

Clock Synchronization

Chapter 8



Rating

- Area maturity

First steps

Text book

- Practical importance

No apps

Mission critical

- Theory appeal

Booooooring

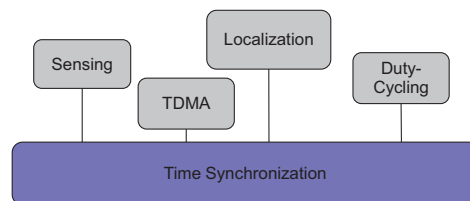
Exciting

Overview

- Motivation
- Clock Sources & Hardware
- Single-Hop Clock Synchronization
- Clock Synchronization in Networks
- Protocols: RBS, TPSN, FTSP, GTSP
- Theory of Clock Synchronization
- Protocol: PulseSync

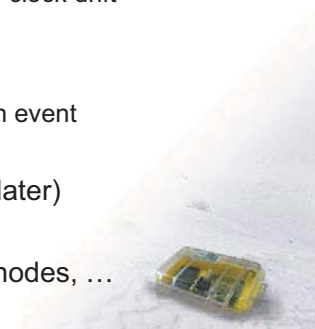
Motivation

- Synchronizing time is essential for **many applications**
 - Coordination of wake-up and sleeping times (energy efficiency)
 - TDMA schedules
 - Ordering of collected sensor data/events
 - Co-operation of multiple sensor nodes
 - Estimation of position information (e.g. shooter detection)
- Goals of clock synchronization
 - Compensate *offset* between clocks
 - Compensate *drift* between clocks



Properties of Clock Synchronization Algorithms

- External versus 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 versus continuous synchronization
 - Periodic synchronization required to compensate clock drift
- A-priori versus a-posteriori
 - A-posteriori clock synchronization triggered by an event
- Global versus local synchronization (explained later)
- Accuracy versus convergence time, Byzantine nodes, ...



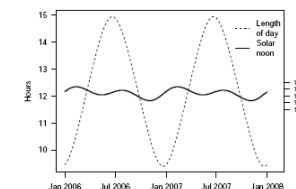
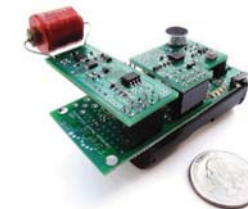
Clock Sources

- 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 distance to the sender, Frankfurt-Zurich is about **1ms**.
 - Special antenna/receiver hardware required
- 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



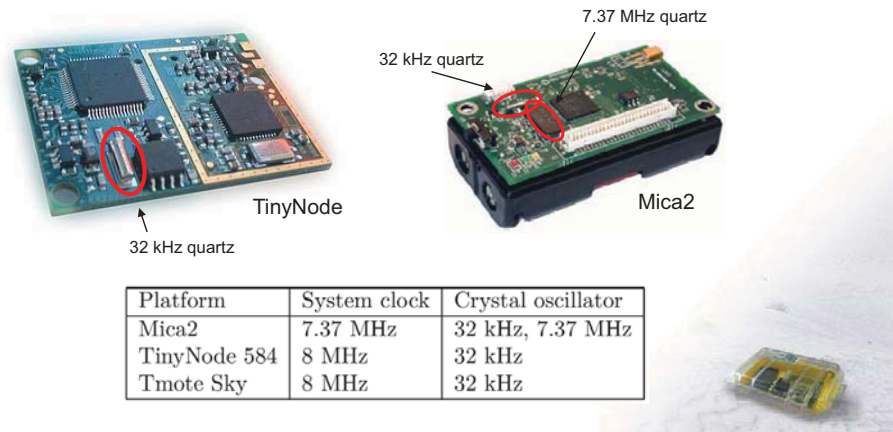
Clock Sources (2)

- AC power lines:
 - Use the magnetic field radiating from electric AC power lines
 - AC power line oscillations are extremely stable (10^{-8} ppm)
 - Power efficient, consumes only 58 μ W
 - 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 Sensor Nodes

- Structure
 - External oscillator with a nominal frequency (e.g. 32 kHz or 7.37 MHz)
 - Counter register which is incremented with oscillator pulses
 - Works also when CPU is in sleep state



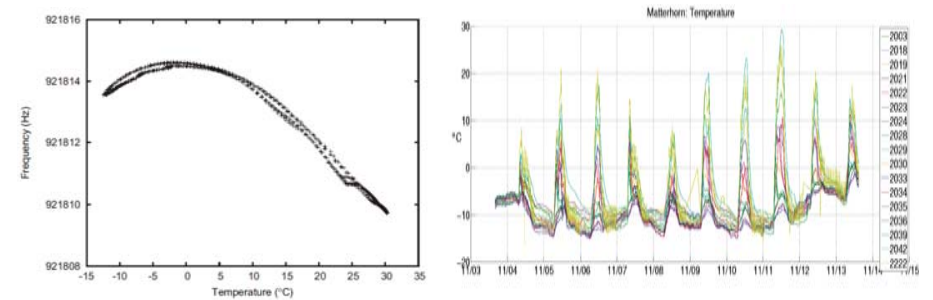
Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/9

Clock Drift

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

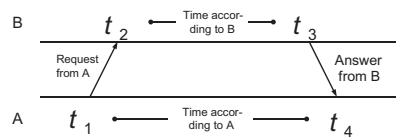


- E.g. TinyNodes have a maximum drift of 30-50 ppm at room temperature



Sender/Receiver Synchronization

- Round-Trip Time (RTT) based synchronization



- Receiver synchronizes to sender's clock
- Propagation delay δ and clock offset θ can be calculated

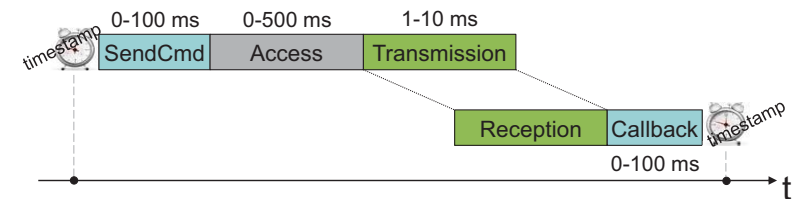
$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

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

Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/11

Messages Experience Jitter in the Delay

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

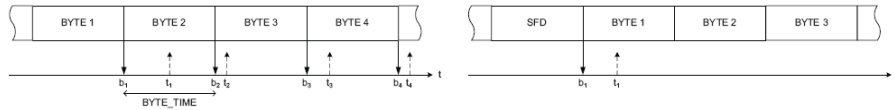


- Solution: Timestamping packets at the MAC layer [Maróti et al.]
 - Jitter in the message delay is reduced to a few clock ticks

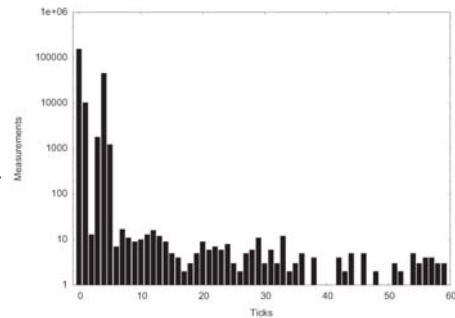
Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/12

Some Details

- 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.

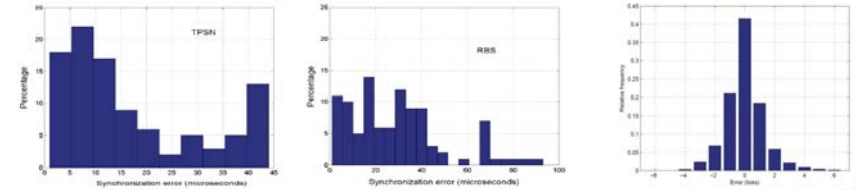


- In sensor networks propagation can be ignored ($<1\mu s$ for 300m).
- Still there is quite some variance in transmission delay because of latencies in **interrupt handling** (picture right).



Symmetric Errors

- Many protocols don't even handle single-hop clock synchronization well. On the left figures we see the absolute synchronization errors of TPSN and RBS, respectively. The figure on the right presents a single-hop synchronization protocol minimizing systematic errors.



- Even perfectly **symmetric** errors will sum up over multiple hops.
 - In a chain of n nodes with a standard deviation σ on each hop, the expected error between head and tail of the chain is in the order of $\sigma\sqrt{n}$.



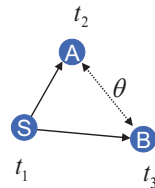
Reference-Broadcast Synchronization (RBS)

- A sender synchronizes a set of receivers with one another
- Point of reference: beacon's arrival time

$$t_2 = t_1 + S_S + A_S + P_{S,A} + R_A$$

$$t_3 = t_1 + S_S + A_S + P_{S,B} + R_B$$

$$\theta = t_2 - t_3 = (P_{S,A} - P_{S,B}) + (R_A - R_B)$$



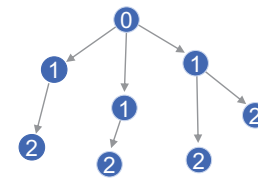
- Only sensitive to the **difference** in propagation and reception time
- Time stamping at the interrupt time when a beacon is received
- After a beacon is sent, all receivers exchange their reception times to calculate their clock offset

- Post-synchronization** possible
- E.g., least-square linear regression to tackle clock drifts
- Multi-hop?



Time-sync Protocol for Sensor Networks (TPSN)

- Traditional sender-receiver synchronization (RTT-based)
- Initialization phase: Breadth-first-search flooding**
 - Root node at level 0 sends out a *level discovery* packet
 - Receiving nodes which have not yet an assigned level set their **level** to +1 and start a random timer
 - After the timer is expired, a new level discovery packet will be sent
 - When a new node is deployed, it sends out a *level request* packet after a random timeout

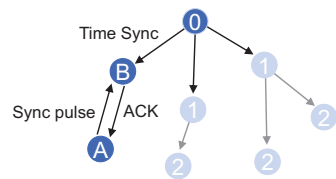


Why this random timer?



Time-sync Protocol for Sensor Networks (TPSN)

- *Synchronization phase*
 - Root node issues a *time sync* packet which triggers a random timer at all level 1 nodes
 - After the timer is expired, the node asks its parent for synchronization using a *synchronization pulse*
 - The parent node answers with an *acknowledgement*
 - Thus, the requesting node knows the round trip time and can calculate its clock offset
 - Child nodes receiving a synchronization pulse also start a random timer themselves to trigger their own synchronization



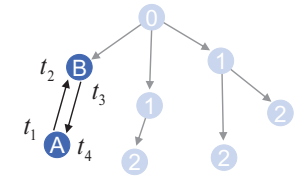
Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/17

Time-sync Protocol for Sensor Networks (TPSN)

$$t_2 = t_1 + S_A + A_A + P_{A,B} + R_B$$

$$t_4 = t_3 + S_B + A_B + P_{B,A} + R_A$$

$$\theta = \frac{(S_A - S_B) + (A_A - A_B) + (P_{A,B} - P_{B,A}) + (R_B - R_A)}{2}$$



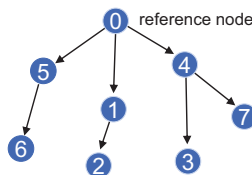
- Time stamping packets at the MAC layer
- In contrast to RBS, the signal propagation time might be negligible
- Authors claim that it is “about two times” better than RBS
- Again, clock drifts are taken into account using periodical synchronization messages
- Problem: What happens in a non-tree topology (e.g. **grid**)?
 - Two neighbors may have bad synchronization?



Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/18

Flooding Time Synchronization Protocol (FTSP)

- Each node maintains both a local and a global time
- Global time is synchronized to the local time of a reference node
 - Node with the smallest id is elected as the reference node
- Reference time is flooded through the network periodically



- Timestamping at the MAC Layer is used to compensate for deterministic message delays
- Compensation for clock drift between synchronization messages using a linear regression table

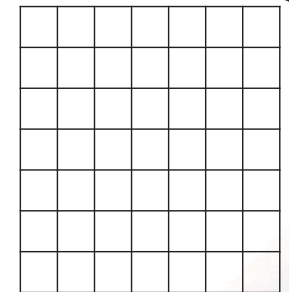
Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/19

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.



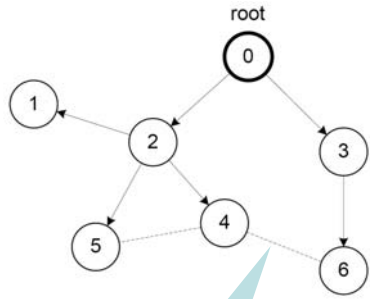
- Example: Grid network, with $n = m^2$ nodes.
- No matter what tree you use, the maximum stretch of the spanning tree will always be at least m (just try on the grid figure right...)
- In general, finding the **minimum max stretch spanning tree** is a hard problem, however approximation algorithms exist [Emek, Peleg, 2004].



Ad Hoc and Sensor Networks – Roger Wattenhofer – 9/20

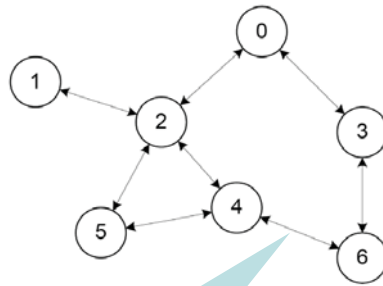
Variants of Clock Synchronization Algorithms

Tree-like Algorithms
e.g. FTSP



Bad local skew

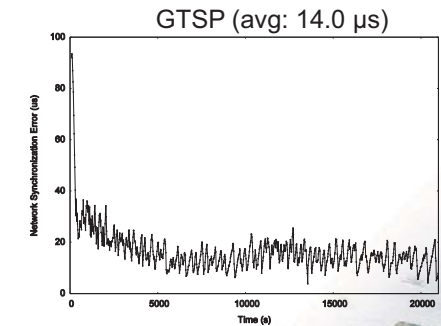
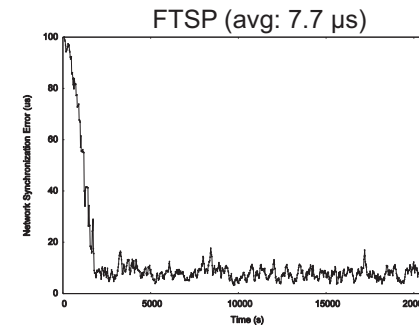
Distributed Algorithms
e.g. GTSP



All nodes consistently average errors to all neighbors

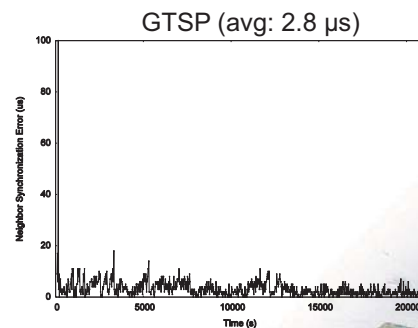
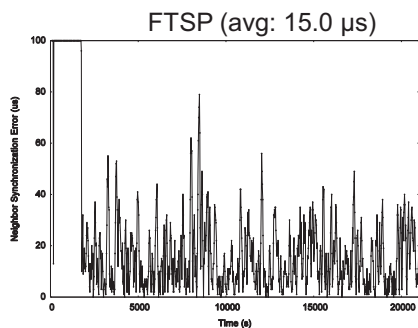
FTSP vs. GTSP: Global Skew

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



FTSP vs. GTSP: Local Skew

- Neighbor Synchronization error (**local skew**)
 - Pair-wise synchronization error between **neighboring nodes**
- Synchronization error between two direct neighbors:



Global vs. Local Time Synchronization

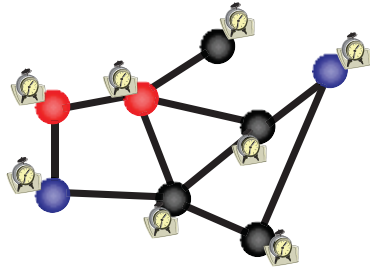
- Common time is essential for many applications:
 - Global** – Assigning a timestamp to a globally sensed event (e.g. earthquake)
 - Local** – Precise event localization (e.g. shooter detection, multiplayer games)
 - Local** – TDMA-based MAC layer in wireless networks
 - Local** – Coordination of wake-up and sleeping times (energy efficiency)



Theory of Clock Synchronization

- Given a communication network
 - Each node equipped with hardware clock with **drift**
 - 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!**

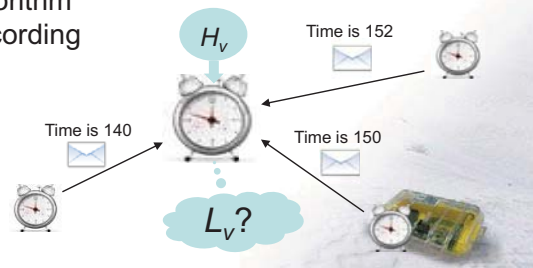
Formal Model

- Hardware clock $H_v(t) = \int_{[0,t]} h_v(\tau) d\tau$ Clock drift ϵ is typically small, e.g. $\epsilon \approx 10^{-4}$ for a cheap quartz oscillator with clock rate $h_v(t) \in [1-\epsilon, 1+\epsilon]$

- 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”)

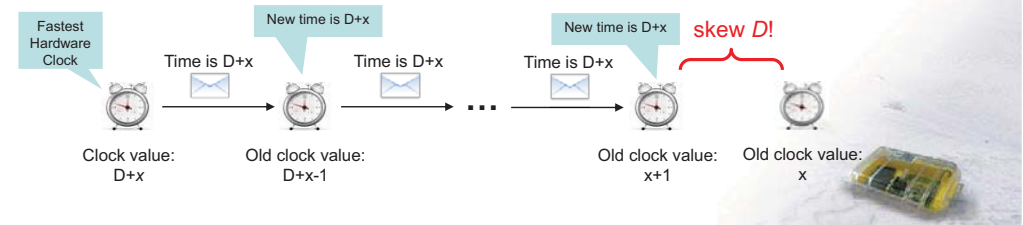
- Message delays $\in [0, 1]$ Neglect fixed share of delay, normalize jitter

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



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

- Question: How to update the logical clock based on the messages from the neighbors? Allow $\beta = \infty$
- 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

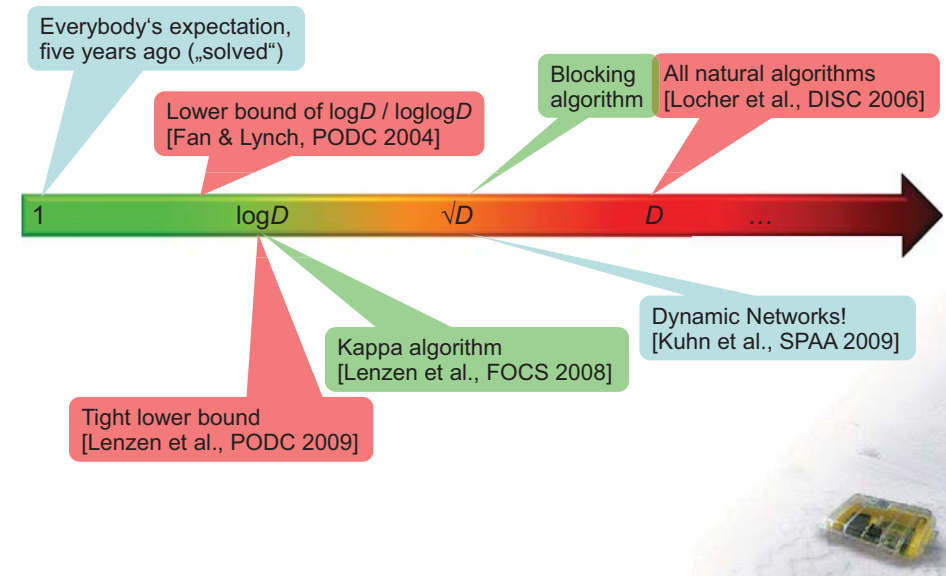
$$L_i(t) := \max(L_i(t), \max_{j \in N_i} L_j(t) - \gamma)$$

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

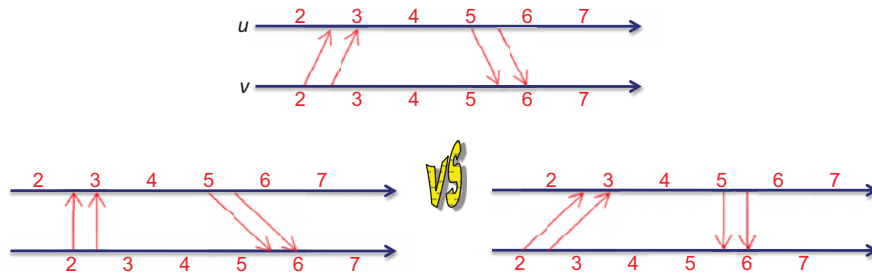
- How can we do better?
 - Adjust logical clock speeds to catch up with fastest node (i.e. **no jump**)?
 - Idea: Take the clock of all neighbors into account by choosing the **average** value?



Local Skew: Overview of Results

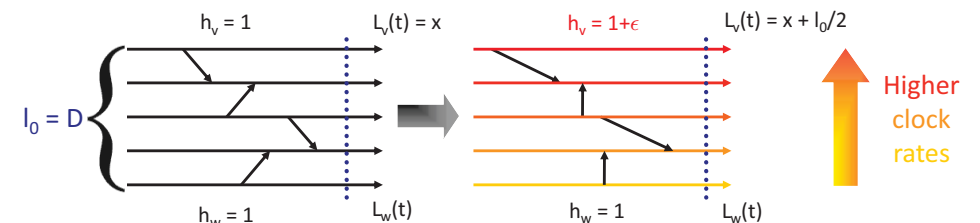


Enforcing Clock Skew



- 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 ($+1/2, -1/4, +1/2, -1/4, \dots$) $\log_{2(\beta-1)/\epsilon} D$ 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	$1+\epsilon$
local skew	∞

We can have both smooth and accurate clocks!

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

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	∞	$\theta(\log D)$	$\theta(\log_{1/\epsilon} D)$	$\theta(\log_{1/\epsilon} D)$	$\theta(\log_{1/\epsilon} D)$

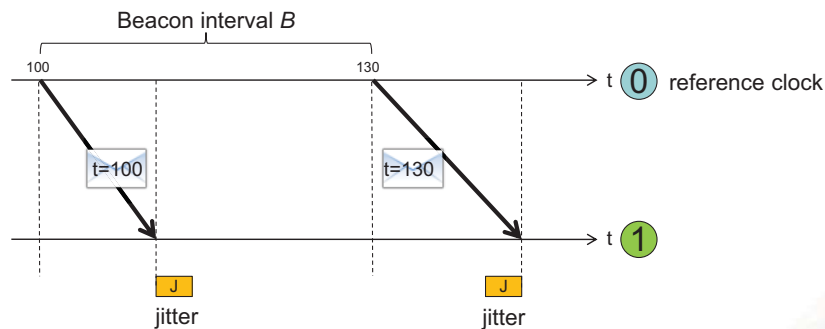
We can have both smooth and accurate clocks!

... 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! 😊

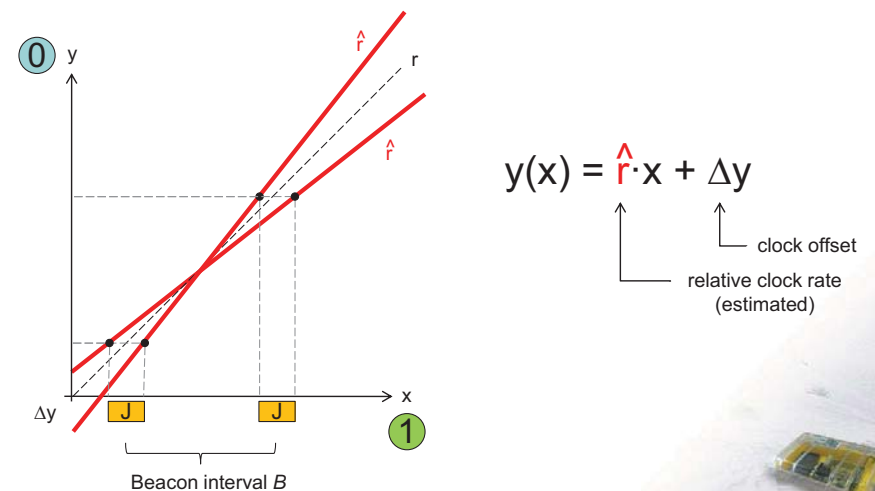
Back to Practice: Synchronizing Nodes

- Sending periodic beacon messages to synchronize nodes



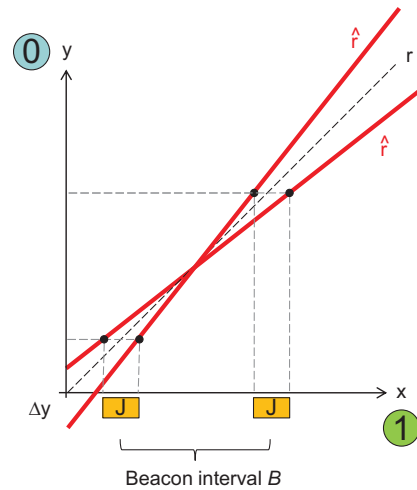
How accurately can we synchronize two Nodes?

- Message delay jitter affects clock synchronization quality



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

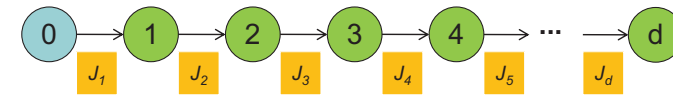
$$|\hat{r} - r| \sim \frac{J}{Bk\sqrt{k}}$$

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
 $\text{stddev}(J_1 + J_2 + J_3 + J_4 + J_5 + \dots + J_d) = \sqrt{d} \cdot \text{stddev}(J)$

Single-hop:

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

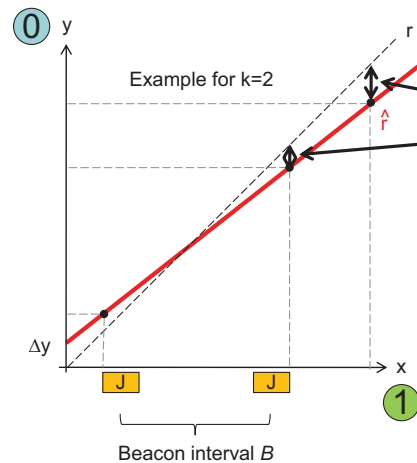


Multi-hop:

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

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



synchronization error

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

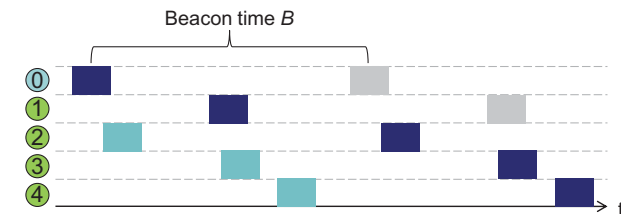
clock offset
 relative clock rate (estimated)

The PulseSync Protocol

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

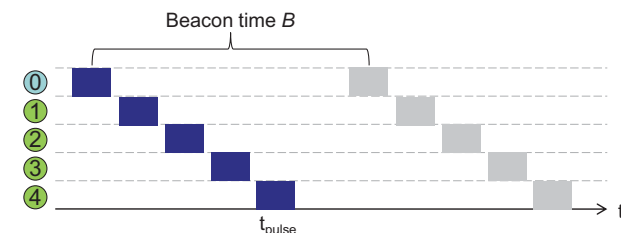
FTSP

Expected time
 $= D \cdot B / 2$



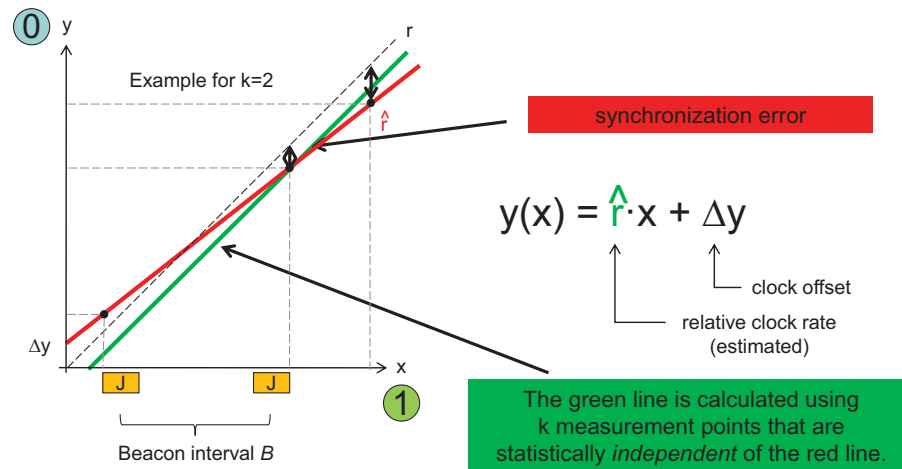
PulseSync

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



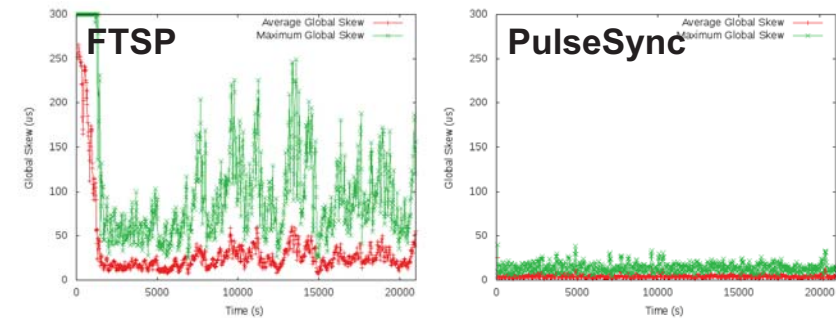
The PulseSync Protocol (2)

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



FTSP vs. PulseSync

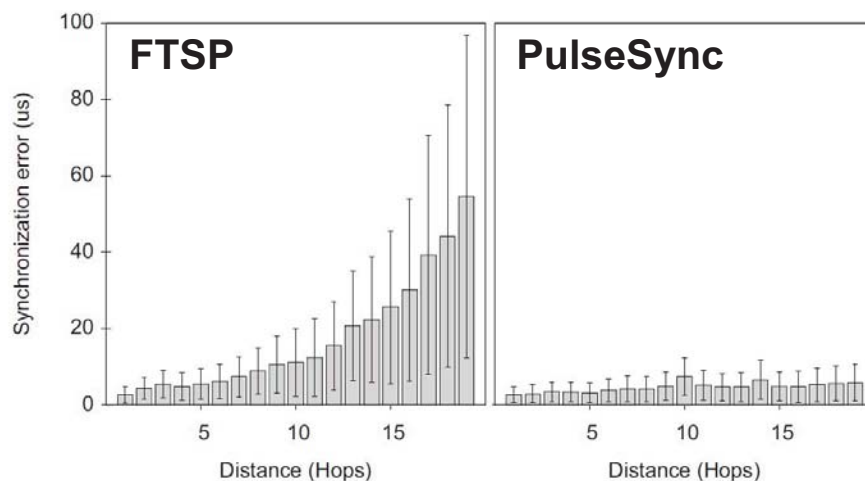
- Global Clock Skew
 - Maximum synchronization error between any two nodes



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

FTSP vs. PulseSync

- Synchronization Error vs. distance from root node



Open Problem

- As listed on slide 9/6, clock synchronization has lots of parameters. Some of them (like local/gradient) clock synchronization have only started to be understood.
- Local clock synchronization in combination with other parameters are not understood well, e.g.
 - accuracy vs. convergence
 - fault-tolerance in case some clocks are misbehaving [Byzantine]
 - clock synchronization in dynamic networks

