in subpath
- Repeat this trick (+½,-¼,+½,-¼,…) $\log_{2(\beta-1)/\epsilon} D$ times

Theorem: $\Omega(\log_{\beta-1/\epsilon} D)$ skew between neighbors

Local Skew: Upper Bound

- Surprisingly, up to small constants, the $\Omega(\log_{(\beta-1)/\epsilon} D)$ lower bound can be matched with clock rates $\in [1,\beta]$ (tough part, not included)
- We get the following picture [Lenzen et al., PODC 2009]:

max rate β	$1+\epsilon$	$1+\Theta(\epsilon)$	$1+\sqrt{\epsilon}$	2	large
local skew	∞	$\Theta(\log D)$	$\Theta(\log_{1/\epsilon} D)$	$\Theta(\log_{1/\epsilon} D)$	$\Theta(\log_{1/\epsilon} D)$

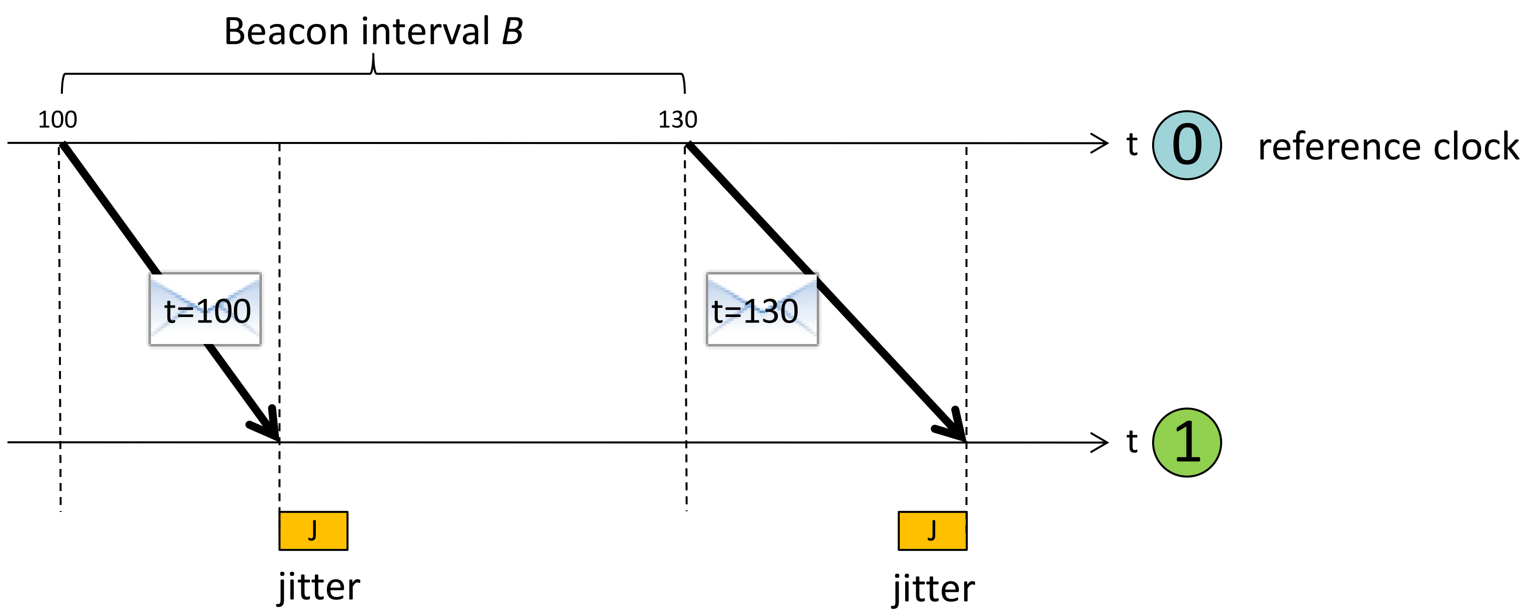
We can have both smooth and accurate clocks!

... because too large clock rates will amplify the clock drift ϵ.

- In practice, we usually have $1/\epsilon \approx 10^4 > D$. In other words, our initial intuition of a constant local skew was not entirely wrong! ☺

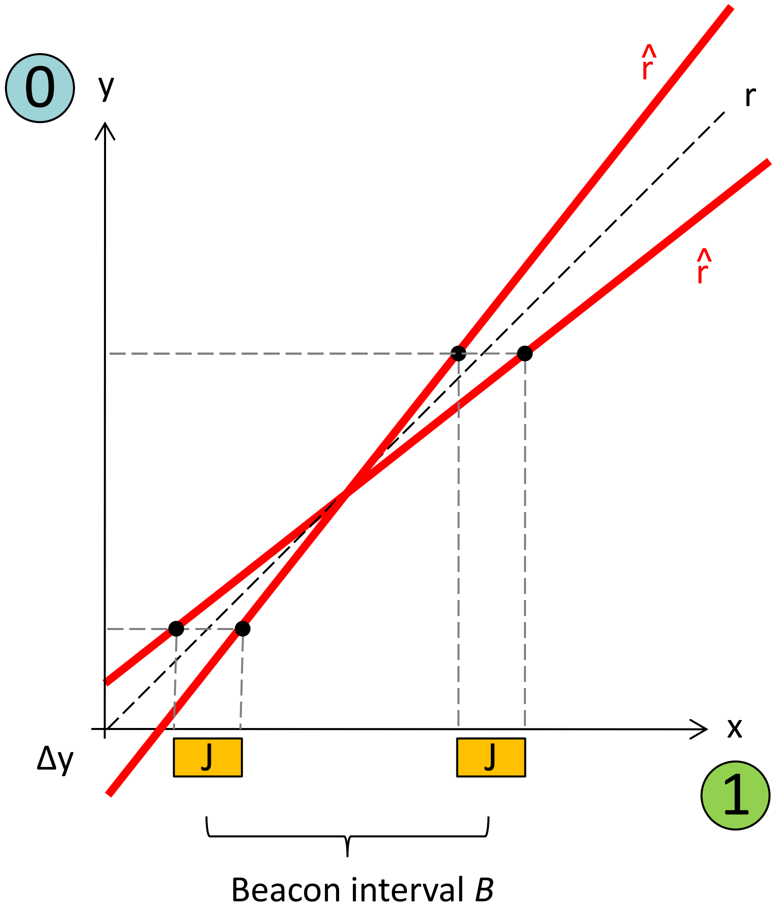
Back to Practice: Synchronizing Nodes

- Sending periodic beacon messages to synchronize nodes



How accurately can we synchronize two nodes?

- Message delay jitter affects clock synchronization quality



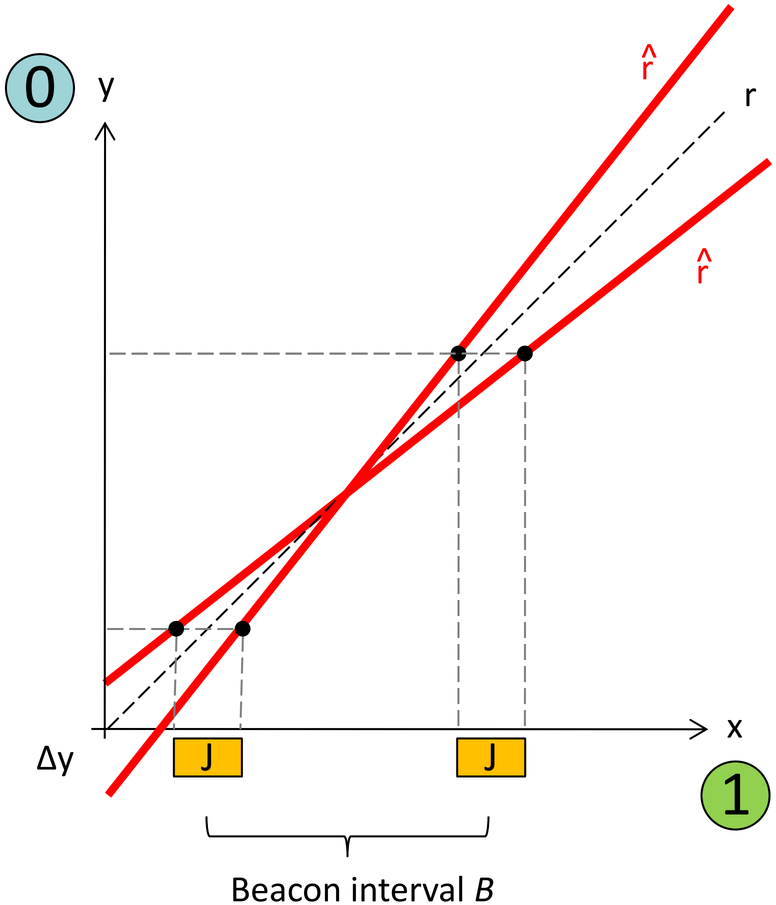
$$y(x) = \hat{r} \cdot x + \Delta y$$

relative clock rate (estimated)

clock offset

Clock Skew between two Nodes

- Lower Bound on the clock skew between two neighbors



Error in the rate estimation:
– Jitter in the message delay
– Beacon interval
– Number of beacons k

$$|\hat{r} - r| \sim \frac{J}{Bk\sqrt{k}}$$

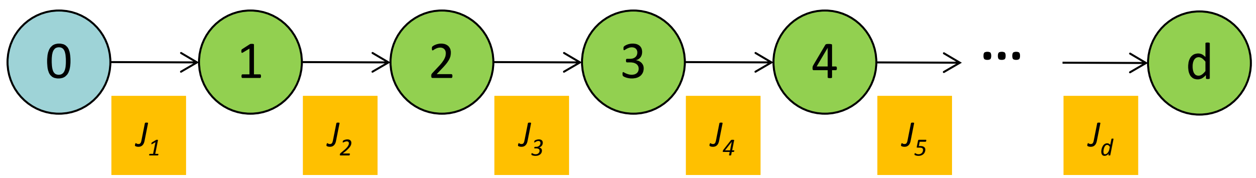
Synchronization error:

$$|\hat{y} - y| \sim \frac{J}{\sqrt{k}}$$

Multi-hop Clock Synchronization

- Nodes forward their current estimate of the reference clock

  Each synchronization beacon is affected by a **random jitter *J***



- Sum of the jitter grows with the square-root of the distance

  stddev($J_1 + J_2 + J_3 + J_4 + J_5 + \ldots J_d$) = $\sqrt{d}$×stddev(J)

Single-hop:

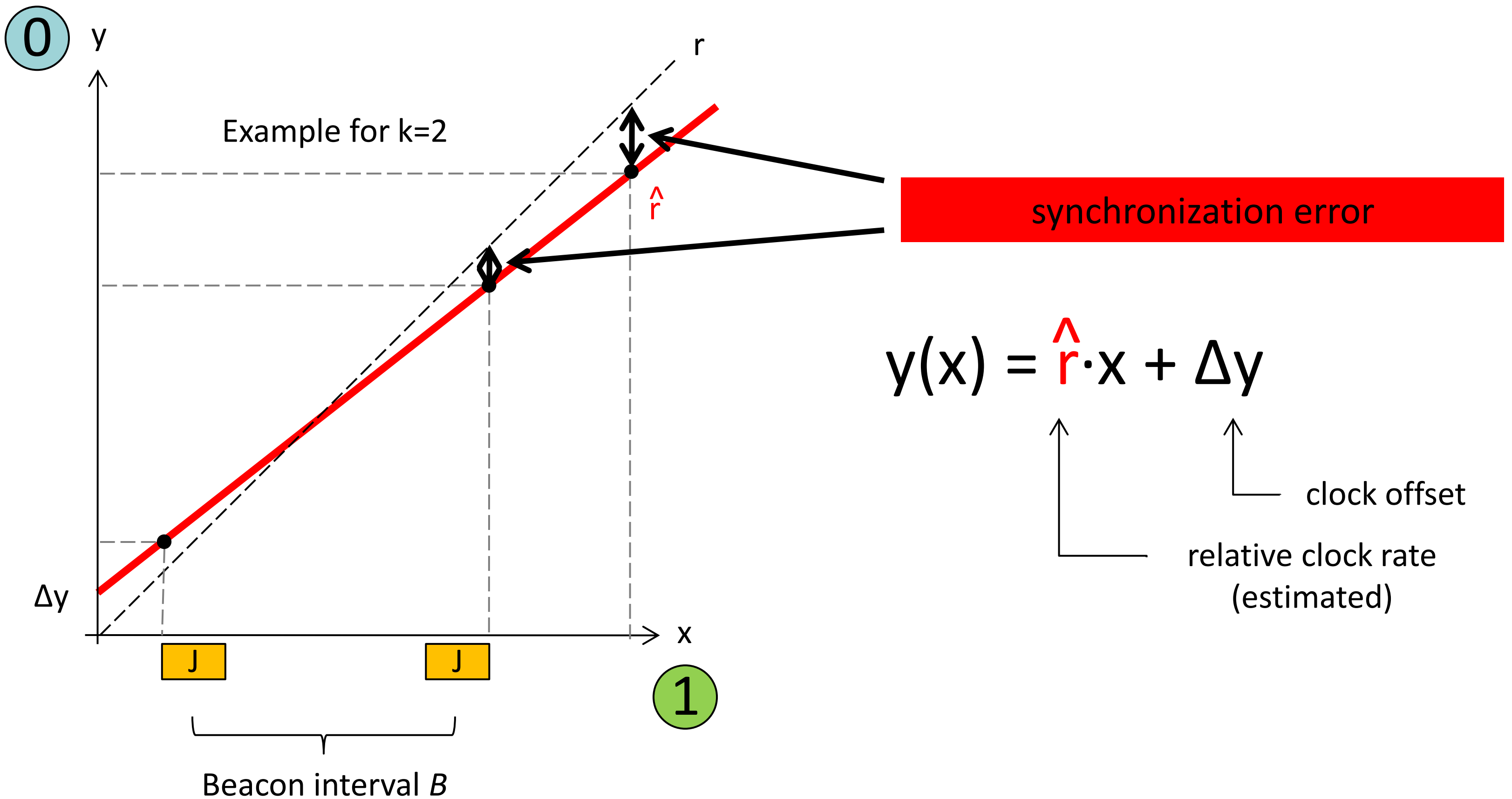
$$|\hat{y} - y| \sim \frac{J}{\sqrt{k}}$$

Multi-hop:

$$|\hat{y} - y| \sim \frac{J\sqrt{d}}{\sqrt{k}}$$

Linear Regression (e.g. FTSP)

- FTSP uses linear regression to compensate for clock drift
  Jitter is amplified before it is sent to the next hop



$$y(x) = \hat{r} \cdot x + \Delta y$$

clock offset

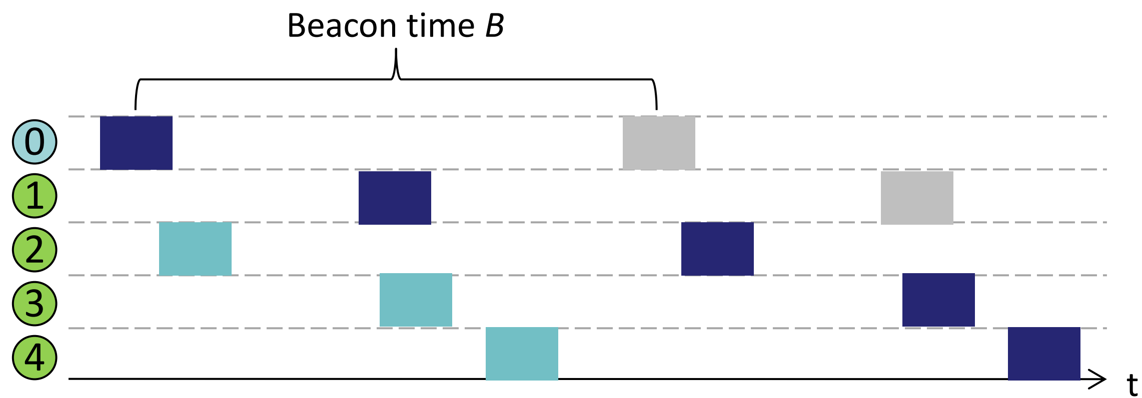
relative clock rate (estimated)

The PulseSync Protocol

- Send fast synchronization pulses through the network
  - Speed-up the initialization phase
  - Faster adaptation to changes in temperature or network topology

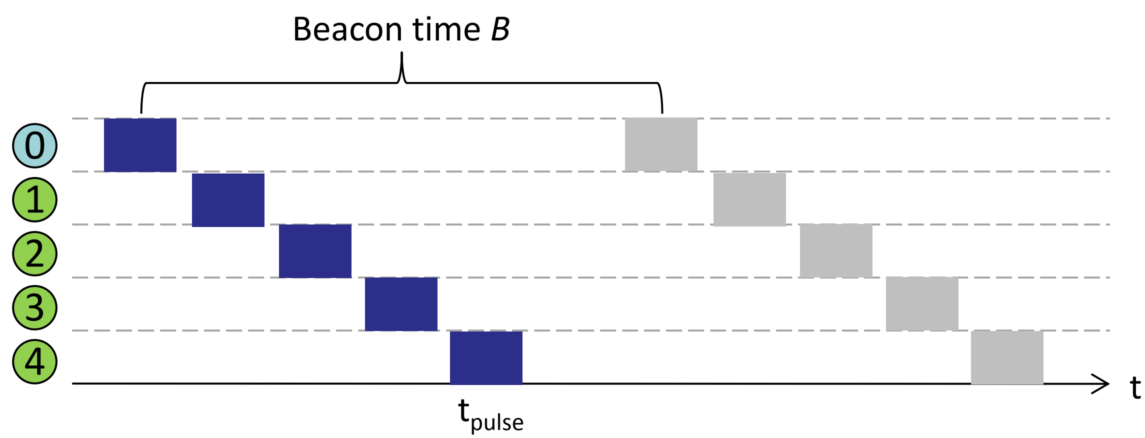
FTSP

Expected time
= D·B/2



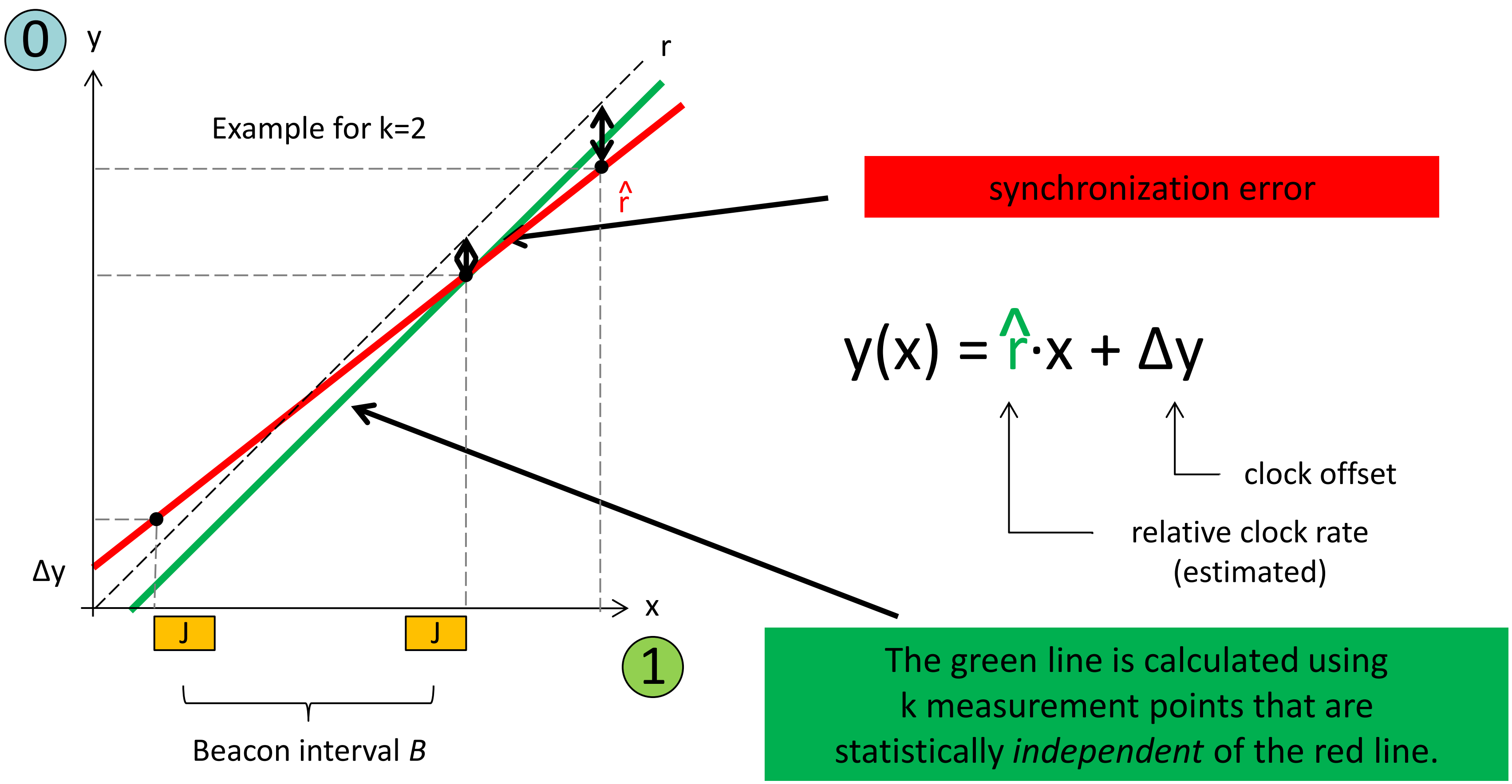
PulseSync

Expected time
= $D \cdot t_{pulse}$



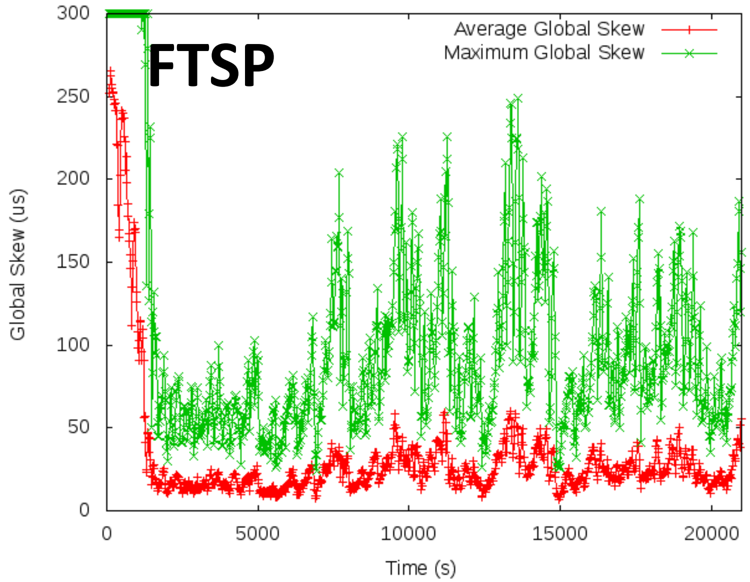
The PulseSync Protocol (2)

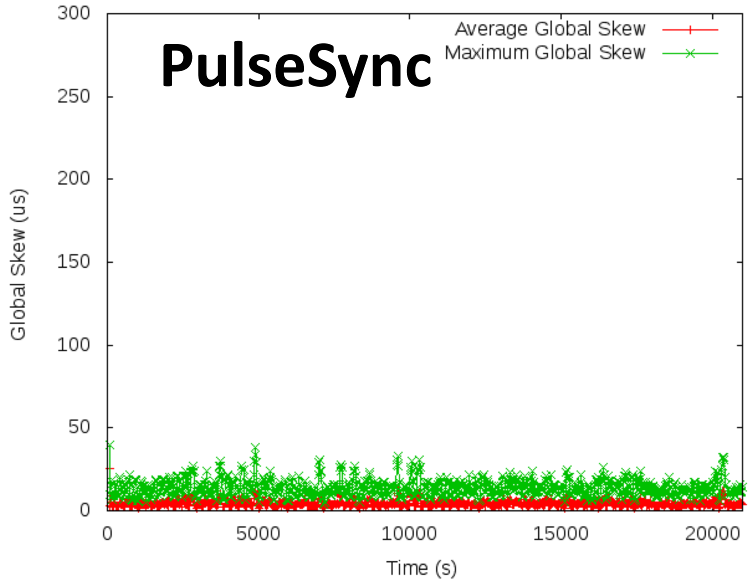
- Remove self-amplification of synchronization error
  - Fast flooding cannot completely eliminate amplification



FTSP vs. PulseSync

- Global Clock Skew
  - Maximum synchronization error between any two nodes

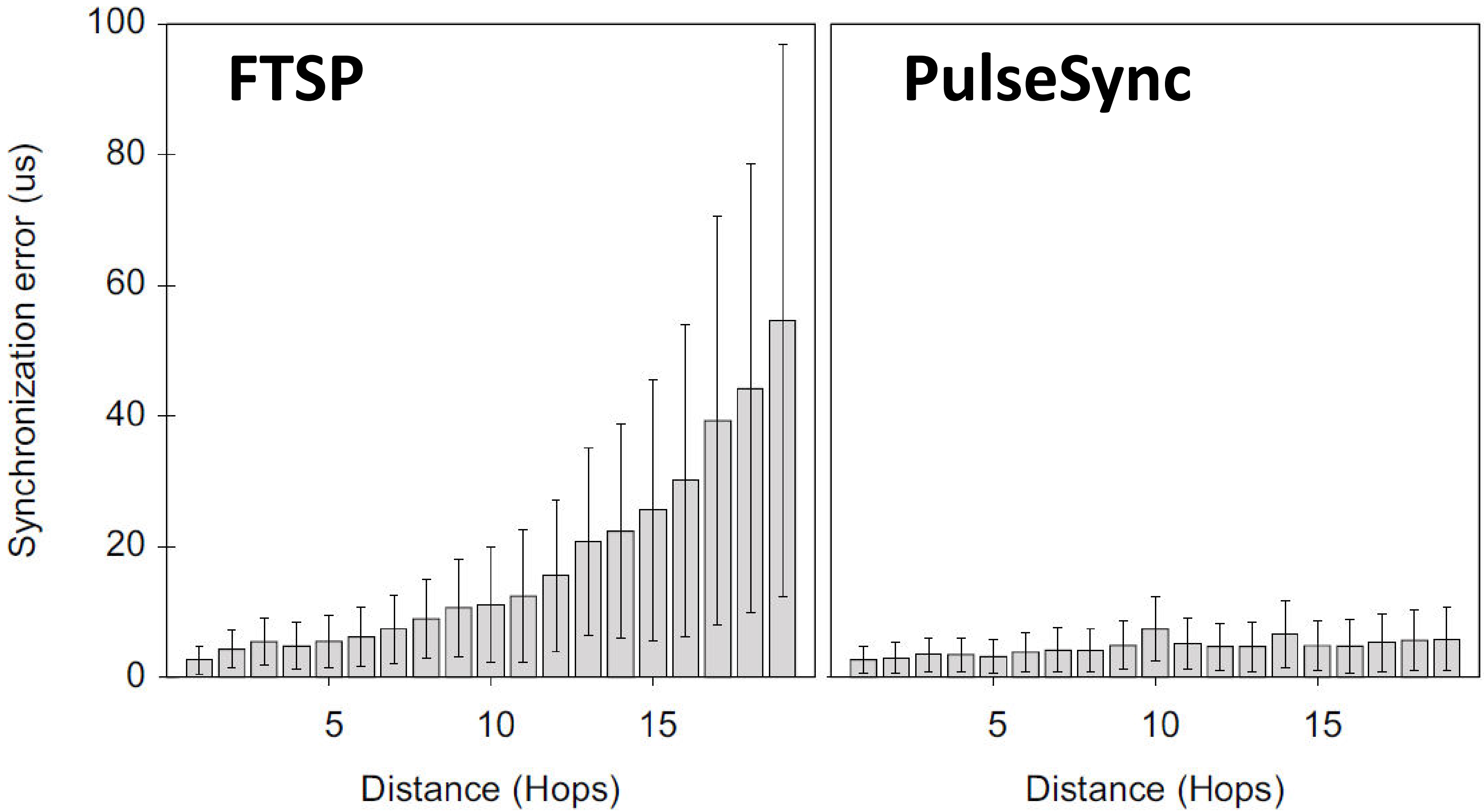




Synchronization Error	FTSP	PulseSync
Average (t>2000s)	23.96 µs	4.44 µs
Maximum (t>2000s)	249 µs	38 µs

FTSP vs. PulseSync

- Sychnronization Error vs. distance from root node



Credits

- The Network Time Protocol was originally designed by David L. Mills, 1985.
- The Precision Time Protocol standard was defined by an IEEE working group for precise networked clock synchronization under John Eidson, 2002.
- The Reference Broadcast Synchronization scheme was first introduced by Jeremy Elson, Lewis Girod and Deborah Estrin, 2002.
- The Flooding Time Synchronization Protocol is due to Miklos Maroti et al., 2004.
- TPSN is due Saurabh Ganeriwal et al., 2003.
- GTSP is due Philipp Sommer et al., 2009.
- Local skew results by Fan & Lynch, Lenzen, Locher, Kuhn, et al.
- Approximation algorithms for minimum max stretch spanning tree, e.g. Emek and Peleg, 2004.
- PulseSync was proposed by Lenzen et al., 2009.

That’s all!

Questions & Comments?

Roger Wattenhofer