Chapter 5
Clock Synchronization

Distributed Systems

SS 2015

Fabian Kuhn

Properties of Clock Synch. 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, …

External Clock Sources

UTC: Coordinated Universal Time

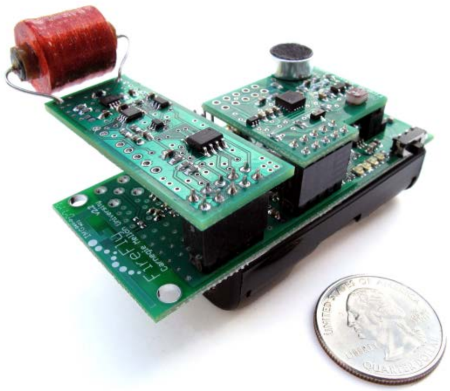
- based on about 200 atomic clocks in about 50 national labs
  - TAI: International Atomic Time
  - UTC = TAI + leap seconds
- transmitted over radio signal from DCF77 (near Frankfurt)

GPS: Global Positioning System

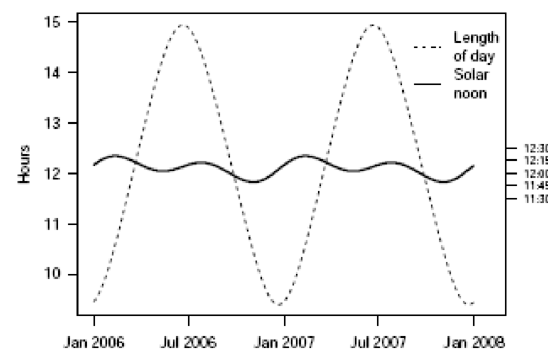
- satellites regularly broadcast their position and time
- satellites are based on USNO time
- signals from satellites allow to exactly position a receiver in space and time
  - and to correct skew due to propagation delay

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

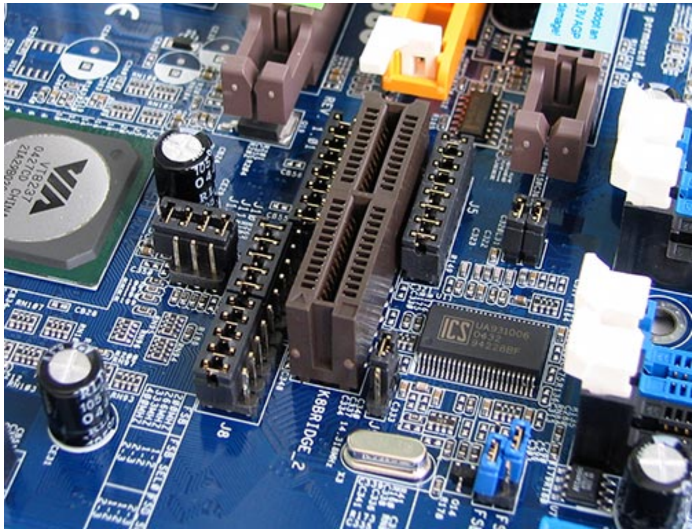


- Sunlight
  - Using a light sensor to measure the length of a day
  - Offline algorithm for reconstructing global timestamps by correlating annual solar patterns (no communication required)



Clock Devices in Computers

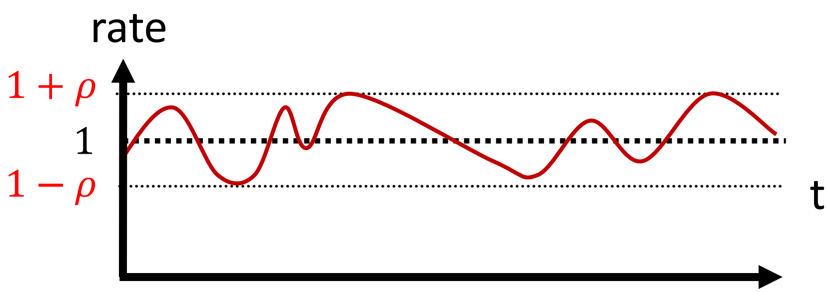
- Real Time Clock (IBM PC)
  - Battery backed up
  - 32.768 kHz oscillator + Counter
  - Get value via interrupt system



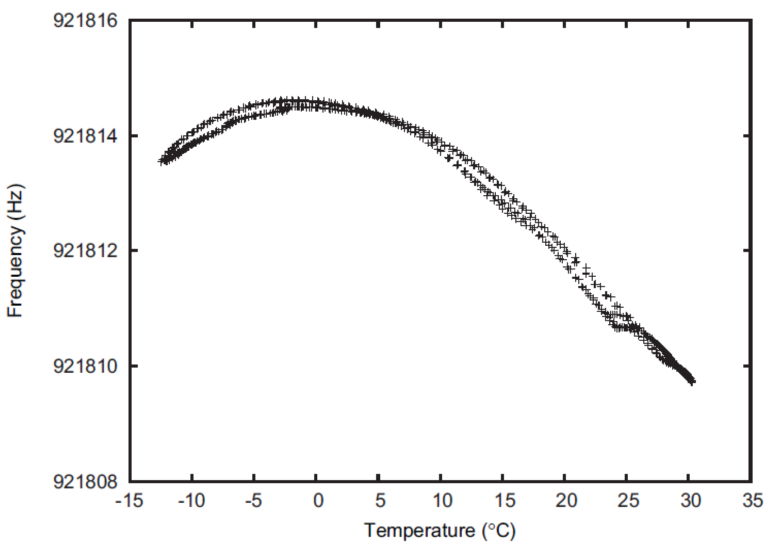
- HPET (High Precision Event Timer)
  - Oscillator: 10 Mhz … 100 Mhz
  - Up to 10 ns resolution!
  - Schedule threads
  - Smooth media playback
  - Usually inside Southbridge

Clock Drift

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



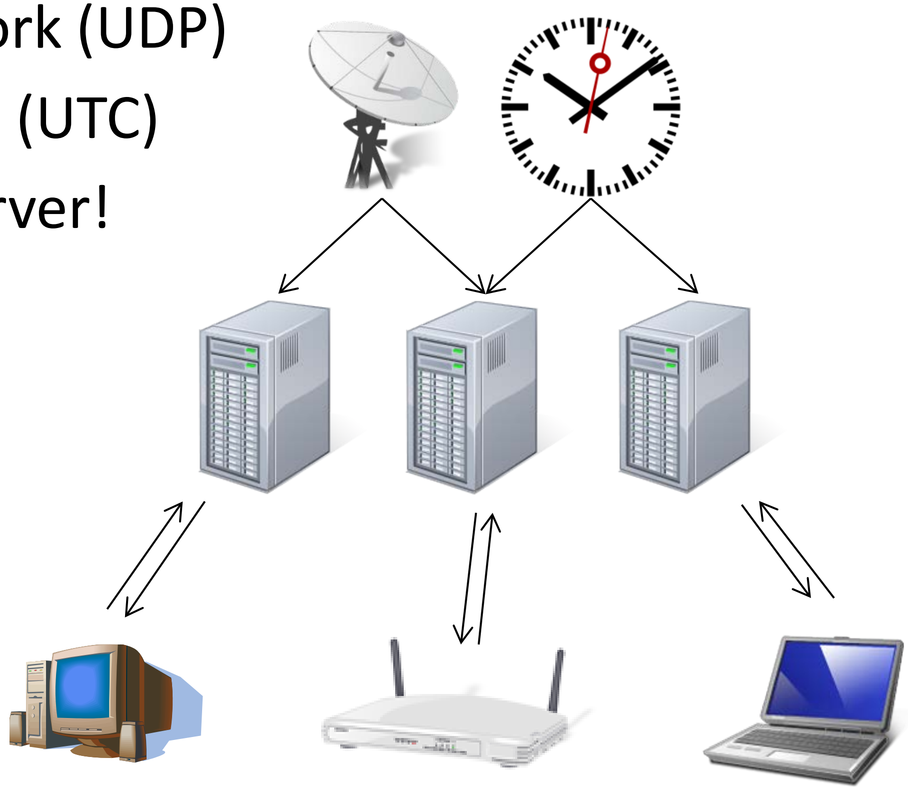
- E.g., TinyNodes have a max. drift of 30-50 ppm (parts per million)



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

Clock Synch. in Computer Networks

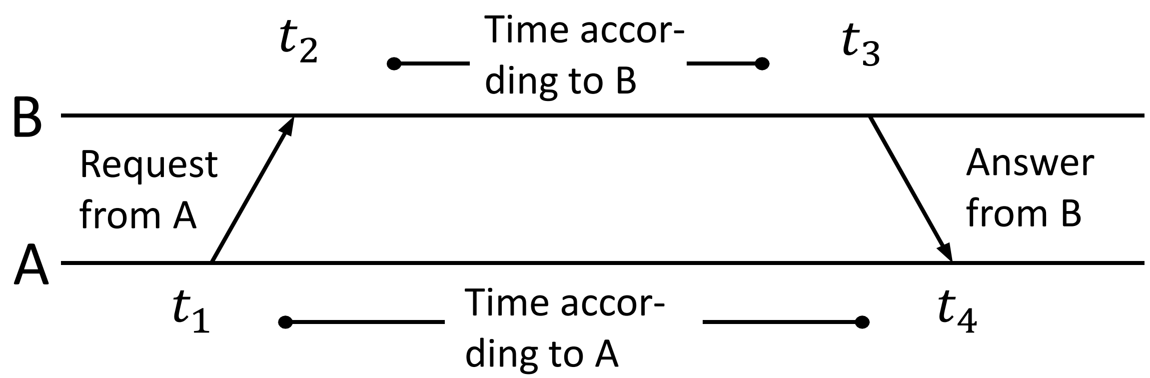
- Network Time Protocol (NTP)
- Clock sync via Internet/Network (UDP)
- Publicly available NTP Servers (UTC)
- You can also run your own server!



- 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

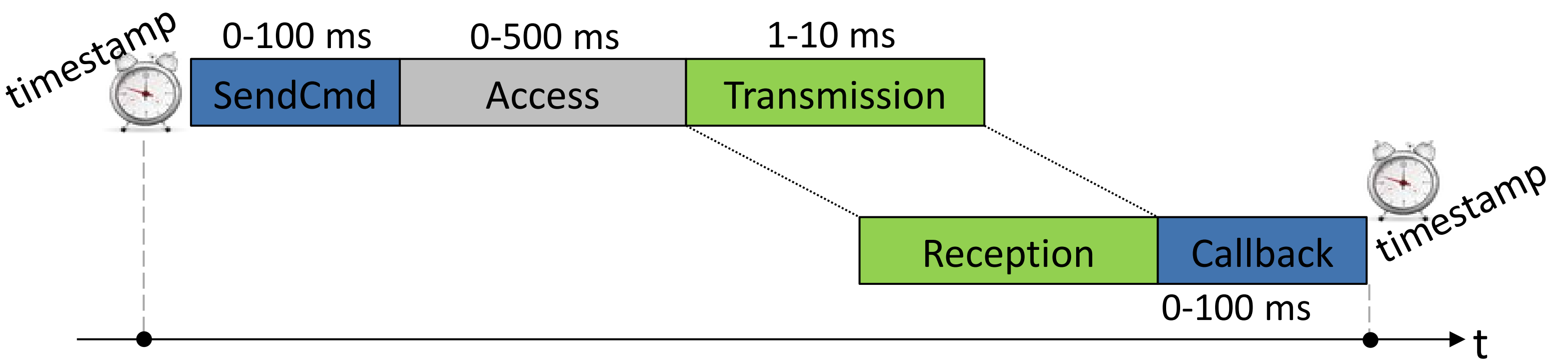
$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

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

Messages Experience Jitter in the Delay

- Problem: Jitter in the message delay

  Various sources of errors (deterministic and non-deterministic)

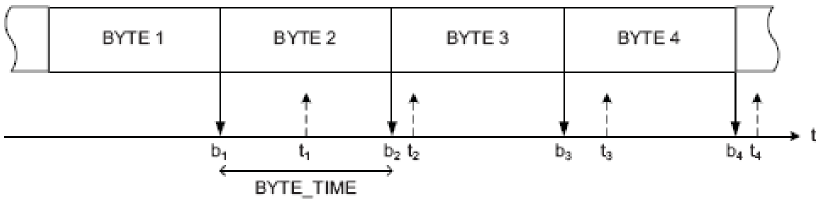


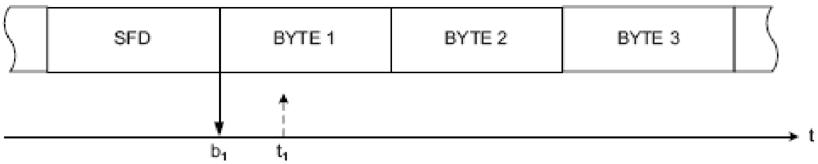
- Solution: Timestamping packets at the MAC layer

  → Jitter in the message delay is reduced to a few clock ticks

Messages Experience Jitter in the Delay

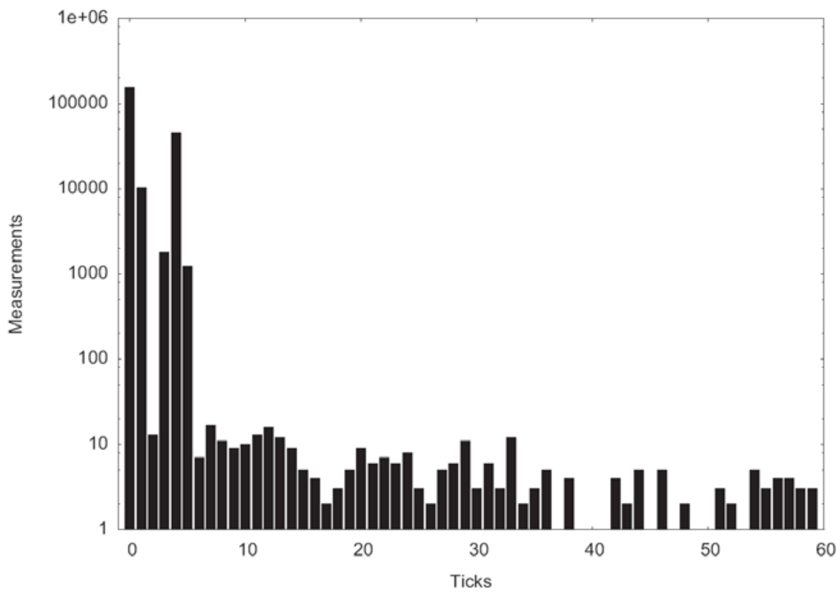
- Different radio chips use different paradigms
  - Left is a CC1000 radio chip which generates an interrupt with each byte.
  - Right is a CC2420 radio chip that generates a single interrupt for the packet after the start frame delimiter is received.





- In wireless networks propagation can be ignored ($< 1\mu s$ for 300m).

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

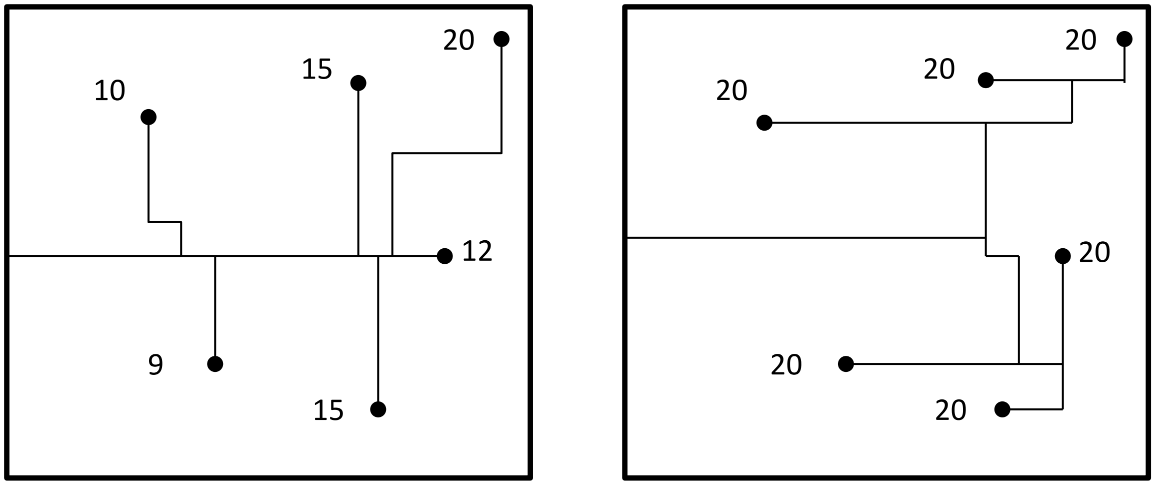


Clock Synch. in Computer Networks (PTP)

- 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

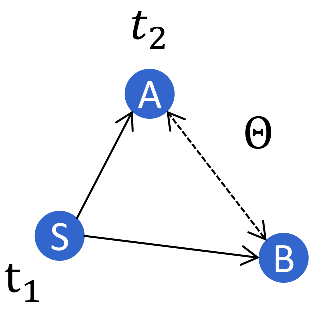
- Synchronous digital circuits require all components to act in sync



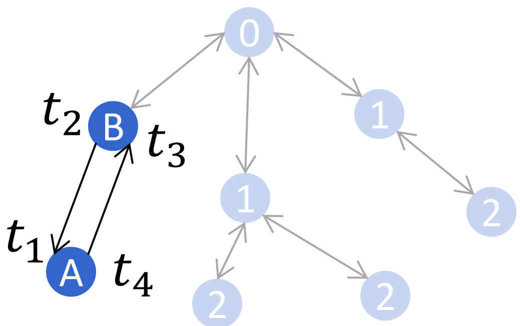
- The bigger the clock skew, the longer the clock period
- The clock signal that governs this rhythm needs to be distributed to all components such that skew and wire length is minimized
- Optimize routing, insert buffers (also to improve signal)

Clock Synch. Tricks in Wireless Networks

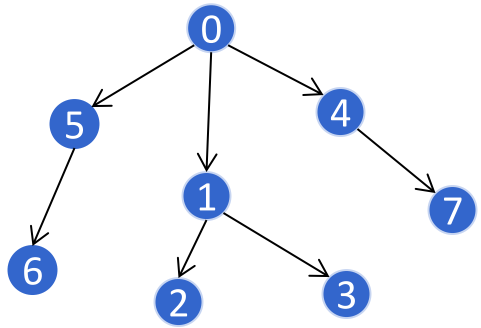
- Reference Broadcast Synchronization (RBS)
  ⟺ Synchronizing atomic clocks
  - Sender synchronizes a set of clocks



- 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



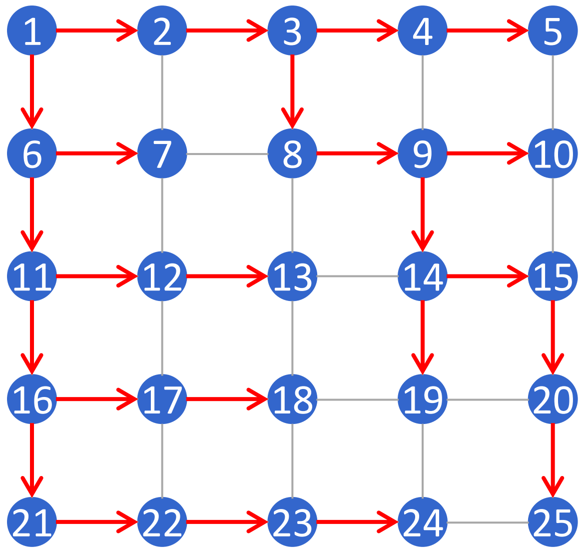
Best Tree for Tree-Based Clock Synch.?

- 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 max. stretch of the spanning tree will always be $\geq m$ (just try on the grid).

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

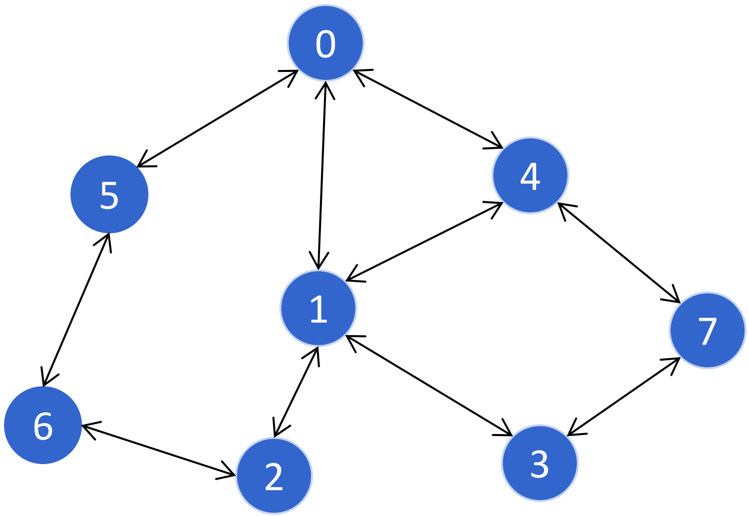


Clock Synchronization Tricks (GTSP)

GTSP = Gradient Time Synchronization Protocol

- Synchronize with *all* neighboring nodes
  - Broadcast periodic time beacons, e.g., every 30 s
  - No reference node necessary

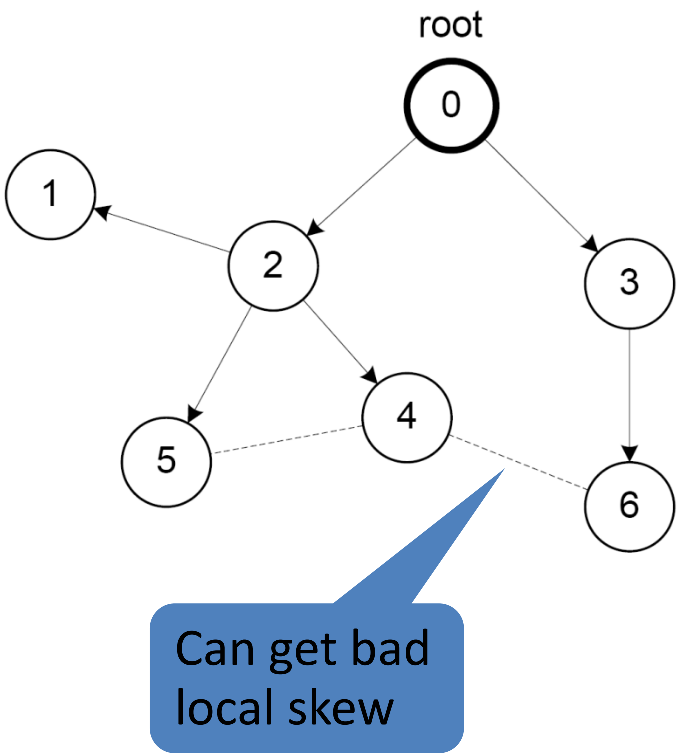
- How to synchronize clocks without having a leader?
  - Follow the node with the fastest/slowest clock?
  - **Idea:** Go to the average clock value/rate of all neighbors (including node itself)
  - Try to adapt to clock rate differences



Variants of Clock Synchronization Algorithms

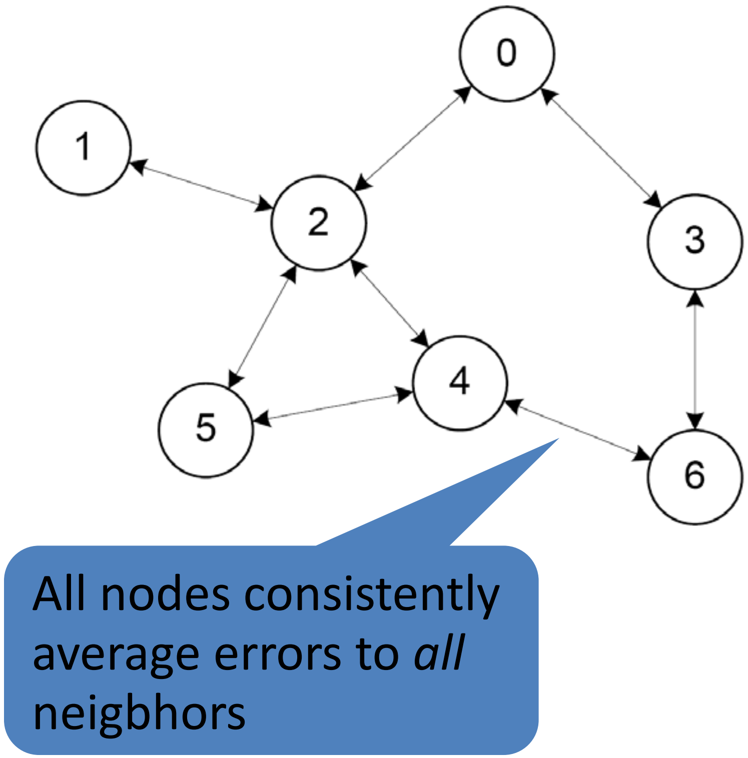
Tree-like Algorithms

e.g. FTSP



Distributed Algorithms

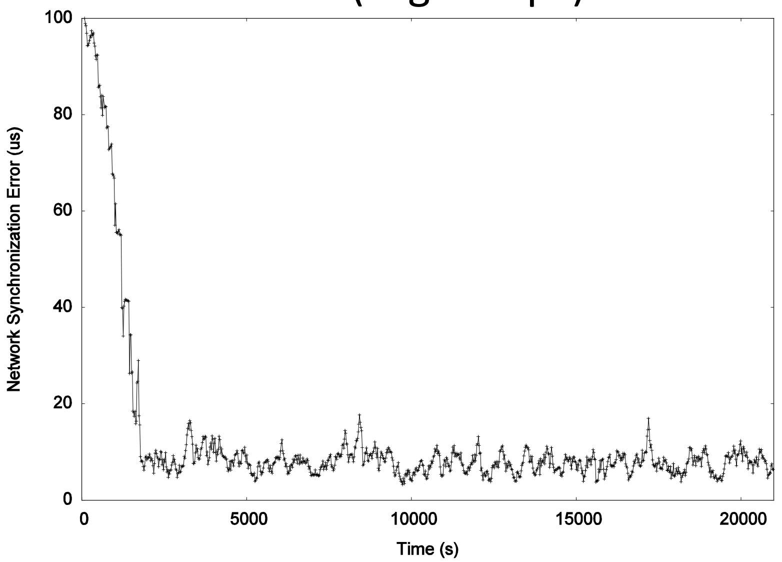
e.g. GTSP



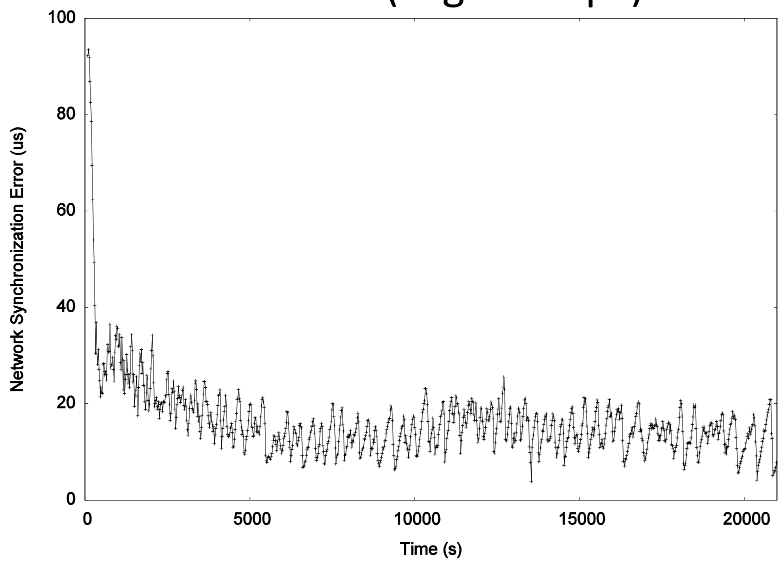
FTSP vs. GTSP: Global Skew

- 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)

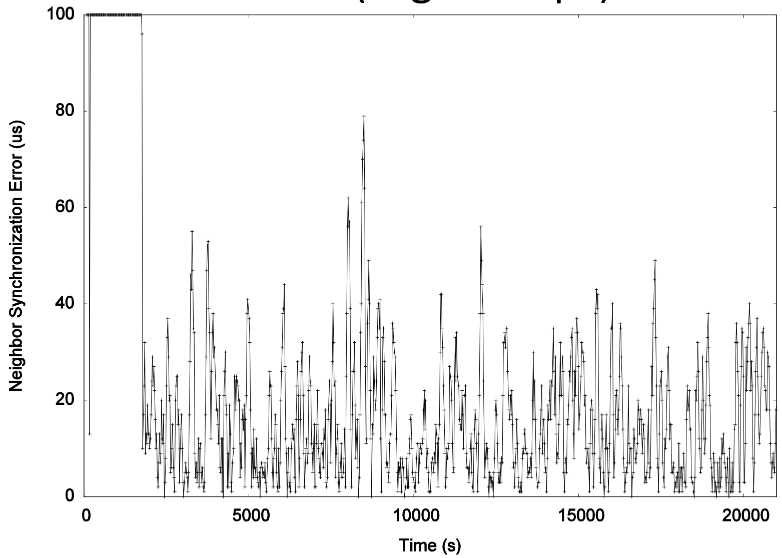


FTSP vs. GTSP: Local Skew

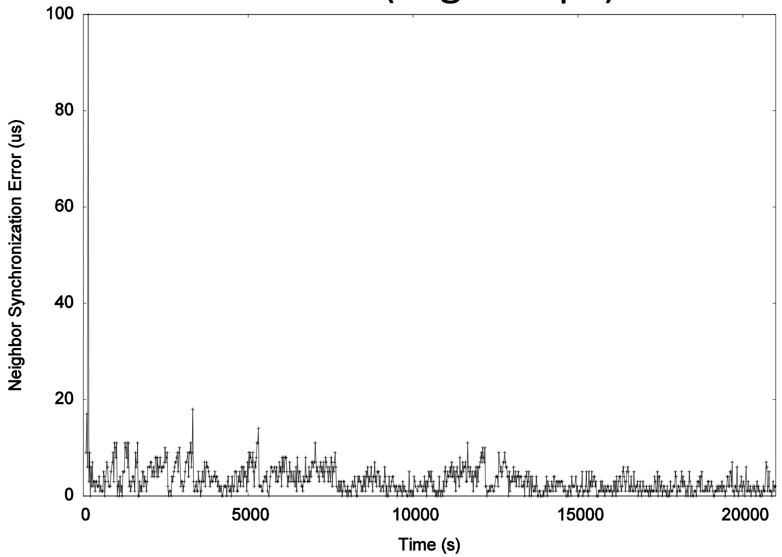
- Neighbor Synchronization error (local skew)
  - Pair-wise synchronization error between neighboring nodes

- Synchronization error between two direct neighbors:

FTSP (avg: 15.0 µs)



GTSP (avg: 2.8 µs)



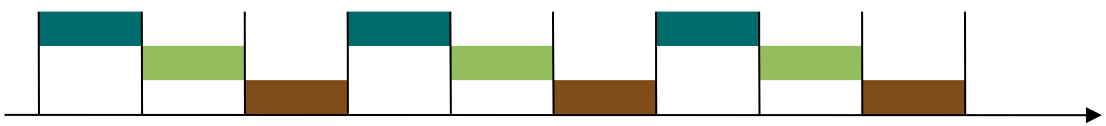
Global vs. Local Time Synchronization

- Common time is essential for many applications:

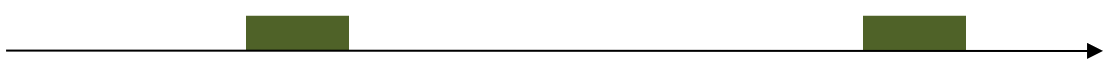
Global
  - Assigning a timestamp to a globally sensed event (e.g., earthquake)

Local
  - Precise event localization (e.g., sensors networks, multiplayer games)

Local
  - TDMA-based MAC layer in wireless networks



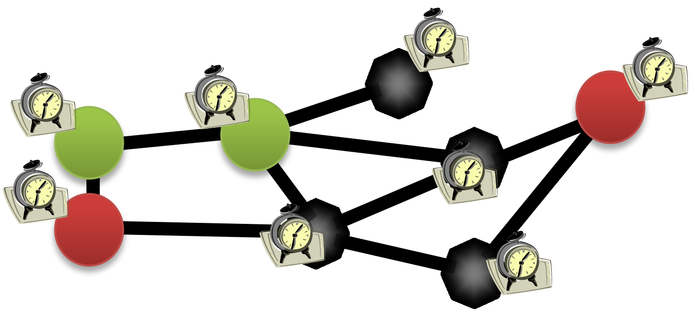
Local
  - Coordination of wake-up and sleeping times (energy efficiency)



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 $h_v(t) \in [1-\rho, 1+\rho]$

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

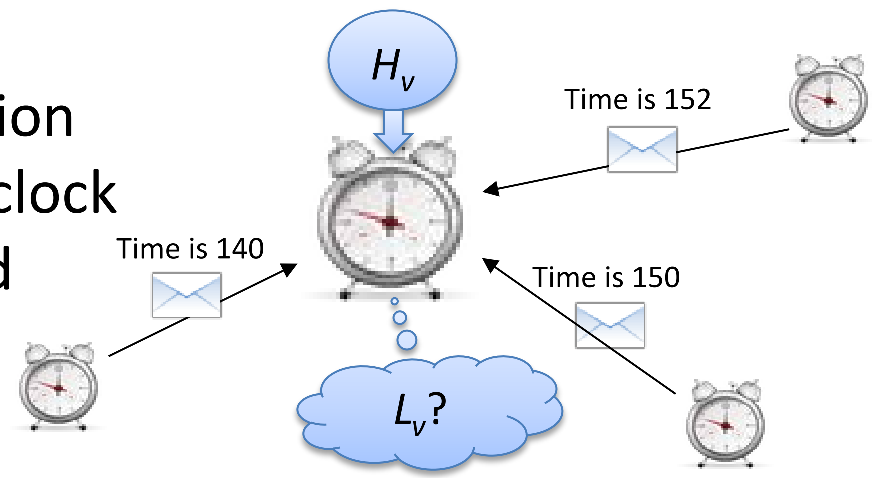
- Logical clock $L_v(t)$ which increases at rate at least $1-\rho$ and at most β

Logical clocks should run at least as fast as hardware clocks

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

Neglect fixed part of delay, normalize jitter to 1

- Goal: a distributed synchronization algorithm to update the logical clock according to hardware clock and messages from neighbors



Synchronization Algorithm $\mathcal{A}^{\max}$

Task: How to update logical clocks based on msg. from neighbors

Idea: Minimize skew to the fastest neighbor

Algorithm $\mathcal{A}^{\max}$

- Set logical clock to the maximum clock value received from any neighbor (if larger than local logical clock value)
- If recv. value $>$ previously forwarded value, forward immediately
- at least forward local logical clock value once every T time steps
  - send out local logical clock value if hardware clock proceeds by $1 - \rho$ since the last time the clock value was sent

Remark: Algorithm allows $\beta = \infty$

(clock values can jump to larger values)

Synchronization Algorithm $\mathcal{A}^{\max}$

Theorem: Alg. $\mathcal{A}^{\max}$ guarantees a global clock skew of at most

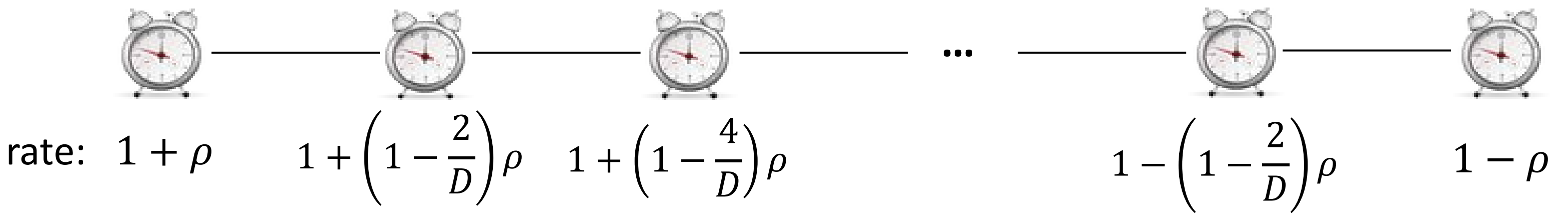
$$(1+\rho)\cdot D + 2\rho \cdot T.$$

(global clock skew = max. diff. between two clock values, D: diameter)

Synchronization Algorithm $\mathcal{A}^{\max}$

Global Skew can be D

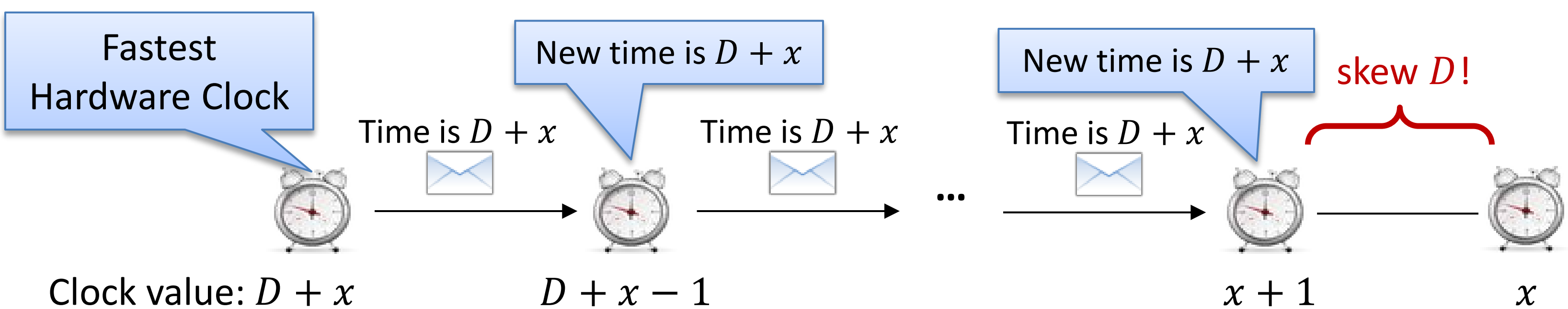
- path of length D, all message delays are 1



- skew between any 2 neighbors grows to 1 before detecting any skew

Local Skew can also be D...

- first all messages have delay 1 $\implies$ skew D between ends of path
- then, messages become very fast (delay ≈ 0)



Synchronization Algorithm $\mathcal{A}^{\max}$

Problems

- Global and local skew can both be $\Theta(D)$
- Clock values can jump (i.e., $\beta = \infty$)

Can we do better?

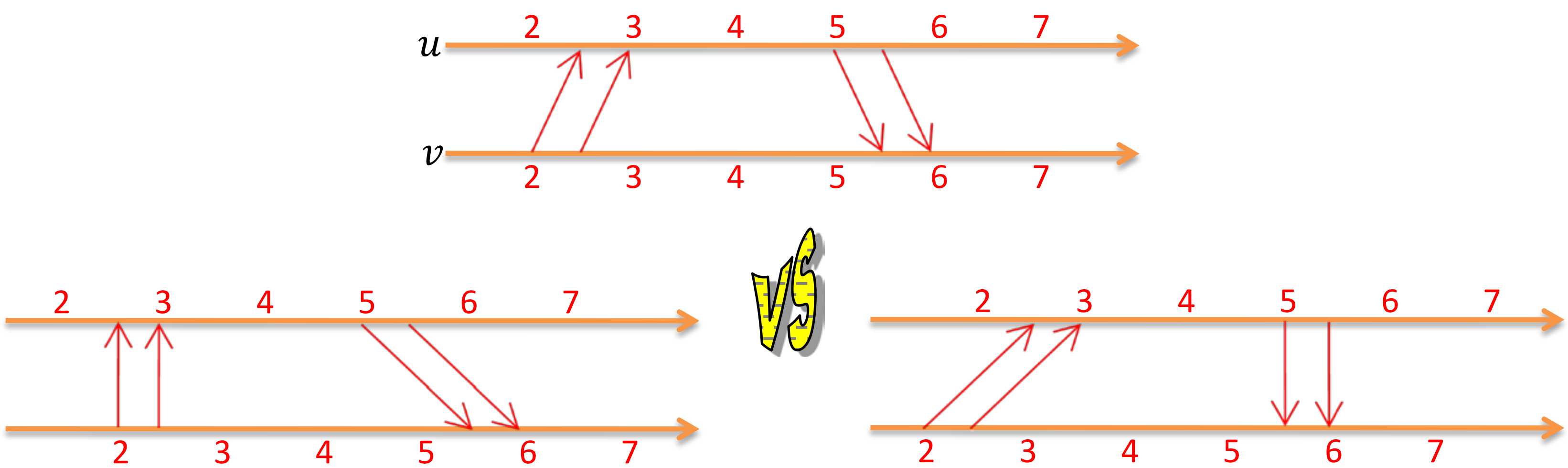
- We can make clocks continues, any $\beta > 2\rho \cdot \frac{1+\rho}{1-\rho}$ works
  – Intuition: If a node u knows of a larger clock value, it sets its logical clock rate to $\frac{\beta}{1+\rho} \cdot h_u(t)$ to catch up $\Longrightarrow$ see exercises!
- Global skew cannot be improved $\Longrightarrow$ see next slides!
- Local skew can be improved, however
  – straightforward, simple ideas don't work [Locher et al., 2006]
  – somewhat surprisingly, $O(1)$ local skew is not possible [Fan et al., 2004]

Global Skew Lower Bound

Theorem: The global skew guarantee of any clock synchronization algorithm is at least $D/2$ (where D is the diameter of the network).

How to Enforce Clock Skew?

- Make messages fast in one direction and slow in the other dir.
- This allows to "hide" a constant amount of skew per edge

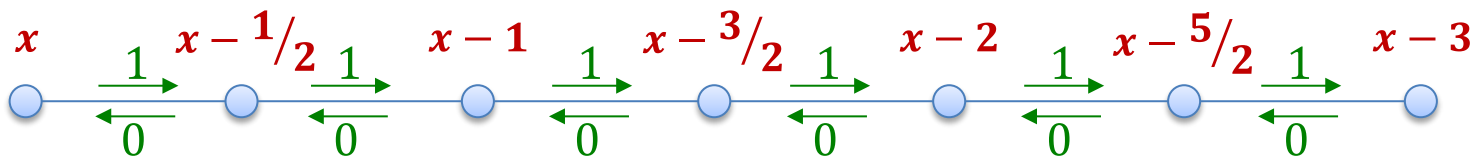


Global Skew Lower Bound

Theorem: The global skew guarantee of any clock synchronization algorithm is at least $D/2$ (where D is the diameter of the network).

Proof Idea:

- Assume that all hardware clocks run at rate 1 (no drift)
- Create two indistinguishable executions (causal shuffels):
  1. Initially: going from left two right, clock skew $-{}^{1}/_{2}$ between neighbors
     Message delays: left to right: 1, right to left: 0

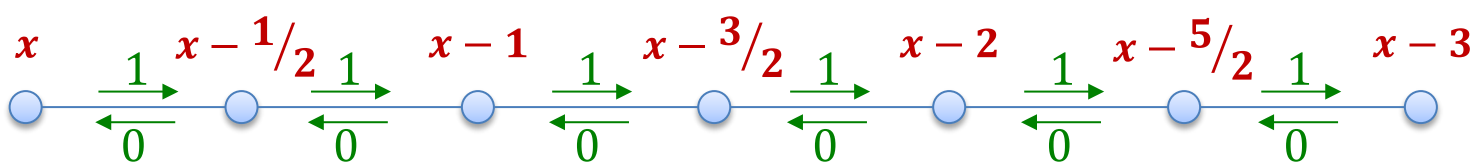


Global Skew Lower Bound

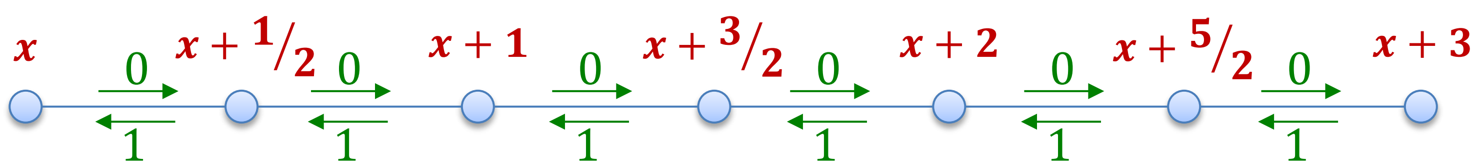
Theorem: The global skew guarantee of any clock synchronization algorithm is at least $D/2$ (where D is the diameter of the network).

Proof Idea:

- Create two indistinguishable executions (causal shuffels):
  1. Initially: going from left two right, clock skew $-{}^{1}/_{2}$ between neighbors
     Message delays: left to right: 1, right to left: 0



  2. Initially: going from left two right, clock skew $+{}^{1}/_{2}$ between neighbors
     Message delays: left to right: 0, right to left: 1

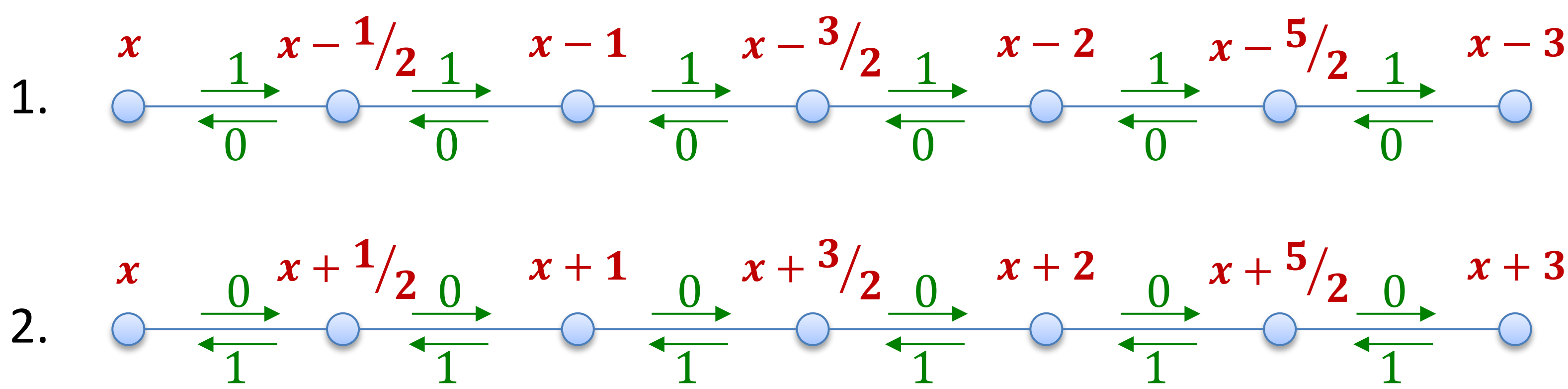


Global Skew Lower Bound

Theorem: The global skew guarantee of any clock synchronization algorithm is at least $D/2$ (where D is the diameter of the network).

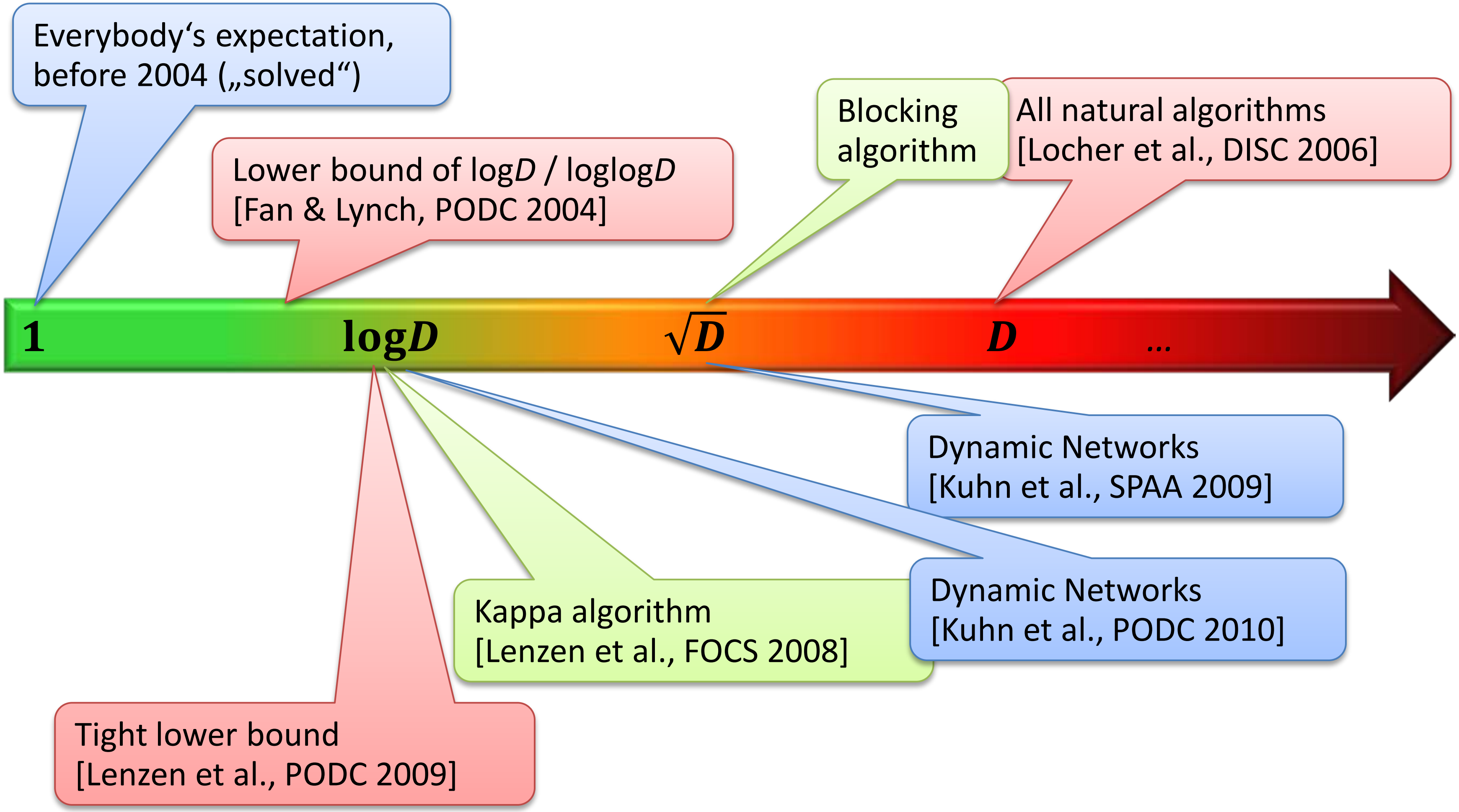
Proof Idea:

- Create two indistinguishable executions (causal shuffels):



- If in execution 1, $L_{v_R}(t) - L_{v_L}(t) = S$,
  in execution 2, we have $L_{v_R}(t) - L_{v_L}(t) = S + D$.

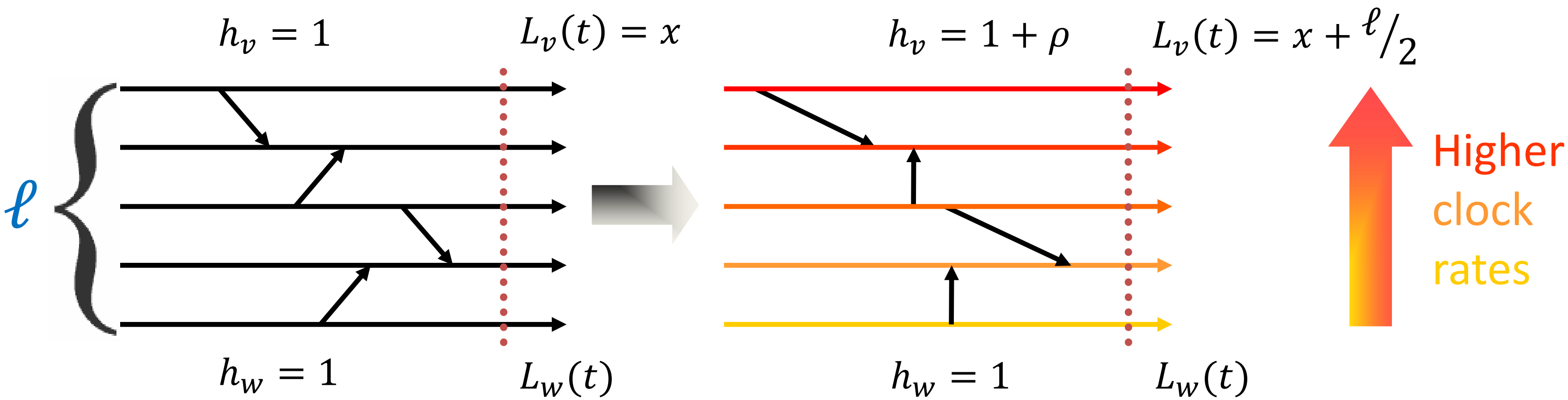
Local Skew: Overview of Results



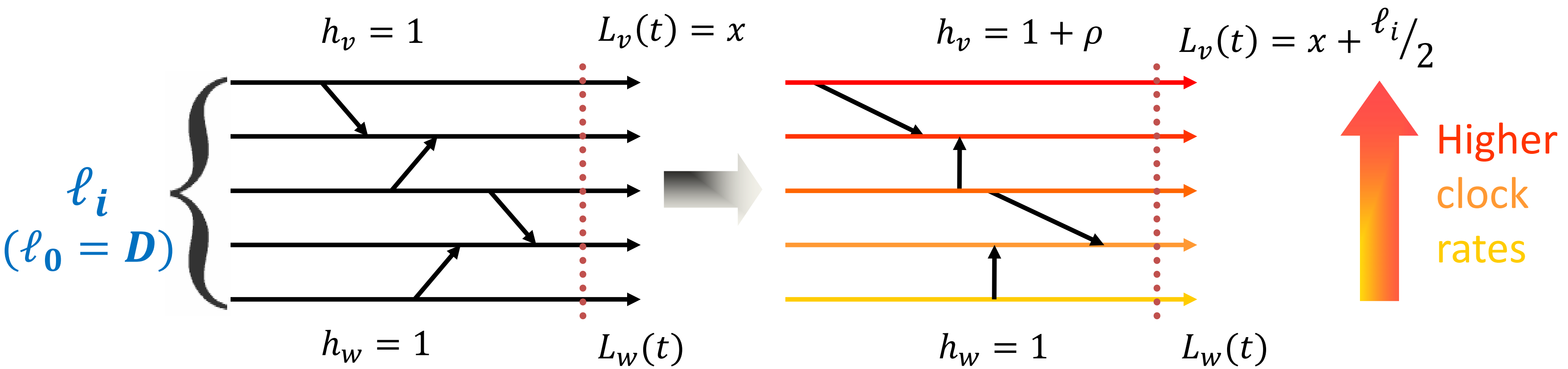
Local Skew: Lower Bound

Claim: For a path of length ℓ:

- Given a an execution of length T where all hardware clock rates are 1 and all message delays are $1/2$.
- Assume at the end of the execution, the clock skew between the endpoints of the path is s
- We can create an indistinguishable execution in which at the end, the clock skew is $s + \ell/2$.



Local Skew: Lower Bound



- Add ${}^{\ell_0}/_2$ skew in ${}^{\ell_0}/_{2\rho}$ time, messing with clock rates and messages
- Afterwards: Continue execution for ${}^{\ell_0}/_{4(\beta-1)}$ time (all $h_x = 1$)

  → Skew reduces by at most ${}^{\ell_0}/_4$ → at least ${}^{\ell_0}/_4$ skew remains

  → Consider a subpath of length $\ell_1 = \ell_0 \cdot {}^{\rho}/_{2(\beta-1)}$ with at least ${}^{\ell_1}/_4$ skew

  → Add ${}^{\ell_1}/_2$ skew in ${}^{\ell_1}/_{2\rho} = {}^{\ell_0}/_{4(\beta-1)}$ time → at least ${}^{3}/_4 \cdot \ell_1$ skew in subpath
- Repeat this trick (+½,-¼,+½,-¼,…) $\log_{2(\beta-1)/\epsilon} D$ times

Local Skew: Lower Bound

time $\frac{\ell_0}{2\rho}$

length $\ell_0 = D$, skew ≥ 0

play with hw clock rates / message delays

time $\frac{\ell_0}{4(\beta-1)}$

length $\ell_0 = D$, skew $\geq \ell_0/2$

all hw clock rates $=$ 1, all message delays $= {}^1/_2$

length $\ell_0 = D$, skew $\geq \ell_0/4$

time $\frac{\ell_1}{2\rho}$

subpath of length $\ell_1 = \ell_0 \cdot \frac{\rho}{2(\beta-1)}$, skew $\geq \ell_1/4$

play with hw clock rates / message delays

time $\frac{\ell_0}{4(\beta-1)}$

length ℓ_1, skew $\geq \ell_1/4 + \ell_1/2$

all hw clock rates $=$ 1, all message delays $= {}^1/_2$

length ℓ_1, skew $\geq \ell_1/4 + \ell_1/4$

subpath of length $\ell_2 = \ell_1 \cdot \frac{\rho}{2(\beta-1)}$, skew $\geq \ell_2/4 + \ell_2/4$

Local Skew: Lower Bound

Theorem: For every clock synch. algorithm, the skew between neighboring nodes can be $\boldsymbol{\Omega\left(\log_{(\beta-1)/\rho} D\right)}$.

Proof Idea:

- Given an exec. where for the last ${}^{\ell_i}/_{4(\beta-1)} = {}^{\ell_{i+1}}/_{2\rho}$ time units all $h_x = 1$ and all msg. delays are ${}^{1}/_{2}$ and witha subpath of length ℓ_i with skew $\geq i \cdot {}^{\ell_i}/_{4}$ (at end of exec.)
- Pick subpath of length $\ell_{i+1} = \ell_i \cdot \frac{\rho}{2(\beta-1)}$ with skew $\geq i \cdot {}^{\ell_{i+1}}/_{4}$
- Use last ${}^{\ell_{i+1}}/_{2\rho}$ time units to increase skew to $\geq i \cdot {}^{\ell_{i+1}}/_{4} + {}^{\ell_{i+1}}/_{2}$
- Add ${}^{\ell_{i+1}}/_{4(\beta-1)}$ time units with all $h_x = 1$ and msg. delays 1
- New skew is still $\geq (i+1) \cdot {}^{\ell_{i+1}}/_{4}$

Local Skew: Lower Bound

Theorem: For every clock synch. algorithm, the skew between neighboring nodes can be $\boldsymbol{\Omega\left(\log_{(\beta-1)/\rho} D\right)}$.

Proof Idea:

- For all $i = 0, 1, 2, \ldots$
- Create subpath of length ℓ_i with skew $\geq i \cdot {}^{\ell_i}/_4$

$$\ell_0 = D, \qquad \ell_i = \ell_{i-1} \cdot \frac{\rho}{2(\beta-1)} = D \cdot \left(\frac{\rho}{2(\beta-1)}\right)^i$$

- Number of iterations: $\Theta\left(\log_{(\beta-1)/\rho} D\right)$

Local Skew: Upper Bound

- Up to small constants, the $\Omega\left(\log_{(\beta-1)/\rho} D\right)$ lower bound can be matched with clock rates $\in [1, \beta]$ (highly non-trivial!)
- We get the following picture [Lenzen et al., PODC 2009]:

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

We can have both smooth and accurate clocks!

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

- In practice, we usually have $\frac{1}{\rho} \approx 10^4 > D$.

  In other words, our initial intuition of a constant local skew was not entirely wrong! ☺